Computer Sciences Department

Intelligent Routing Using Network Processors: Guiding Design Through Analysis

Madhu Sudanan Seshadri
John Bent
Tevfik Kosar

Technical Report #1480

April 2003

UNIVERSITY OF WISCONSIN MADISON

Intelligent Routing using Network Processors: Guiding Design through Analysis

Madhu Sudanan Seshadri, John Bent, Tevfik Kosar*

University of Wisconsin, 1210 West Dayton Street, Madison, WI 53706
{madhu, johnbent, kosart}@cs.wisc.edu

October 3, 2002

Abstract

The explosive growth of Internet traffic and the increasing complexity of the functions performed by network nodes have given rise to a new breed of programmable microprocessors called network processors. A major hurdle in designing these processors is a lack of understanding of their workload characteristics. Dearth of literature tackling specific architectural issues has also resulted in an unclear understanding of the design space of these processors.

We provide a comprehensive survey of ideas and features employed by current network processor designs. We also explore the contribution of academia in this area. Lastly, as an initial exploration of this large design space, we analyze packet traces from different sources and show that caching can be gainfully employed to enhance performance for routing table lookup functions in network processors.

Key words: *Network Processors, routers, routing table lookup, packet trace analysis, anonymized routing tables.*

1 Introduction

As networks start accomodating complex applications like VoIP, firewall services and virtual private networks (VPNs), the amount of packet processing that a network switch needs to do increases. In simple packet forwarding scenarios, a router parses the header to find the destination address and sends the packet in the right direction as rapidly as possible. But when the network starts to provide value added services like the ones described above, the router needs to analyze headers more thoroughly without compromising the speed of the network. This puts a lot of strain on the processing capabilities of the router.

*Contact Author. Tel:+1(608)262-5945

Current routers address this problem by combining a general purpose processor with application specific integrated circuits (ASICs) that implement proprietary algorithms for providing the additional functionality. But the ASICs are typically large, are expensive($200-$400) and often take as long as 18 months [21] to develop. A high-end router employing about a dozen of these has a long time to market and is very expensive.

An additional challenge is that once deployed it is very difficult to modify or enhance the functionality to keep up with changing needs and evolving protocols. Network processors address these problems by providing the flexibility that ASICs lack while keeping costs down. By building these processors with programmable off-the-shelf components, the time to market can be reduced substantially [21]. These advantages have contributed to the emergence of the network processor market as the fastest growing segment of the microprocessor industry [10].

While there is a consensus in the industry about the advantages of a network processor, there is substantial variance in the microarchitecture and chip organization employed by each company. We compare the strategies followed by different companies in some important aspects of processor design with the objective of identifying common trends and exploring the design space for these processors.

2 Design space exploration

The tasks performed by a router can be separated into two planes: the control plane and the data plane. The control plane tasks, which are managed by a general purpose processor, perform functions such as executing routing table updates, flow management, traffic shaping, gathering statistics and managing various engines in a distributed architecture. General purpose processors are well suited for these tasks by providing the flexibility to address chang-

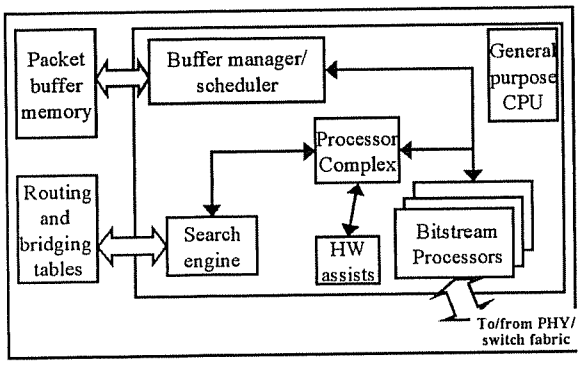

Figure 1: **Block diagram of a typical Network Processor.**

ing needs of the protocols that implement these functions [15].

The data plane is involved in processing the bits within a packet and is often referred to as "wire speed" processing. Specialized packet engines perform the necessary functions in this plane.

2.1 System Architecture

The operation of a typical network processor is depicted by the block diagram as shown in Figure 1. Packets move in and out of the network processor through the PHY switch interface. The bitstream processors receive the serial stream of packet data and extract the information needed to process the packet, such as the IP source/destination address, type of service bits or TCP source/destination port numbers [6]. The packet payload is then dispatched to the buffer manager unit which writes the packet into the packet buffer memory. The extracted control information is passed on to the processor complex which constitutes the programmable part of the network processor(NP).

