

Network Processors and Co-Processors

Scott Thomas & Kurtis Kredo II
CSC 564, Winter 2003

February 10, 2003

Abstract

Recent advancements in network technologies have allowed for such high bandwidths that the cabling is no longer a limiting factor. The restricting factor now becomes the router hops in between the source and destination. This paper will discuss network processors and how they can be used to increase router throughput. General processor architectures will be covered and testing results will be shared. Then some commercial products will be explained as well as benchmarking suits used to test these processors to view the increased performance. Next, current areas of research and development will be presented, such as: scheduling, load balancing, and QoS applications.

The second part of the paper will cover network co-processors. These co-processors can be thought of as intelligent network interface cards (NICs). The goal of these NICs is to offload as much network related processing as possible from the host computer to the NIC. This allows for such things as multimedia compression, encryption/decryption, firewalling, and intrusion detection all to be performed on the NIC so that the host is free to perform other functionality. The paper will cover some specific examples of using a co-processor to do multimedia processing as well as security. Lastly, we will cover some implementations of co-processors found in academia today.

1. Introduction

Communication devices in current data communication networks are relied upon to do a great deal of work while performing under severe processing time constraints. Additionally, every day these devices are pushed to perform more operations as users request expanded functionality from the networks they are connected to. To satisfy these demands developers have made many improvements in the design of network devices and the associated technology. Large amounts of bandwidth have become available – although not fully utilized yet – in wired networks through the use of fiber optics. The latency through a network device was improved in the past by using application specific integrated circuits (ASICs) instead of general purpose computing technologies running communications software. These advances, however, created a condition where the network could not meet all of the needs placed upon it. ASICs provided an improvement in processing speed, and for a while kept up with the link speeds, but current link speeds can easily overload even the most advanced network devices. Another drawback of using ASICs is that it is impossible to adjust or add functionality to a device once it has been created, so networks become static entities that are very resistant to change. This is a problem for such areas as wireless networks where the protocols change fairly quickly (when the cost of replacing the infrastructure is considered). A relatively new technology, however, has appeared that may alleviate many of these problems. Network processors and co-processors are specialized hardware that combine the versatility of a general software implementation (through programming) with the speed of an ASIC (through specialized hardware that is optimized to handle network traffic).

This paper will focus on the development and current uses of network processors and co-processors. Section 2 discusses network processors including: a further discussion of the problems in current networks, an introduction to network processor architectures – both theoretical and current production chips, an overview of network processor benchmarks, and a review of current research topics from academia. Network co-processor applications, such as multimedia and security applications, and the prototypes CiNIC and MemNet, are covered in section 3. A conclusion closes the paper in section 4.

2. Network Processors

Problem

The problem with the world today is that the development of physical medium to transmit data has outgrown the physical hardware processing power. In short, the cabling is no longer the bottleneck in data communications. Light can now be used to transmit data from location to location. With the introduction of such protocols as Wave Division Multiplexing (WDM), networks can carry multiple, concurrent transmission channels and grow to deliver throughput rates that grow with the number of wavelengths used [1,2]. With current technologies, the total number of wavelengths that can be used is greater than forty, which allows for more than fifty terabytes of data per second to travel through a fiber optic cable. Now that we have the physical media that can move this much data, we need to improve the hops that the data must pass through on its journey. Unfortunately, these routers cannot be avoided and on long trips across the country or the world, data will pass through tens of routers.

Routers have three different areas that we can look to improve: throughput, latency, and application specific packet processing. Throughput is the total amount of incoming data the router can process at a specific time. The latency of a router is how long the router will hold the packet while it does the required processing. Lastly, application specific packet processing is the different jobs that routers are being given. These jobs include packet classification, packet forwarding, data encryption/decryption, and so on. We will now analyze each of these areas to determine which area is the problem in the routers present today.

