Performance, Scalability, and Flexibility in the RAW Network Router

by

Anthony M. DeGangi

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSSETTS INSTITUTE OF TECHNOLOGY
August 2004

Author...
Department of Electrical Engineering and Computer Science
August 23, 2004

Certified by...
Umar Saif
Research Scientist
Thesis Supervisor

Accepted by...
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Performance, Scalability, and Flexibility in the RAW Network Router

by

Anthony M. DeGangi

Submitted to the
Department of Electrical Engineering and Computer Science

August 23, 2004

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Conventional high speed internet routers are built using custom designed microprocessors, dubbed network processors, to efficiently handle the task of packet routing. While capable of meeting the performance demanded of them, these custom network processors generally lack the flexibility to incorporate new features and do not scale well beyond that for which they were designed. Furthermore, they tend to suffer from long and costly development cycles, since each new generation must be redesigned to support new features and fabricated anew in hardware. This thesis presents a new design for a network processor, one implemented entirely in software, on a tiled, general purpose microprocessor. The network processor is implemented on the Raw microprocessor, a general purpose microchip developed by the Computer Architecture Group at MIT. The Raw chip consists of sixteen identical processing tiles arranged in a four by four matrix and connected by four inter-tile communication networks; the Raw chip is designed to be able to scale up merely by adding more tiles to the matrix. By taking advantage of the parallelism inherent in the task of packet forwarding on this inherently parallel microprocessor, the Raw network processor is able to achieve performance that matches or exceeds that of commercially available custom designed network processors. At the same time, it maintains the flexibility to incorporate new features since it is implemented entirely in software, as well as the scalability to handle more ports by simply adding more tiles to the microprocessor.

Thesis Supervisor: Anant Agarwal
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Umar Saif
Title: Research Scientist

Acknowledgments

I would like to thank Anant Agarwal for the opportunity to work on this project. I would also like to thank Umar Saif for his invaluable input and ideas throughout the design process of this project.

I would also like to thank my colleagues in the Computer Architecture Group, especially Michael Taylor and David Wentzlaff for helping me learn the Raw development environment as well as debug many, many bugs. I would also like to thank James Anderson for his work on this router project, and a special thanks to Eugene Weinstein for introducing me to this group.

Last but not least, I would like to thank my parents, for allowing me to study at MIT and for their love and support.

Table of Contents

Introduction 10
  1.1 Packet Routing Overview 10
  1.2 Forwarding Tasks 11
  1.3 Network Processor Overview 12
  1.4 Case for new design 14
  1.5 Raw Overview 16
The Raw Network Processor 19
  2.1 Overview 19
  2.2 Stage Descriptions 20
    2.2.1 Stage 1: Table Lookup 20
    2.2.2 Stage 2: Compute Header Checksum 22
    2.2.3 Stage 3: Decrement TTL, Update Header, and Send to Output Queue or Discard 22
    2.2.4 Stage 4: Queue Header for Output Linecard 23
    2.2.5 Network Usage 24
    2.2.6 External Devices 24
  2.3 Version Descriptions 25
    2.3.1 Version I 25
    2.3.2 Version II 26
    2.3.3 Version III 28
    2.3.4 Version IV 28
  2.4 Scalability 29
  2.5 Buffering 30
Performance Evaluation 33
  3.1 Implementation 33
  3.2 Experimental Methodology 33
  3.3 Forwarding Rate 35
  3.4 Throughput 36
  3.5 Latency 38
  3.6 Scalability Performance Measurements 40
Suggestions for Design Modifications and Conclusion 44
  4.1 Suggestions for Architectural Design Modifications 44
  4.2 Conclusion 45
Bibliography 46

List of Figures

Figure 1-1. IPv4 Packet Header 12
Figure 1-2. Intel IXP2400 Block Diagram 13
Figure 1-3. Intel's Recommend Use for the IXP2400 13
Figure 1-4. The Raw Microprocessor 17
Figure 2-1. The Raw Network Processor – Stage Layout 20
Figure 2-2. The Raw Network Processor – Network Usage 27
Figure 3-1. Raw Network Processor Forwarding Rates 36
Figure 3-2. Raw Network Processor Throughput Rates 37
Figure 3-3. Intel IXP2400 Latency. 39
Figure 3-4. Forwarding Rate of Version III, comparing the 2 and 4 linecard versions on 64-byte packets 41
Figure 3-5. Forwarding Rate of Version III, comparing the 2 and 4 linecard versions on 1500-byte packets 41
Figure 3-6. Forwarding Rate of Version III compared with the entirely dynamic network based Version III 42
Figure 3-7. Throughput of Version III compared with the entirely dynamic network based Version III 42
Figure 3-8. Forwarding Rate of all versions, using captured internet 43