Under program control, the processor, if needed, extracts additional information from the packet and submits the relevant part to the search engine, which looks up the next-hop IP address, classifies the packet or does a virtual circuit/path identifier lookup if the packet is recognized as an asynchronous transfer mode (ATM) cell using the routing and bridging tables and appropriately designed hardware assists. Based on the results returned, the processor instructs the scheduler to determine the appropriate departure time of the packet. The necessary modifications to the packet header are performed when the packet is transmitted to the bitstream processor.

Instruction	Functionality
FIND_BSET	find first bit set in any 16 bit register field
HASHx_64	perform 64 bit hash
CTX_ARB	swap contents and wake on event

Table 1: **Intel's IXP ISA additions.**

2.2 Processing Power

An OC-48 link operates at a speed of 2.5 Gbps. To sustain this rate, 6 million packets have to be processed per second. This translates into a throughput requirement of 2.5 billion instructions/second for simple packet forwarding. With OC-768 speeds (40 Gbps) this scales to a whopping 40 billion instructions/second [6]

2.3 System Configuration

Network processors are available as standalone products such as the IBM's Rainier [15] and Intel's IXP 1200 [21] and are also available as co-processors to a conventional RISC microprocessor. An example of the latter is Sitera's IQ2000 [13] which is designed to work with a conventional microprocessor (usually a MIPS or PowerPC chip). Another example is Agere's PayloadPlus Chipset [14] which is a set of 3 chips assisting a a PowerPC processor.

2.4 Instruction Set Architecture

Keeping in mind the special requirements of packet processing, vendors have augmented the traditional RISC ISA with special instructions. Some samples from Intel's IXP 1200 [21] are shown in Table 2.4. augmented the traditional RISC ISA with special instructions. Some samples from Intel's IXP 1200 [21] are shown in Table 1.

Frank Engel et.al. [3] recommend a combined "compute and jump" instruction. Their idea is to replace the conventional

```
subi r1 1h
bnz r1 #11
```

sequence of instructions with a single

```
subbi r1 1h #11
```

instruction. They claim that a compute operation followed by a conditional branch occurs frequently enough in the workloads they studied to warrant this change. The advantages are compact code size and with single cycle execution, substantial speedups in loop intensive code. This is also supported by Wolf et.al [22], who recommend augmenting memory-register transfer instructions with memory-port transfer instructions.

Many other ISA additions have been proposed as well. Paul Bromley et. al [4] recommend use of special instructions for network-specific computation, field extraction, byte alignment, boolean computations and conditional opcodes (to reduce branches). Wolf et.al [22] recommend use of MMX-VLIW type of instructions to efficiently implement data streaming functions. SiByte's SB-1 NPU core has a "paired-single" SIMD instruction which operates on a pair of 32-bit values stored in a 64-bit register [12]. Vendors have also included table lookup instructions and customized branch instructions suited to analyzing and modifying packet headers [16].

2.5 Microarchitecture

The microarchitecture of network processors is marked by emphasis on streaming data throughput and heavy use of architectural parallelism. Designs span all the way from non-pipelined such as IBM's Rainier and CPort's C-5 to aggressively pipelined like Agere's Payload Plus, SiByte's SB-1250, EZChip'sNP-1 and Cisco's Toaster2 [15].

Chip multiprocessing with hardware multithreading seems to be a popular technique to exploit the huge thread-level parallelism available in packet processing workloads. Sitera's IQ2000 has four 32-bit scalar cores with 64-bit memory interfaces, so each core can perform a double-word load or store in a single clock cycle. Each core has 5 identical register files (32 registers, 32 bits wide, triple-ported). This arrangement allows each core to run five concurrent threads of execution with fast context switching, because they don't have to save their register states in memory [13].

SiByte's SB-1250 contains multiple integrated SB-1 cores. Each core is a four-issue in-order superscalar design with 6 functional units [12]. IBM's Rainier integrates 16 picoprocessors with 1 PowerPC core in a single chip. Each picoprocessor has support for 2 hardware threads. Two picoprocessors are packed into a dyadic protocol processor unit and share a tree search engine and internal memory [15].

Intel's IXP1200 integrates 6 RISC microengines with a StrongArm core. Each microengine has its own physical register file, a 32-bit ALU with a basic 5-stage pipeline, a single cycle shifter, an instruction-control store (4K of SRAM), a microengine controller and 4 program counters - one for each thread that a microengine can execute in parallel [21].