First, in the area of bandwidth, there are many routers available that can handle multiple 10/100 base-T links. However, when considering the throughput of links such as an OC 192, the choices drop off dramatically. An OC 192 runs at speeds over ten gigabits per second. This is an exceptional amount of data for a standard router to handle and there are limited devices that can handle this. Cisco's top of the line router, the 12416 Internet router, can handle up to sixteen ten Gbps streams for a total combined bandwidth of 160 Gbps full duplex [3]. This is an awful lot of bandwidth, but there is still plenty of room to improve especially when we can top out at 50 terabits per second per stream.

The next area of router slow down involves the latency of a router. The time it takes a router to process a packet depends on multiple things. First, if the signal is in light, the packet must be converted into electrical signals, interpreted, processed, then returned to light and dropped back onto the fiber. If the signal is already in electrical encoding, the packet must be routed; involving a hash into a table and the packet is on its way. In an experiment performed on January 28, 2003, the average latency of a router between San Luis Obispo and Boston was calculated to be approximately one millisecond per router. One millisecond per router is an acceptable time, especially considering the fact that most sites on the Internet can be reached in ten to twenty hops. This shows there is little room for improvement, as straight IP forwarding is being performed at an acceptable rate.

Lastly, routers are being asked to perform more complex things than just IP forwarding. Routers at the edge of autonomous systems are now being asked to do all sorts of packet classification and filtering. MPLS is an emerging technology that places a heavy burden on ordinary routers. Other Quality of Service (QoS) protocols such as diffserv can bring routers to their knees. Some routers are used in the implementation of Virtual Private Networks (VPNs) which means that encryption must be performed inside these routers. Network Address Translation (NAT) is used on many routers to share an Internet connection with tens or hundreds of users. All of these things are in the process of being moved to the routers and will cause an overall slowdown of network performance if the routing hardware does not keep up.

Solution

Developers realized that something needed to be done in order to keep up with the increase in data flowing through the network. The first step in the direction of speeding up routers was to replace the bus that the NIC and the CPU were connected to with a switched fabric [4]. As the demand increased further, these software based routers were replaced with ASICs. However, ASICs turned out not to be flexible enough to adapt to the changing Internet, the production time was also too long to keep up with technology. Through the need for something with the speed of an ASIC, yet with the programmability of software emerged the network processor.

The network processor is a hybrid between a complete software solution and a pure ASIC solution. These processing units are capable of supporting the high I/O bandwidth that will be required of them. A key characteristic of networks that a network processor must be able to handle is the parallelism involved with network processing. A router will have multiple interfaces all with data streaming in and out requiring different operations. The network processor must be able to handle all of this data in parallel to keep up with the amount of data that is being transmitted across.

Processor Architectures

This section will discuss some of the performance testing that has been done on different processor architectures to determine the ideal choice for a network processor. The tests that have been done are representative of actual jobs a router would be expected to perform. These three tests are IP forwarding, MD5 checksum, and 3DES encryption. These three processes were tested over four different types of processors: Super-scaler (SS), fine-grained multithreaded (FGMT), single chip multiprocessor (CMP), and a simultaneous multithreaded processor (SMT) [5].

The first processor tested was a super-scaler processor. This processor is an out-of-order processor with a seven-stage pipeline to promote instruction level parallelism (ILP). The second type of processor tested was the fine-grained multithreaded processor. This processor is very similar to the super-scaler processor, however it added support for multiple hardware thread contexts. This allows different threads to take advantage of cycles that would have been ordinarily lost on a super-scaler by swapping a new thread into the slot. This allows for a larger throughput of instructions. The chip-multiprocessor is what can be thought of as a standard multiprocessor system. Each processor has its own execution pipeline, register files, fetch units, etc. The disadvantage of this processor is that the processors are simple and do not support ILP, and the resources cannot be shared between processors. The last processor included in the testing was a simultaneous multithreaded processor. The processor can be thought of as a combination of the second and third processor architectures all combined into one chip. This makes for one heck of a processor, but cannot truly be compared equally to the rest as the complexity and cost is much higher than its counterparts.

The results gathered in the research are not surprising. The CMP and the SMT outperformed the other two processor architectures. The CMP, having slightly less throughput, but much less complexity and the SMT having higher throughput at the expense of a very complex processor architecture.