List of Tables

Table 1-1. Features Comparison Matrix..16
Table 3-1. MLFFR and associated Latency..39
Table 3-2. Raw Network Processor Task Times...39

Chapter 1

Introduction

Packet routing is a task of fundamental importance to the operation of the internet. Routers serve as gateways between lines of communications, ensuring that packets make the appropriate hops on their way to their final destinations. Presently IP routers are built using custom designed forwarding engines hardwired in microchips, called network processors. These chips achieve high performance, but are expensive to produce and must be redesigned if protocols change. This thesis outlines the design and implementation of a network processor entirely in software using the general purpose Raw microprocessor to demonstrate that programmable parallel microchips can achieve similar performance to custom designed ASICs. The network processor implements the forwarding of IPv4 packets as specified in RFC 1812 [2]. An IPv4 router was chosen because it represents a very difficult protocol to route, as IPv4 packets exhibit both non-deterministic size and non-deterministic arrival times. However, the design can be easily modified to handle many other routing protocols. The Raw microprocessor is a general purpose chip that has a unique parallel design particularly well suited for packet forwarding. This parallel architecture is exploited to create a design that is both flexible and scalable, and at the same time delivers superior performance.

1.1 Packet Routing Overview

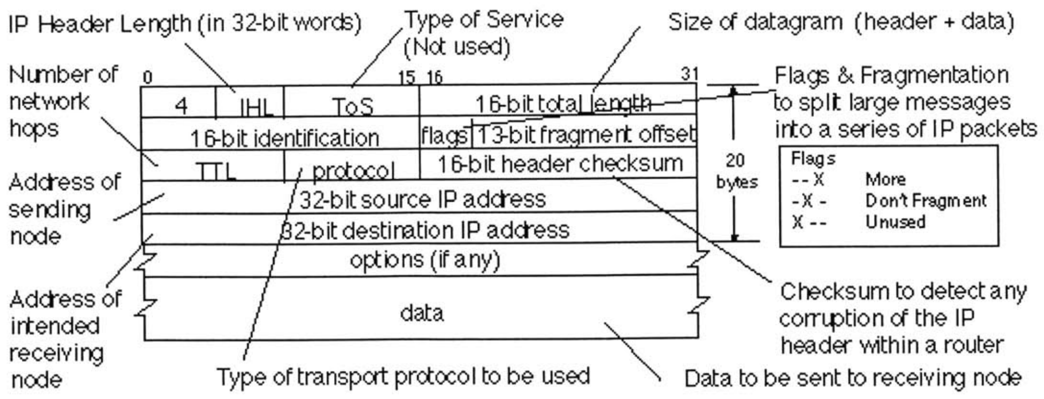
Internet packet routers are highly optimized computer systems whose sole purpose is to route packets along their way to their final destination. Since the time it takes for a packet to reach its final destination is constrained by the speed of the intermediate forwarding engines, fast and efficient routers are of utmost importance to the speedup of the internet.

When a packet arrives at the router it is either destined for the router itself or must be forwarded along the way to its final destination. The vast majority of packets arriving at the router merely need to be checked and forwarded along to the next link on the way to their final destination. The processing necessary for this task is trivially small compared to that necessary for router management, yet must be handled extremely quickly to maximize the performance of the router. These packets are passed to the data-plane of the router, which is responsible for the fast and efficient forwarding of packets. This thesis is concerned with the operation of the data-plane of the router.

In contrast, packets destined for the router itself contain routing information, which the router uses to build its forwarding table, which determines where and how to forward incoming packets. These packets arrive infrequently compared with most traffic through the router and take significantly more time to process since they contain information for complex routing protocols such as BGP and OSPF. As such these packets are removed from the streams of traffic that must merely be forwarded and sent to a control-plane. This control-plane processes these packets and is responsible for the management of the router. This aspect of the router is beyond the scope of this thesis. For more information on incorporating the control-plane into the Raw network processor design see [1].