Lexra's LX8000 features a cluster of MIPS-like cores with support for fast context switching and chip multiprocessing [10]. Cisco's Toaster 2 contains 16 XMC (express microcontroller) cores. Each core is an enhanced ARM-7 LIW design executing a 64-bit instruction word (that consists of 2 RISC instructions) per cycle [16]. EZChip's NP-1 has 4 different processor cores, each of which is optimized for performing a specific portion of the packet procesing task. These 4 cores are arranged in a 4-stage superpipeline, meaning that each stage has many copies of the appropriate core. The NP-1 has 64 cores altogether [8]

XStream Logic's network processor is based on the dynamic multistreaming (DMS) technique (also known as simultaneous multithreading). The processor core can support eight threads. Each thread has its own instruction queue and register file. The core is divided into 2 clusters of 4 threads each. Every clock cycle, each cluster can issue up to 16 instructions-four from each thread-and four of the 16 are selected and dispatched to one of the four functional units in that core for execution [8]. The DMS core has 9 pipe stages and features a MIPS like ISA [9].

Wolf et.al. recommend using VLIW designs [22]. A few companies have taken this approach. Cisco's Toaster 2 as discussed earlier is a LIW design. Motorola's NetDSP has a SC-140 core jointly developed by Motorola and Lucent. The core is a six-issue VLIW DSP with 16 function units, including four multiply-accumulate (MAC) units that can execute 1.2 billion operations at the chip's target frequency of 300 MHz [11].

A study by Patrick et.al [5] comparing SMT, CMP, FGMT (fine grained multithreading) and superscalar designs for network processors seems to favour SMT. But the results are based on a limited set of programs that are not standardized and may not be representative of the entire workload of a network processor. Also, in our opinion, the configurations used for each of the above designs may have been favourable to SMT.

2.6 Branch Prediction

Branch prediction for network processors doesn't appear to have received as much attention in the literature as other areas. While some processors, like SiByte's SB-1 core, have a sophisticated branch prediction mechanism [12], other designs, such as Sitera's IQ2000, feature only static branch prediction with the branch being predicted always not taken [13]. As of now, the effect of branch prediction on network processor performance seems to be unclear.

2.7 Memory Bandwidth

Estimates of required memory bandwidth to offchip DRAMs depends very much on the amount of memory available on-chip and the way packets are processed. Thus while IBM estimates that it requires 40Gb/s bandwidth for a 10Gbps throughput [6], EZChip says that it requires 500 Gb/s for the same throughput [8]. But either way, current DRAM solutions are far from delivering the expected

bandwidth required for these processors. Techniques like pipelined acccess to memory and very highly interleaved memory structures can be used to alleviate this memory bottleneck. In the future, multiple parallel DRAMs and on-chip ultrawide DRAMs may come to be employed.

2.8 Benchmarks

Benchmarking network processors is a difficult process because of the wide variety of applications that these processors handle and the wide categories into which these products fit in. Vendors quote number of packets processed per second but this doesn't take into the account the amount of processing done per packet. As a consequence, this metric is not very useful [13]. As of now, there is no standard benchmark available. But a Network Processor Benchmark Forum has been formed with the goal of developing benchmark suites in all relevant application domains within the NP space [17].

Parallel efforts in academia have resulted in the publication of a paper titled "Commbench" which describes a suite of applications intended to serve as a benchmark for network processors [22]. The suite is divided into a set of 4 header processing applications and 4 packet processing applications. The authors present results of simulations made on an UltraSparc processor. Their conclusion is that the benchmarks have better instruction cache miss rates compared to SPEC (3.8% vs 8.3% for a 1 KB cache, 0.1% vs 0.5% for a 32KB cache). They also find that a 16 KB data cache is sufficient to limit miss rates to less than 1%.

The same group has also published a paper describing a benchmark titled "NetBench" [23]. They compare the instruction mix of Netbench with SPEC and show that Netbench executes a lot more arithmetic instructions and fewer load/store instructions compared to SPEC as shown in Figure 2. They claim that only 67 instructions out of the 242 available in the ISA were used by the benchmarks and just 30 instructions were responsible for over 95% of all executed instructions. They argue in favor of a relatively simple RISC core with a fairly limited instruction set. Their stand seems to be vindicated as judged by the design of network processors in the industry.

3 From the general to the specific

As we have described it, the design space of network processors is shown to be very large indeed. As such, it is beyond the scope of this paper to thoroughly examine all areas of this design space. Rather we hope to encourage others to join us in this effort of both exploring the general design space and examining each of its areas in some depth.