Network Processor Benchmarks

Benchmarks are the most commonly used tools to compare processors; however network processors require special consideration for evaluation. Current benchmarks, such as SPEC [6], Whetstone, and Dhrystone, have been shown to be inaccurate for the evaluation of network processors [7,8]. The general purpose programs included in current common benchmarks do not represent the applications that will run on a network processor. For example, the SPEC benchmark includes (among others) a gcc compiler, a compression algorithm, an AI program, a JPEG compression algorithm, and a database program. While it is possible (or probable) a network processor will run a general compression algorithm and a JPEG compression or transcoding algorithm, most of the applications listed above have no relevance for a network processor, so any results provided by these benchmarks are questionable at best. It is possible that the instruction characteristics, memory access rates, and correct branch prediction rates are the same for applications in the general purpose benchmarks as those in applications for a network processor, but without testing it would appear unlikely to most. Two benchmarks that have been proposed specifically for network processors are CommBench [7] and NetBench [8].

CommBench and NetBench, created at Washington University in St. Louis and UCLA, respectively, were both created to provide accurate results for testing current network processors. Both benchmark suites include applications that are expected to be running on a variety of core and edge routers. Applications include (from both benchmarks): CRC calculation, table lookup, IP routing, scheduling, NAT, IP chains, Diffie-Helman, MD5 digest algorithm, IP fragmentation, TCP monitor, CAST encryption, Lempel-Ziv compression, Reed-Solomon forward error correction, and JPEG compression. It is important to note that both benchmarks use applications that operate over only the headers of a packet (routing and scheduling) and application that operate over the entire packet payload (compression, encryption, and digest algorithm). As routers are becoming more powerful and the core networks are increasing in speed, more

applications are being pushed to network edges. Because of this any network processor benchmark suite that did not include both types of applications may not accurately represent the computation load experienced by a router.

The creators of CommBench and NetBench have shown that router applications are different than the applications used in the general purpose benchmarks and even those used in MediaBench [9] (a general purpose multimedia and communication systems benchmark). In [7] the authors compared the CommBench benchmark suite to the SPEC general purpose benchmark suite and found that there were significant differences in: the program size (code and object), the number of instructions that are executed (for 90 and 99 percent of the execution time), the types of instructions executed most often, and the cache miss rates (for a given cache size). Similarly, the authors in [8] showed that the amount of instruction level parallelism, the accuracy of branch predictions, and the most common instructions are significantly different between NetBench and MediaBench. These two groups have shown that the common general purpose benchmarking suites produce invalid results when applied to a network processor.

Commercial Network Processors

With an understanding of the tools necessary to test network processor designs we can now look at the current state of processor design in the commercial market. After an introduction to network processor architecture we will examine two specific chips that are currently available: the Intel IXP2850 [10] and the IBM NP4GS3 [11,12]. Current network processors are composed of: a set of small, but fast, processors, called micro engines or pico-processors; a control processor; off chip memory interfaces; hardware acceleration blocks; and interface blocks for off chip access (such as through a PCI bus or a switched interface). The micro engines, between eight and sixteen per network processor, are the main functional blocks of a network processor and perform most of the computation, especially any processing that must be done at high line speeds. Micro engines are specialized, minimal multithreaded RISC processors that are optimized with hardware to be able to switch the thread of operation very quickly (within a few clock cycles). To provide a more general (but slower) computational base, network processors contain a control processor that may handle any processing that the micro engines are unable to handle, prepares the micro engines on boot, and handles many of the general system functions.

Current network processors provide interfaces to memory (such as DRAM or SRAM) due to the limited availability of chip space. These memories may hold the packet data while it is being processed, state information such as routing tables, or other data that the programmer desires. While the goal of a network processor is to provide flexibility, there are several computations that may be incorporated into hardware to improve the performance of the processor. These blocks may perform calculations that are standard, and so are unlikely to change, or calculations that are too computationally intensive to perform in software at high line speeds (i.e., encryption and authentication). Also provided on most network processors are interfaces to various other system resources such as: a PCI bus for connection to another general purpose processor, a media interface for connection to linecards, and a switch fabric interface for connection to other network processors within the same device.