1.2 Forwarding Tasks

When a packet arrives at the router it must be processed before being forwarded to the next hop along the path to its destination. An IP packet consists of both a header and payload. The header contains the information used by routers to select the path of the packet, while the payload contains the information used at the packet's final destination. The payload is typically not processed by the router. Packet headers may vary from 20 to 64 bytes and packets may be up to 65,536 bytes. Figure 1-1 shows a diagram of an IP version 4 packet header.



IP Header Length (in 32-bit words)
Type of Service (Not used)
Size of datagram (header + data)
Number of network hops
0
15 16
31
4
IHL
ToS
16-bit total length
16-bit identification
flags
13-bit fragment offset
TTL
protocol
16-bit header checksum
32-bit source IP address
32-bit destination IP address
options (if any)
data
Flags & Fragmentation to split large messages into a series of IP packets
20 bytes
Flags
-- X More
-X - Don't Fragment
X -- Unused
Address of sending node
Address of intended receiving node
Checksum to detect any corruption of the IP header within a router
Type of transport protocol to be used
Data to be sent to receiving node

Figure 1-1. IPv4 Packet Header.

The forwarding engine of the router must perform the following tasks:

Validation	– check the version information, time to live (TTL), and checksum
Lookup	– look up the output port that the pack will be sent out onto its final destination; the output port is indexed by the packets destination address
Update	– decrement TTL and update checksum
Forwarding	– queue the packet in the linecard of the output port

1.3 Network Processor Overview

To handle the task of packet forwarding efficiently, semiconductor manufacturers build network processors as custom designed ASICs specially suited to the tasks involved in packet routing. These network processors then serve as the core processing and forwarding units inside network routers. Network processors afford some degree of programmability; however, their underlying hardware design greatly constrains how they may be programmed. These network processors serve in contrast to pure ASIC based designs, in which everything is done solely in hardware, so they are not programmable.

Network processors are typically built around a memory-oriented, bus-based design. To be able to handle varying numbers of linecards running at different speeds, standard network processors tend to employ special concentrator units to aggregate all incoming packet streams into a single pool in shared memory. The computing units of the network processor can then

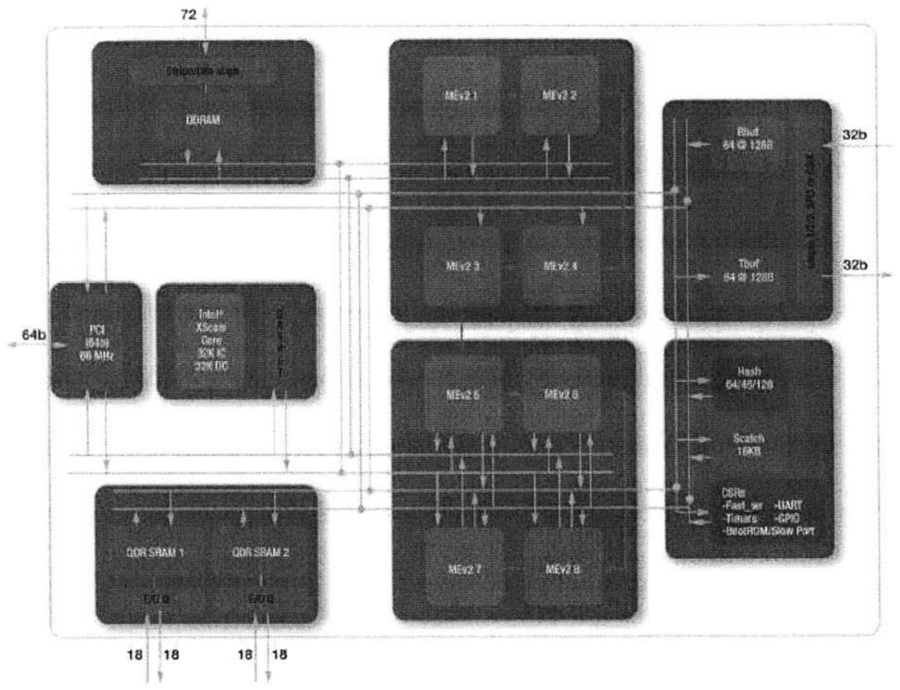


Figure 1-2. Intel IXP2400 Block Diagram [5]