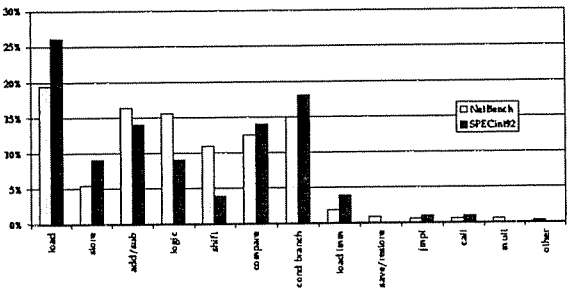

Figure 2: **NetBench Vs SpecINT92**

As a first step in this direction, we have choosen to focus on the routing table lookup functionality of the network processor. Since routing decisions must be made for every packet moving through the system, we consider this a very important area in which to focus our efforts.

4 Methodology

To examine the design space of the routing table lookup in a network processor, we have collected and analyzed real network packet traces. We are aware of the boot-strapping problem that we face in this endeavor. Predicting the future from the past is an NP-complete problem. Computer designers perhaps face this problem more directly than designers of other systems given the fast moving pace of the computer industry. For example, no-one could have predicted twenty years ago the current popularity of the internet.

Even more difficult than predicting growth is attempting to predict the innovations by which enterprising users can use new technologies in novel and unexpected ways. This is the dilemma we have faced in attempting to explore the design space of network processors by examining contemporary packet traces. However, in the absence of packet traces from the future, this is an unsolvable problem. We can only hope to design systems that answer contemporary concerns and attempt to make them sufficiently robust and flexible enough to adapt as users take advantage of the improved technologies.

4.1 Packet trace acquisition

Many organizations, like the ones listed in Table 2, make packet traces publicly available. These traces are collected by the National Laboratory for Applied Network Research (NLANR) [1] and made available at their website. However, due to privacy considerations both the destination and source IP addresses of the packets traces have been anonymized.

Router	Organization
AIX	NASA-Ames interconnect
COS	Colorado State University
SDC	SDSC commodity connection
MEM	University of Memphis
BWY	Columbia University (Broadway)
NCA	National Center for Atmospheric Research

Table 2: **Organizations providing traces to NLANR.**

Although some analyses are still possible with these anonymized traces, evaluating routing table lookups is more difficult for two reasons. The first is that the trace files cannot be run against a routing table because anonymized routing tables are not available. The second is that it is impossible to know which machine addresses exist within the same physical network.

In addition to doing some preliminary analysis of the publicly available anonymized traces, we have initiated discussions with local network administrators here at the University of Wisconsin. With their help, we are devising new schemes that allow the consistent anonymization of both routing tables and network packet traces. It is our belief that the failure of NLANR to devise a similar scheme has hampered current network research. We hope that our work in this area will convince NLANR to follow suit.

4.1.1 Historical background

However, consistently anonymizing network routing tables and packet traces requires complicated software algorithms. Historically, 32 bit addressed networks have been divided into three categories, Class A, B and C networks. This distinction refers to the length of the network mask that allows routing tables to aggregate collections of machines into logical networks. Without this aggregation, routing tables would need entries for every assigned IP address on the internet.

These classes however have led to an inefficient allocation of IP addresses that threaten to exhaust the available 32 bit address space. The reason for this is the lack of flexibility in the system. For example, the smallest possible assignable network, Class C has 256 IP addresses; when a class C network was assigned to a network of only two machines, 253 IP addresses were wasted. This problem gets exponentially worse for each larger network class: the smallest Class B network wastes more than sixty-five thousand addresses and the smallest Class A network wastes almost seventeen million addresses.

In response to this inefficient allocation, Classless Interdomain Routing [7] was introduced in 1993. CIDR allows networks to be of arbitrarily sized between one and seventeen million as long as they are multiples of two.

4.1.2 Anonymizing CIDR tables

With the old Class A, B and C networks, it would be trivial to anonymize routing tables and packet traces consistently. The reason for this is that the familiar "dot" notation of IP addresses (x.y.w.z) corresponds perfectly with the network mask boundaries. Each of the four segments of the address can be viewed as a unique 8 bit identifier. All networks have masks of 8, 16 or 24 bits and therefore do not violate any of the "dot" boundaries.

The simplest algorithm to anonymize pre-CIDR network routing tables and packets would be to assign each integer between 0 and 255 a different unique value also between 0 and 255. The routing tables and packet traces could be read and each segment of each IP address could be replaced with the random mappings. Privacy would be protected and network identities would be preserved.

However, CIDR destroys the conventional "dot" notation which has been retained only for its convenience to the human user. Network mask lengths can fall anywhere within the 32 bit address. CIDR tables need much more complicated algorithms and data structures to be anonymized. Additionally, the encryption created by the anonymization scheme must be strong enough to prevent any decodings.