The structural blocks are the same for most network processors, but there is still a great deal of flexibility (as will be shown) available in the system architecture and what blocks are provided. One of the main differences between the IXP2850 and the NP4GS3 is the organization of the micro engines; Intel uses a pipelined architecture and IBM uses a parallel architecture. For Intel's IXP2850 chip the micro engines are organized into a single stream of 16 micro engines where they each perform a simple operation on a packet before passing it to the next micro engine in the sequence. It is worth noting that Intel's chips may be programmed so the micro engines are grouped into an arbitrary number of smaller sequences or they may even operate independently. In contrast IBM organizes its NP4GS3 chip by grouping the micro engines into eight pairs with each pair of micro engines operating on a packet until the processing is completed. The choice of chip architecture also dictates that amount of memory and cache the micro engines will have. For example, the NP4GS3 has 64kB of instruction memory per micro engine (instead of the 32kB on the IXP2850) to allow for the longer and more complex processing. Another difference between these two chips is the amount of hardware acceleration they each provide. The IXP2850 operates at a very high frequency (1.6GHz) and has few hardware acceleration units while the NP4GS3 operates at 133MHz and has a large number of hardware functional blocks (such as a tree search engine, a scheduler, and a packet classifier). A final difference to mention is the ease of programming the two chips. The IBM chip trades generality for speed by having the additional hardware blocks performing many operations that

would otherwise have to be done in software and with specialized threads (only certain threads may perform some operations). On the other end, the IXP2850 may be harder to program since the programmer has much more control over the logical system organization and the lack of some hardware blocks. The programmer is also responsible for the decision of where to modularize code to divide among the micro engines in an Intel chip.

Current Research

Commercial applications can tell us the current state of network processors, but we must look to academia for future trends. Recent work includes implementations of routers using network processors and general purpose processors, analytical examination of the design of network processors, research in how scheduling within a network processor may occur, ways to balance the load across a set of network processors, and an example of how network processors may be used to provide QoS.

A group at BBN Technologies developed a router based on network processor designs [13]. Their work provided an example of what could be done if the flexibility of software were combined with architecture similar to those seen in current network processors. The router was built using a set of Alpha processors arranged as independent processing units and a single Alpha processor acting as an overseer. Each of the independent processors would process a packet and forward it as need, acting in an analogous way to the micro engines in the network processors described previously. Also similar to current network processors was the use of a designated chip as a control processor that would handle complex packets (such as control or signaling packets) and oversee system operation. The authors list several advances in their router: a switched backplane, the use of forwarders that existed separate from the linecards, knowledge of the complete routing table at the forwarders, and the implementation of a simple QoS. A switched backplane – as opposed to the then-common shared bus architecture – allowed the forwarders (the name given to the processors that processed the packets) to send data amongst themselves and the linecards very quickly. Having the forwarders separate from the linecards provides a layer of abstraction that allows the forwarders to be upgraded without altering any other system objects and allows the forwarders to handle traffic from any type of physical media (the router in this implementation was limited to IP traffic). Since their forwarders were general

purpose processors themselves they had a great deal of system resources that they could utilize, so the authors had each forwarder store the complete routing table on memory local to each forwarder. This decision would limit the latency associated with a cache miss when only some of the routing table was stored locally. The control processor handled updating the routing table caches for each forwarder. While their router provided a very basic QoS (limited to classifying packets based on header information and a simple outbound queuing based on the assignment), it was able to do this at operating speed. The work done at BBN Technologies to provide flexibility by using software controlled packet processing and using a distributed system of independent processors is easily seen in current designs.

Further architectural design and analysis of network processors was done at Washington University in St. Louis [14]. This group experimentally determined the optimum number of processors, instruction and data cache sizes, and number of I/O channels for a network processor design constrained by chip area. For each simulation the design under test ran three different workloads composed of header and packet processing applications (specifically, those from [7]) that would be used in a router. The design that had the highest number of instructions executed per second and fit within a particular chip area was determined to be the best. It is worth noting that many of the optimal values determined by this work (16 processors, 16kB of instruction and data cache, and two I/O channels) are very similar to what is seen in current chips.