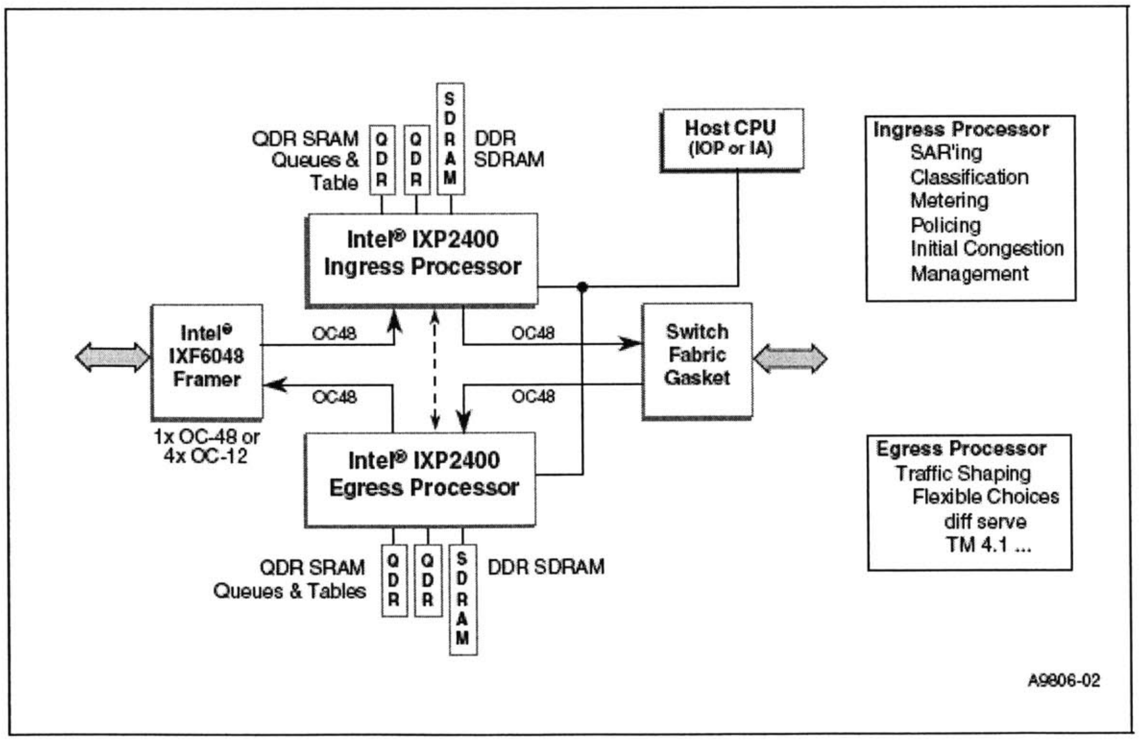


Figure 1-3. Intel's Recommend Use for the IXP2400 [4].

process the packets as if they were from a single stream. For example, as can be seen in Figure 1-2, Intel's IXP2400, its second generation network processor, uses this approach. Linecards are connected to a unit which aggregates the incoming packets before sending them to the receive unit of the IXP2400. Once inside the chip, packets are processed by eight multi-threaded microengines, each with their own small amount of local memory and state. To maintain low latency connections between the different functional units of the processor, the different units are grouped together and connected directly by point-to-point busses; the DRAM interface is one group, as are the SRAM, PCI, and transmit/receive interfaces. To exploit the parallelism inherent in packet processing, the eight micro-engines are also grouped into two groups of four. These eight micro-engines are chained together using what Intel calls Hyper Task Chaining, so that the individual engines can efficiently exchange state information between them. In this design, packets are stored in the off-chip, shared DRAM while being processed on the chip; the processing micro-engines then communicate with the DRAM over the dedicated point-to-point bus. Figure 1-3 highlights Intel's recommended system design for a gigabit Ethernet router using two IXP2400 chips. Interesting to note is that the task of switching the packets to their appropriate output ports is left to an external switching fabric. Since packets are stored aggregated in shared memory, both when entering and exiting the chip, an external device is required to send them along their way to the appropriate output port after processing.