A naive implementation would merely start at a random offset within the 32 bit address space and assign the next available, properly aligned[1] network to each routing table entry. But this scheme can exhaust the IP address space before all routing table entries can be assigned. This means that any anonymizing scheme must employ some sort of first-fit or best-fit allocation strategy to prevent exhausting the IP space.

4.1.3 A modest proposal

What we have done in order to most efficiently allocate anonymized networks to routing table entries is read the entries into a STL [20] multi-map container sorted descendingly by length of the network mask. Encryption is provided by randomly swapping networks of the same size. The networks are anonymized by iterating through the sorted, randomized networks and assigning each the next available, aligned network. The starting point is irrelevant in this case; for the sake of simplicity, we are currently starting from zero. An example of this anonymization is shown in Table 3.

With this encoding scheme, CIDR routing tables and packet traces can now be consistently anonymized. By us-

[1] Each network must be assigned a starting IP address such that the entire network shares the common netmask prefix. Although this sounds complicated, in practice it can be enforced simply by assigning each network a 32 bit starting address that is evenly divisible by the size of the network.

Routing table network	Anonymized network
4.24.146.76/30	0.0.0.0/30
4.24.145.36/30	0.0.0.4/30
4.24.147.8/30	0.0.0.8/30
4.24.144.60/30	0.0.0.12/30
4.0.0.0/24	0.0.1.0/24
12.1.83.0/24	0.0.2.0/24
12.1.248.0/24	0.0.3.0/24
12.1.245.0/24	0.0.4.0/24
3.0.0.0/24	0.0.5.0/24
12.2.41.0/24	0.0.6.0/24
12.2.99.0/24	0.0.7.0/24
12.2.7.0/24	0.0.8.0/24
12.2.6.0/24	0.0.9.0/24
12.2.97.0/24	0.0.10.0/24
12.0.252.0/23	0.0.12.0/23
12.2.94.0/23	0.0.14.0/23
4.43.240.0/22	0.0.16.0/22
12.2.88.0/22	0.0.20.0/22
24.141.16.0/20	0.0.32.0/20
24.141.64.0/19	0.0.64.0/19
24.141.32.0/19	0.0.96.0/19
24.141.0.0/17	0.0.128.0/17
24.141.0.0/16	0.1.0.0/16
47.16.0.0/15	0.2.0.0/15
47.8.0.0/14	0.4.0.0/14

Table 3: **Anonymized CIDR routing table entries.** Entries in the routing table are sorted descendingly by network mask length. Networks of the same network size are then randomly shuffled. Each network is then assigned the next available and properly aligned addressed starting from any 32 bit offset (shown here as zero).

ing patricia tries [2], longest prefix matching quickly finds the appropriate routing table entry for every IP address in the packet trace. Our algorithm stores each routing table entry in a patricia trie structure along with its anonymized network value and a hash table. The hash table is used to provide each seen IP address with the next available address within the anonymized network.

Each packet trace file can then be read, filtered through the anonymization data structures and written back to disk. In this way, we have developed a novel scheme to allow researchers the ability to run valid trace files against valid routing tables without violating the privacy of the owners of the trace files.

It should be noted that the randomization of the machine id's is therefore based on the order by which they were read from the trace files. Although this is weaker encryption than that used to anonymize the network addresses, it is sufficient given the strong encryption of the network addresses. Additionally, this scheme is designed to operate on multiple packet traces using the same anonymization mapping. Therefore, the order in which the traces files are read is randomized and reverse engineering of the machine addresses remains difficult even assuming that the network address could be compromised.

Router	AIX	MEM
Trace File Size	360 MB	180 KB
Number of Packets	8,200,000	4250
Number of Distinct IP's	126,000	85
Throughput	40 MB/sec	15 KB/sec

Table 4: Sample Trace information from two different routers

4.1.4 Physical requirements

Using network statistics [18] collected by the DoIT[2]organization, we have formed an idea about the hardware required to snoop network routers and collect packet traces. These expectations have been further validated by an analysis of the publicly available traces from NLANR as discussed in Section 5 and shown in Table 4.

In our opinion, the speed of the trace traffic is easily handled by current hardware. However, the sheer magnitude of the traces can be somewhat overwhelming. For example, the DoIT routers transfer approximately 200 megabits per second of network traffic. This is handled easily with a gigabit NIC. This traffic consists of an average of 40,000 packets per second. Depending on the packet header information being collected, upto 50 bytes may be written to disk for each packet. This amounts to a bandwidth of 2 MB/s which is easily matched by today's commodity disks. However, these 2 MB/s add up very quickly, filling 1 GB every eight minutes, a commodity disk with 80 GB capacity in 10 hours and a terabyte in less than six days.