Research similar to the above two groups has been done by a group at Princeton University [15,16,17,18]. Over many years of research this group has promoted a three tiered network processor architecture (similar to current system architectures), considered how to add functionality to a router, and has developed many operating routers based on their designs. The system architecture proposed by this group (first done in 1999) is that of a three layer system with the fastest, but most limited processing occurring on the bottom layer, a top layer consisting of a very general processor that does not operate as quickly as the lowest layer, and a middle layer that compromises on speed and generality. This architecture was designed so that most of the traffic would get processed at the lowest level and any traffic that could not be handled at that layer would get passed to the next level up. Similarly if a packet could not be handled at the middle layer it could be handed to the highest layer. One prototype built with this architecture

consisted of an Intel IXP1200 (an earlier version of the IXP2850 discussed earlier) and an Intel Pentium III. This router was tested to determine the limitations and capabilities of the three tiered architecture, the organization of packet processing across the three levels, and the possibility of adding functionality to the router dynamically. Much of the work concerning the architecture is applicable to future work since current chips are set up to work in a similar three tier structure.

Once it is determined where to divide the processing among the levels in the hierarchy and router is in place a decision must be made for each packet as it enters the router. Three common scheduling algorithms that may be used are: Round Robin, Link, and distributing packets to micro engines on a first come first serve basis. However, for a network processor application these scheduling algorithms are not optimal. Scheduling packet processing in a Round Robin fashion will cause some micro engines to go unused as the scheduler waits for processing to finish on a complex packet. Problems also occur if Link scheduling – queuing packets by assuming packet length determines processing time – is used since the processing time for a packet is very loosely related to the size of a packet. The best solution of these three is to use the micro engines as a resource pool and to give a packet to the next available micro engine. A subtle inefficiency with this algorithm occurs when packets with different processing requirements are given to a micro engine, which causes the micro engine's instruction cache to become invalid and forces it to reload the instructions for the new packet type. These problems were considered in [19] where the authors propose a predictive scheduler. For each application or packet type that a network processor will handle the predictive scheduler maintains the amount of time a packet will take to process, independent of the packet size, and the amount of time per byte the packet will take to process. The scheduler maintains two statistics per application since the processing time for some applications is unaffected by packet size (routing) while others are heavily dependent on it (encryption). After processing is finished the actual processing time is returned to the scheduler so that it may update its statistics and adapt to any changes in data flows.

A research area similar to scheduling is that of load balancing between a set of network processors. Two similar procedures were proposed in [20] and [21] that attempt to balance traffic between multiple network processors by aggregating traffic into flows and assigning a

flow to a network processor. There are minor differences between the two procedures so, except where stated otherwise, the following discussion applies to both. Incoming traffic to a set of network processors is first handled by a balancer (it may be implemented as part of a network processor) whose sole purpose is to assign traffic to flows and balance those flows among the set of processors. In order to handle traffic at high data rates the classification and assignment of packets must occur quickly, so the result of a hash function is used as a tag for that traffic flow. Inputs into the hash function are usually portions of the packet headers such as source and destination address, source and destination port, and protocol number. Since multiple inputs may have the same hash result traffic is automatically grouped into flows and the granularity of the flows may be controlled by adjusting the hash function. The hash results are kept in memory and are assigned to a processor so as to provide quick access through devices such as table lookup. The balancer must have some knowledge of the relative load experienced by each processor, and this is the main difference between the two proposed balancers. The authors in [20] proposed the use of queue lengths as a determination of processor load. Each processor had a queue of packets ready to be processed, that was accessible by the balancer, whose depth was related to the processor load. In [21] the authors had each processor explicitly report its status back to the balancer on a routine basis. Using either approach, once the balancer determines that the processors are unequally loaded it must readjust the flows. The balancer is able to do this quickly by simply adjusting the table that maps hash result to processor. There are two main advantages to a balancer with this design: it is very fast due to the use of a hash function and the ease at which flow adjustments may be made, and it does not require the balancer to maintain state information on any flow that passes through the set of network processors.