1.4 Case for new design

As can be seen from the Intel IXP2400 example, typical network processors employ a very specialized design suited for packet routing. While this approach is capable of yielding the performance necessary, these specialized silicon chips have long and costly development cycles which must completely redesigned from generation to generation. Intel's IXP1200, for instance, is a redesign of its first generation IXP1200. While both are based on the same ideas, microengines for processing, grouping of functional units, and a memory-oriented architecture, the second generation chip required a redesign of the internals of these features as well as changing the floorplan for the chip to accommodate them. Furthermore, the second generation chip required fabricating an entirely new microprocessor, a task that takes months and millions of dollars.

In contrast, this thesis proposes using a general-purpose, tiled microprocessor, the Raw microprocessor, to implement a network processor entirely in software using a stream-oriented, rather than memory-oriented, design. Using this approach, the Raw network processor is able to both meet the performance demanded of it as well as be flexible enough to handle changes in requirements, since the processor is implemented entirely in software. Since it uses a general purpose chip, development time and cost are greatly reduced, as subsequent generations need only require updating the software rather than fabricating new chips. Using a tiled architecture with stream-oriented conventions enables the network processor to take advantage of the inherently parallel tasks involved in packet routing, while still running on a general purpose chip. Furthermore, since the tiled architecture already has built in capabilities for streaming data between different tiles, the chip can be used both as a network processor, to process incoming packets, and as a switching fabric, a significant improvement over traditional network processors.

Since the Raw microprocessor fully exposes the pin resources of the chip to the programmer, we were able to experiment with a number of different buffering schemes. The exposed pin resources in conjunction with directly addressable on-chip networks provide flexibility not found with traditional network processors, in which the programmer is constrained by the design choices made by the chip manufacturer. For such processors, the programmer has no freedom to change attributes such as the buffering discipline, these choices are already

	Flexibility	Scalability	Performance
ASIC	No	No	Yes
Network Processor	Limited	Limited	Yes
General Purpose CPU	Yes	Yes	No
Raw	Yes	Yes	Yes

Table 1-1. Features Comparison Matrix.

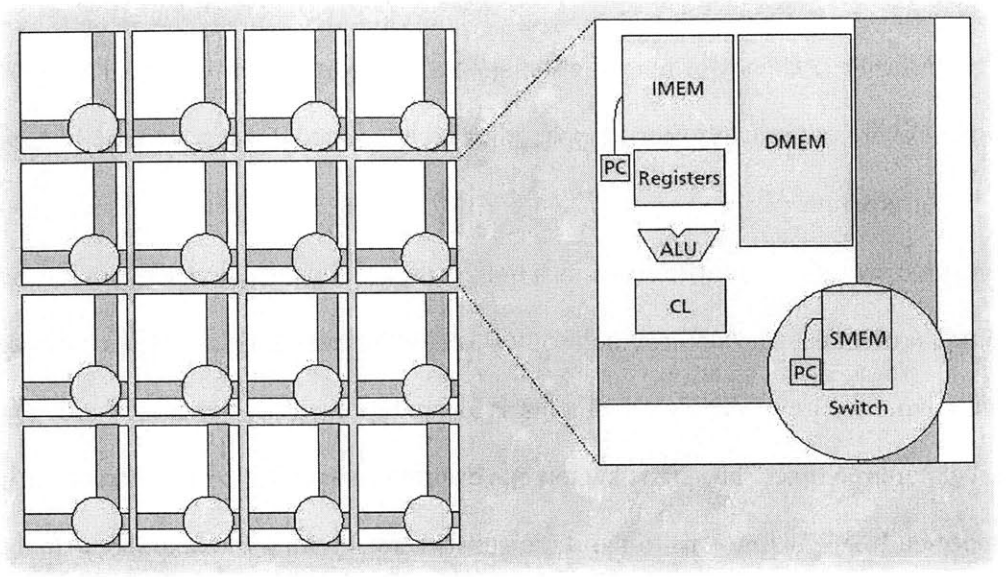
hardwired into the silicon. In addition, we chose to experiment with a decentralized design so as to ensure the scalability of our design. Using a decentralized design also has the advantage of avoiding performance degrading synchronization messages, a problem with which traditional centralized network processors must contend. Table 1-1 presents a features comparison matrix for several approaches to building network processors.