4.1.5 Current status

Although complicated, the anonymization algorithms have been implemented. And although the physical requirements of packet collection are easily manageable, we have still found the acquisition of packet traces to be surprisingly difficult. Initially, political considerations of privacy were a stumbling block, especially due to our somewhat unique demand for accompanying routing tables. However, our novel anonymization software seems to have dissipated these political concerns.

[2]Department of Information Technology at University of Wisconsin-Madison

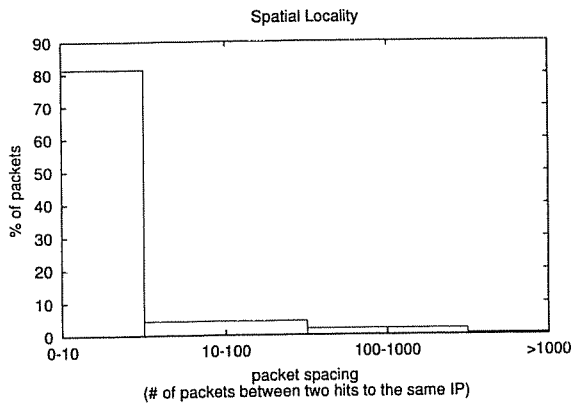

Figure 3: **Spatial locality at a single router.** This histogram shows the spatial locality of packet references as recorded at SDSC on April 31, 2001. Note that fully 80% of all packets were destined for an IP address which had been previously seen within the last ten packets.

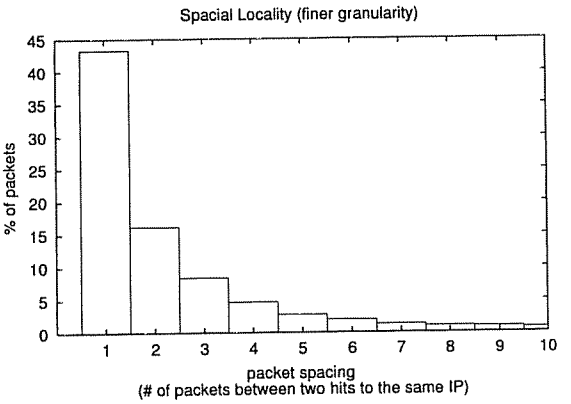

Figure 4: **Finer granularity of spatial locality at a single router.** This histogram is the same data as plotted in Figure 3 but is zoomed in on the x-axis. Note that almost half of the packets were destined for the same IP address as the preceding packet.

4.2 Packet trace analysis software

We have developed software to analyze the packet traces and measure the spatial and temporal locality of references, average packet lengths, and router throughputs. The analysis software basically consists of three parts: parsing the input files, interpretation of the acquired data and drawing the corresponding graphs.

The NLANR trace files come in many different binary file formats. Although there are some publicly available scripts which allow conversions from the different file formats into simple ASCII representations, we have chosen to do our own parsing of the binary files. Our first reason to do this was the fact that there are storage and time overheads involved in doing conversions as a pre-process stage to the analysis. Secondly, the perl scripts are not very consistent and produce different ascii formats for the different binary representations, which would require us to either modify the perl scripts or add functions to parse each of their different ASCII output formats. Finally, the perl scripts do not always read all the fields out of the packet traces. These factors motivated us to do our own binary parsing of the trace files and hide its complexity from the rest of our analysis code. This is abstracted in the code by using virtual, object-oriented `read` methods.

We used a fast hash-table based implementation to store and search the parsed data in memory. This implementation has improved the speed of our analysis tool by more than an order of magnitude. We choose the simple linked list initially due to its ease of implementation. While debugging our program we read only small trace files; it was only when our code was finished that we attempted to read a large trace file and found that our implementation was prohibitively slow. At this point, we swapped in the faster data structure and are now analyzing packet traces quickly enough so that it is hidden by the I/O latency of moving the trace from disk.

Generation of the charts is done by our analysis tool at runtime by redirecting the output of the analysis to gnuplot. Again, this was done out of consideration for both storage and time.

5 Experimental results

The analysis results showed that the traces from different routers have extensive differences in basic characteristics such as the router throughput and the number of distinct IP addresses to which packets passing through that router are forwarded. Although the traces were collected for the same intervals of time from all of the routers, the trace file sizes differ by more than three orders of magnitude in the most extreme cases. Table 4 compares two sample routers in terms of throughput and number of distinct IP addresses hit and shows these extremes.