Another application that can be applied to network processors is a specific QoS protocol called differentiated services (diffserv). The design goal of diffserv is to support a range of network services that are differentiated on the basis of performance [22]. Packets are classified, using a variety of options, at edge routers thus making the majority of the trip simply based on the marking of the packet. Some advantages of diffserv include that as the classification is only done at the edge routers, it is scalable. Also, assuming everything obeys the rules, it will provide a better than best effort delivery service. The specifics of diffserv can be found in RFC 2475 [23].

Some tests have been done specifically on Intel's IXP1200 network processor with diffserv. The benchmarking performed shows that the processor can handle up to 1.8Gbps doing standard IP forwarding, but drops to a staggering 200Mbps when diffserv is turned on [24]. The research group determined that the drop in throughput was due to SRAM memory accesses being performed by the processor. The group alleviated this problem by implementing a different searching algorithm during the classification of packets. This transferred the bottleneck to the microengines. The solution for this is to upgrade to a higher model IXP network processor with better microengines.

3. Network Co-Processors

In a recent study performed at Rutgers University, an apache web server spends about 70% of execution time doing network processing [25]. This is time that the host CPU could have used to process data or requests from new clients. The idea behind network co-processors is to place more intelligence in the network interface card, thus network co-processors are sometimes referred to as intelligent network interface cards. The purpose of network co-processors is to offload some or all of the networking functionality off the host computer's CPU. In addition to relieving the host CPU of network processing, the addition of a CPU to the NIC allows the NIC to do processing on its own. Some applications that can take advantage of this include encryption/decryption, compression/decompression, and multimedia data processing.

One application of network co-processors that has been researched is the use of the co-processor to handle the computation associated with receiving multimedia data over a network. The authors in [26] built a system that pushed all of the multimedia and network processing for a video conferencing application onto a NIC. The co-processor would receive data from the network and move it directly into the memory of a frame buffer via DMA in order to limit the amount of host resources required to display the video. While the actual implementation this group was able to accomplish was limited (the co-processor was only able to move bitmap images from the network to memory) it showed the possibilities of this type of system. Future work the authors mentioned included pushing multimedia encoding and decoding onto the co-processor.

Security is another feature that may be placed onto a co-processor. The authors in [27] describe a system where the firewall is no longer centralized, but is placed on the co-processor at each host. One advantage with a host-based firewall is that there is no longer a single point of attack for hackers to focus on, and if a system is compromised the amount of damage a hacker can do is more limited in this approach than it is for a centralized approach. A disadvantage, however, is that administration becomes more difficult when the system may be spread across an entire complex or an even larger geographical area. To ease the administration of the firewall routines the authors proposed to have a security protocol handle the authentication and encryption for an administrator when a host's firewall needs to be adjusted or updated. A pleasant side effect of this authenticated firewall adjustment is that normal users will not be able to get to the firewall rules and accidentally cause a security hole.

Two different implementations of a network co-processor developed in academia will now be presented. The first is the Cal Poly intelligent Network Interface Card (CiNIC). This project stems originally from a grant from the 3com Corporation almost 5 years ago. The CiNIC is implemented currently using a Dell Pentium III 550 MHz as the host computer. The co-processor is implemented using a PCI-to-PCI non-transparent bridge with an ebsa-285 board as the controller over this secondary PCI bus. The StrongArm processor on the ebsa-285 board is the co-processor and accesses the NIC plugged into the secondary PCI bus. The host and co-processor communicate through a region of memory on the ebsa that is mapped into the host's memory space. This memory is divided into two halves, one for each direction of communication. The memory is managed through the use of a polling process on a bit associated with each half.

The software implementation has evolved drastically in the past few years. Originally, network operations were hijacked at the system call level and copied to the co-processor. However, this hit a dead end due to problems with the fork() and dup() system calls. The file descriptor / system call level was too high in the stack to try and hijack the function calls. We decided to move down into the stack deeper to the family level. This is the level where the kernel takes different actions depending on whether the user desires a UNIX, internet, or raw socket. We

decided to add our own type of socket called AF_CINIC. Therefore, when our module is installed, users can create AF_CINIC type sockets and our modules will handle the transmission of the calls to the co-processor. Once the call is transported to the co-processor, our software takes care of executing the call and returns the appropriate data back to the host.