Despite the differences in the loads and usages of different routers, analysis of the traces showed that all network traffic behaves similarly in terms of spatial and temporal locality and average packet lengths. For most of the routers, about 80% of the packets are forwarded to an IP address which had been previously hit within the last 10 packets. And more interestingly, about 40% of the packets are forwarded to the same IP address as the very previous packet. We believe that these results stress the importance of caching in routers, and show that even minimal caching of the routing table could result in a hit ratio of more than

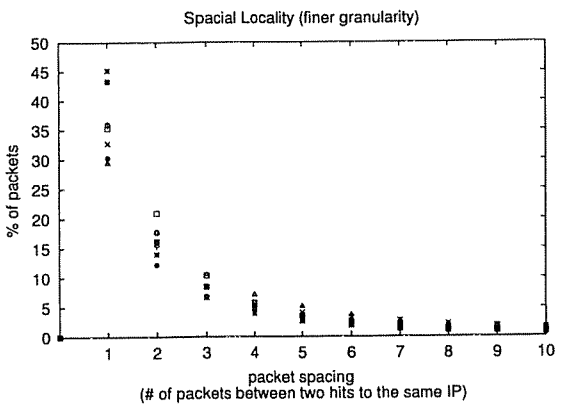

Figure 5: **Scatter plot of spatial locality.** This scatter plot parallels Figure 3 and shows that spatial locality measurements are consistent across different traces collected from different routers.

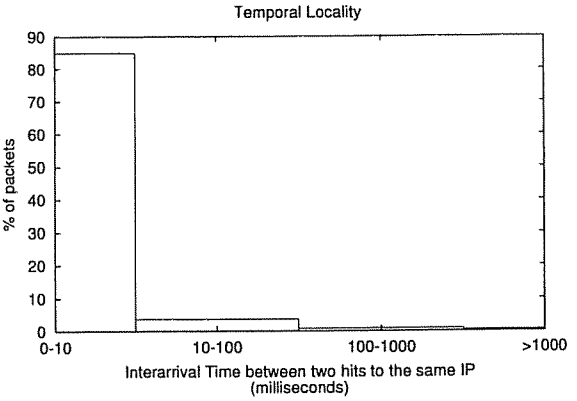

Figure 6: **Temporal locality at a single router**. This histogram measures the temporal locality from a trace file collected at SDSC on April 31, 2001. Similar to the spatial locality measurements, this histogram shows the high frequency in which destination addresses are reseen in the packet trace.

80%. Figure 3 shows the spatial locality information for the router at SDSC on April 31, 2001, and Figure 4 shows a finer granularity chart for the same data. Our analysis showed that these results are also consistent for almost all traces from different routers on different days as shown in the scatter plot in Figure 5.

Analysis on temporal locality is very parallel to the spatial locality and also confirms the importance of caching of the routing table. More than 80% of the packets are forwarded to an IP address which had been previously hit within the last ten milliseconds. Figure 6 shows the temporal locality information for the SDSC router on April 31, 2001.

Finally, the last metric which our analyses measured was the average packet length. Figure 7 shows that more than 60% of the packets from one trace are smaller than 100 bytes in size. The scatter plot in Figure 8 shows that while this is mostly consistent across different routers that there is in some cases a fair amount of large packets as well.

This is particularly salient given recent proposals from Juniper [19] that are stretching the definition of the routing RFC's. Amid loud cries of protest from rival Cisco, Juniper has begun optimizing throughput by reordering its internal flow of packets. This strategy may result in larger numbers of packets being buffered at the router and this is even more viable given our observations that most network packets are very small sized.

6 Future work

The IP addresses from the anonymized packet traces can be looked up against routing tables from several different network access points to see if the reference patterns change substantially from one point in the network to another. This can have important implications for the design of routing table caches.

However packet forwarding is only one of many functions performed by a network processor. The collected packet traces can conceivably be used as input data sets to a simulator that can generate code depending on the type of function that needs to be performed on a packet. The instruction traces thus obtained can then be used to study network processors more effectively.

7 Conclusion

Network processors are a new and exciting offshoot of conventional microprocessors. With the increasing complexity of network applications, they are rapidly gaining importance and have acquired a niche of their own in the microprocessor market. However due to their recent appearance, the workloads for these processors and the architectural requirements are not understood clearly and are being debated fiercely. We have taken a first step towards understanding the design space of these processors by doing a survey of current design strategies. We have also attempted to make a contribution of our own by collecting and analyzing packet traces.

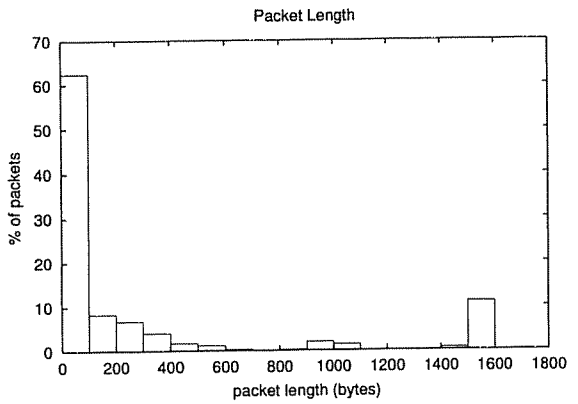

Figure 7: **Packet lengths at a single router.** This histogram measure the length of packets collected at SDSC on April 31, 2001. While most packets are smaller than 100 bytes, there are a significant number which approach 1500 bytes, the physical ethernet limit.

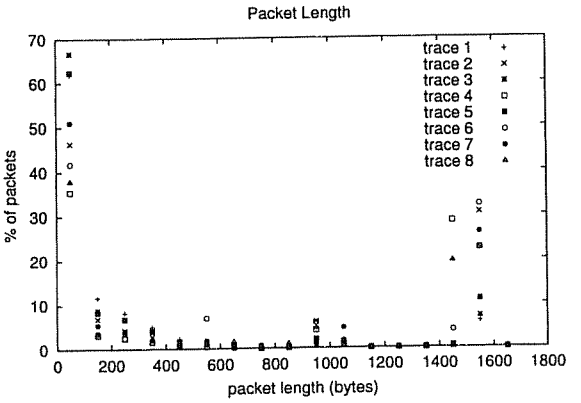

Figure 8: **Scatter plot of packet lengths.** This scatter plot reveals that the measurement of packet length as seen in Figure 7 is consistent across multiple traces from multiple routers. While most packets are small, there are a non-neglible number of large packets as well.

References

[1] National laboratory for network analysis (NLANR). http://moat.nlanr.net/.

[2] T. Chiueh and P. Pradhan. High performance ip routing table lookup using cpu caching. *IEEE Infocomm*, 1999.

[3] Frank Engel et.al. A New Network Processor Architecture for High-Speed Communications. *IEEE Workshop on Signal Processing Systems*, page 543, 1999.

[4] P. Bromley et.al. Network Processors: A Perspective on Market Requirements, Processor Architectures and Embedded S/W Tools. *Design Automation and Test*, pages 420–427, 2001.

[5] Patrick Crowley et.al. Characterizing Processor Architectures for Programmable Network Interfaces. *International Conference on Supercomputing*, May 2000.

[6] W.Bux et.al. Technologies and Building Blocks for Fast Packet Forwarding. *IEEE Communications Magazine*, page 70, January 2001.

[7] S. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (CIDR): an address assignment and aggregation. RFC 1519, September 1993.

[8] Linda Geppert. The New Chips on the Block. *IEEE Spectrum*, pages 66–68, January 2001.

[9] P. Glaskowsky. Networking Gets XStream. *Microprocessor Report*, November 2000.

[10] P. Glaskowsky. Network Processors Multiply. *Microprocessor Report*, page 37, January 2001.

[11] T. Halfhill. First StarCore DSP Targets Networking. *Microprocessor Report*, page 18, 1999.

[12] T. Halfhill. SiByte Reveals 64-Bit Core for NPUs. *Microprocessor Report*, 2000.

[13] T. Halfhill. Sitera Samples Its First NPU. *Microprocessor Report*, page 53, May 2000.

[14] K. Krewell. Agere's Pipelined Dream Chip. *Microprocessor Report*, pages 34–36, June 2000.

[15] K. Krewell. Rainier Leads PowerNP Family. *Microprocessor Report*, page 10, January 2001.

[16] L.Gwennap. Cisco Rolls Its Own NPU. *Microprocessor Report*, pages 19–22, November 2000.

[17] Dr.Adolfo Nemirovsky. Towards Characterizing Network Processors: Needs and Challenges. *White Paper*, November 2000.

[18] David Plonka. Wisconsin network performance statistics. http://wwwstats.net.wisc.edu/.

[19] Chuck Semeria. Internet processor II ASIC: Rate-limiting and traffic-policing features. White paper, Juniper Networks, September 2000.

[20] A. A. Stepanov and M. Lee. The Standard Template Library. Technical Report X3J16/94-0095, WG21/N0482, 1994.

[21] T.Halfhill. Intel Network Processor Targets Routers. *Microprocessor Report*, page 1, September 1999.

[22] T. Wolf and M. Franklin. Commbench - A Telecommunications Benchmark for Network Processors. *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 154–162, April 2000.

[23] T. Wolf and J. Turner. Design Issues for High Performance Active Routers. *Washington University, Tech. Report*, June 1999.