Currently, many of the network functionality calls are implemented on the CiNIC. The CiNIC can now act in both roles of the infamous rcopy and server. The calls, socket(), connect(), bind(), listen(), accept(), send(), and receive() have all been implemented and tested. In order to support larger scale applications such as Netscape or ftp programs, a module needs to be developed on the host that will have these already compiled programs use the AF_CINIC socket type instead of the normal internet AF_INET socket.

Now that the CiNIC is working, there are many new projects available to be pursued. One project is the minimization of copies between the NIC and user space on the host. Currently, this process involves many copies: NIC to co-proc kernel space, co-proc kernel space to shared memory, shared memory to host kernel space, host kernel space to host user space. The desired result is a copy from the NIC to shared memory and then from shared memory to the host's user space memory. This would drastically reduce the latency currently in the system and would equal the number of copies currently in a standard computer with a standard NIC. Another project to be worked on is implementing an interrupt driven communication system. Currently each CPU is wasting all available cycles checking to see if data has been delivered to shared memory which is very inefficient. We envision using interrupts to signal the other processor that data is in the shared memory and needs to be serviced. A project that is just beginning as a master thesis is to implement encryption on the co-processor. Using an FPGA, additional hardware instructions can be programmed to speed up the time it takes to encrypt the data using the DES encryption algorithm.

MemNet [28] is a network co-processor that goes by the name of a TCP server. The MemNet architecture is similar to the CiNIC, however is implemented in a different method. The same goal to offload the TCP/IP processing kept them motivated in their work. However, instead of having hardware connected, they implemented a secondary server to handle the TCP/IP

communications. The two computers communicate via a SAN. They are further in development than the CiNIC and have already performed tests on their platform. The results agree with what would be expected. There is added latency due to the copy of the data across the SAN, but the amount of data (throughput) increases by a few hundred replies per second.

4. Conclusion

There are many open areas of research involving network processors and co-processors. Network processors have evolved quickly and their applications are being seen daily. The type of processor to use in the network processor has been an area of research. As was shown in the paper, something maybe more important than the actual processor used, is finding a way to speed up the memory accesses. Another area of research is implementing certain instructions in their own processors. For example, the IBM PowerNP uses a separate processor to do functions such as checksum evaluation and string copying. In the future, more and more functions are going to be demanded of routers. These include NAT, QoS, and VPN support. These operations will need to be performed at near line speed in order to keep the latency at or below one millisecond, where it is now.

We saw two different designs of network co-processors. The CiNIC uses a shared memory approach to minimize the time of data transmission between the host and the co-processor. Rutgers' MemNet uses a TCP server type idea where a whole server is dedicated to handling the TCP/IP processing. The two computers are connected via a SAN. This allows multiple computers to all use the same TCP server to perform their network processing. There are many applications that can also be implemented on these network processors. We saw examples such as multimedia data processing and distributed firewalls. Areas of new research include implementing encryption / decryption, and special checksum evaluation operations.

References

- [1] G. N. Rouskas and M. H. Ammar, "Multi-destination communication over single-hop lightwave WDM networks," in the Proceedings of IEEE INFOCOM '94, Toronto, Canada, pp. 1520-1527, June 1994.
- [2] G. N. Rouskas and M. H. Ammar, "Minimizing delay and packet loss in single-hop lightwave WDM networks using TDM schedules." in the Proceedings of ICC '95, pages 1267-1271, June 1995.
- [3] Cisco Systems, Data sheet on the 12416 series Internet router. Available online at http://www.cisco.com/warp/public/cc/pd/rt/12000/12416/prodlit/itro_ds.pdf
- [4] R. Haas, C. Jeffries, et al, "Creating Advanced Functions on Network Processors: Experience and Perspectives," IBM Research Report RZ-3460, November 2002.
- [5] P. Crowley, M. Fiuczynski, J.-L. Baer, and B. Bershad, "Characterizing Processor Architectures for Programmable Network Interfaces." In the Proceedings of the 2000 International Conference on Supercomputing, Santa Fe, NM, May 2000.
- [6] Standard Performance Evaluation Corporation [1995]. SPEC CPU95 - Version 1.10, August 21, 1995.
- [7] T. Wolf and M. Franklin. "CommBench - A Telecommunication Benchmark for Network Processors," in the Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, pp. 154-162. April 2000.
- [8] G. Memik, W. Mangione-Smith, and W. Hu, "NetBench: A Benchmarking Suite for Network Processors." in the Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD). Nov. 2001.
- [9] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems." in the Proceedings of the International Symposium on Microarchitecture, IEEE Micro-30, 1997.
- [10] *Intel IXP2850 Network Processor*. Product Brief, Intel Corporation, 2002. <http://developer.intel.com/design/network/products/npfamily/>
- [11] *PowerNP NP4GS3 Network Processor*. IBM Corporation, 2000. <http://ibm.com/chips>.
- [12] M. Peyravian and J. Calvignac. "Fundamental Architectural Considerations for Network Processors." Uncorrected Proof, To Appear in *Computer Networks*, 2003.
- [13] C. Partridge, P. Carvey, et al. "A Fifty Gigabit per Second IP Router." *IEEE/ACM Transactions on Networking*, pp 237-248, June 98.
- [14] T. Wolf, M. Franklin, and E. Spitznagel, "Design Tradeoffs for Embedded Network Processors." Washington University Technical Report WUCS-00-24. July 2000.

- [15] L. Peterson, S. Karlin, and K. Li, "OS Support for General-Purpose Routers." IEEE HotOS Workshop. March 1999.
- [16] T. Spalink, S. Karlin, and L. Peterson, "Evaluating Network Processors in IP Forwarding." Princeton University Technical Report TR-626-00. November 2000.
- [17] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb, "Building a Robust Software-Based Router Using Network Processors." in the Proceedings of the 18th ACM Symposium on Operating Principles. pp. 216-229, Oct. 2001.
- [18] S. Karlin and L. Peterson, "VERA: An Extensible Router Architecture." *Computer Networks*, Vol. 38, Issue 3 February 2002.
- [19] T. Wolf, P. Pappu, and M. Franklin, "Predictive Scheduling of Network Processors." Uncorrected Proof, To Appear in *Computer Networks*. 2003.
- [20] G. Dittmann and A. Herkersdorf, "Network Processor Load Balancing for High-Speed Links." in the Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems. pp. 727--735. July 2002.
- [21] L. Kencl and J. Le Boudec, "Adaptive Load Sharing for Network Processors." in the Proceedings of INFOCOM 2002. June 2002.
- [22] H. Smith, "Slides for Computer Networks II," Fall, 2002.
- [23] S. Blake, et al. "An Architecture for Differentiated Services," RFC 2475, December 1998.
- [24] Y. Lin, Y. Lin, S. Yang, and Y. Lin, "DiffServ over Network Processors: Implementation and Evaluation," in the Proceedings of the 10th Symposium of High Performance Interconnects Hot Interconnects. (HotI '02).
- [25] K. Banerjee, "TCP Servers: A TCP/IP Offloading Architecture for Internet Servers, Using Memory-Mapped Communication." Graduate school thesis – Rutgers University. October 2002.
- [26] M. Fiuczynski, R. Martin, et al. "On Using Intelligent Network Interface Cards to Support Multimedia Applications." in the Proceedings of 8th International Symposium on Network and Operating Systems Support for Digital Audio and Video, pp. 95-98. July 1998.
- [27] D. Friedman and D. Nagle, "Building Firewalls with Intelligent Network Interface Cards." Carnegie Mellon University Technical Report CMU-CS-00-173. May 2001.
- [28] M. Rangarajan, K. Banerjee, J. Yeo and L. Iftode. "MemNet: Efficient Offloading of TCP/IP Processing Using Memory-Mapped Communication." Rutgers University Technical Report, DCS-TR-485, May 2002.