1.5 Raw Overview[1]

Tiled Architecture

The Raw architecture supports an ISA that provides access to all of the gate, pin, and wiring resources of the chip through suitable high level abstractions. As shown in Figure 1-4 the production version of the Raw processor divides the chip into a matrix of 16 identical, programmable tiles. However, the Raw architecture allows for the chip to be scaled to any arbitrary rectangular matrix of tiles such that the number of tiles per side is a power of two. Each tile contains an 8-stage in-order single-issue MIPS-style processing pipeline, a 4-stage single-precision pipelined FPU, a 32 KB data cache, a programmable switch for the static networks, a dynamic network router, and 32KB and 64KB of software-managed instruction caches for the processing pipeline and static router, respectively. Tiles are sized so that all signals can propagate across the tile within exactly one cycle.

[1] For more information see [8].



A Raw processor is constructed of multiple identical tiles. Each tile contains instruction memory (IMEM), data memories (DMEM), an arithmetic logic unit (ALU), registers, configurable logic (CL), and a programmable switch with its associated instruction memory (SMEM).

Figure 1-4. The Raw Microprocessor [10].

On-chip Networks

The tiles are interconnected by four 32-bit full-duplex on-chip networks. There are two identical static networks, which have routes specified at compile time, and two identical dynamic networks, which have routes determined at runtime. Each tile is connected only to its four cardinal neighbors. Every wire is registered at the input to its destination tile. This means that the longest wire in the system is no greater than the length or width of a tile. This property ensures high clock speeds, and the continued scalability of the architecture.

The design of Raw's on-chip networks and their interface with the processing pipeline are its key innovative features. These on-chip networks are exposed to the software through the Raw ISA, giving the programmer or compiler complete control of the wiring resources of the chip, and the ability to orchestrate the transfer of data between the computing resources of different tiles. Effectively, the wire delay is exposed to the user as network hops. To go from corner to corner of the processor takes 6 hops, which corresponds to approximately six cycles of wire delay. To minimize the latency of inter-tile scalar data transport, which is critical for ILP, the on-chip networks are not only register-mapped but also integrated directly into the bypass paths of the processor pipeline.

The static switch in each tile contains a 64KB software-managed instruction cache and a pair of routing crossbars. The static switch routes data on the two static networks between adjacent tiles and the core of the tile on which it is located. This switch executes code determined at compile time. The static switch can execute a variety of route instructions drawn from a simplified ISA. Instructions to the static network switch for a tile take the form:

OP $R1,$R2 ROUTE $2->$csti, $2->$cNo, $2->$cSo, $cSi->$cEo

where OP is one of the instructions from the limited ISA, consisting mainly of move, branch, and decrement instructions. Registers $R1 and $R2 specify one of four static network registers. The arguments following the ROUTE specify a non-conflicting routing of data through the tiles. Data is routed one word at a time when all of the operands for the instruction have been satisfied.

Raw's two dynamic networks support cache misses, interrupts, dynamic messages, and other asynchronous events. The two networks are structurally identical and use dimension-ordered routing: data is routed first in the X direction, then in the Y direction from its source to destination tile to avoid deadlock situations. Data on the dynamic networks is routed as autonomous packets of at most 32 words; an entire packet of data must pass through the dynamic network router before another packet can be routed in the same direction. One of the dynamic networks, the memory dynamic network, is reserved for memory operations, particularly cache operations for the tile cores. The other, the general dynamic network, is free for use by the programmer without constraint.

Chapter 2

The Raw Network Processor

2.1 Overview

To fully exploit the parallel nature of the tiled Raw microprocessor, the network processor is broken into a four stage pipeline and implemented for a four port forwarding engine. Thus each of the 16 tiles on the Raw chip is used for one pipeline stage corresponding to one port. By taking this approach, the network processor is able to fully exploit the inherent parallelism of the tasks involved in processing packets as well as the inherently parallel nature of processing separate parallel streams from the individual linecards. Each stage in the processing pipeline is directly mapped to a task necessary for packet processing. Furthermore, in processing the packets, the design takes full advantage of the stream-oriented nature of the Raw chip, rather than relying on a more traditional memory-oriented approach.

The Raw network processor has undergone several revisions, each of which relies on exactly the same basic pipeline stages. In all versions the packet header is streamed into the pipeline corresponding to the linecard from which it originated via the static network. Thus if one pipeline is stalled the others remain unaffected. Each pipeline stage then processes the header in parallel, before sending the packet to its appropriate egress linecard. Each subsequent revision to the design of the network processor makes more intelligent use of the on-chip networks to yield better performance. The general layout of the router, in particular for that of the final versions, can be seen in Figure 2-1 and Figure 2-2.

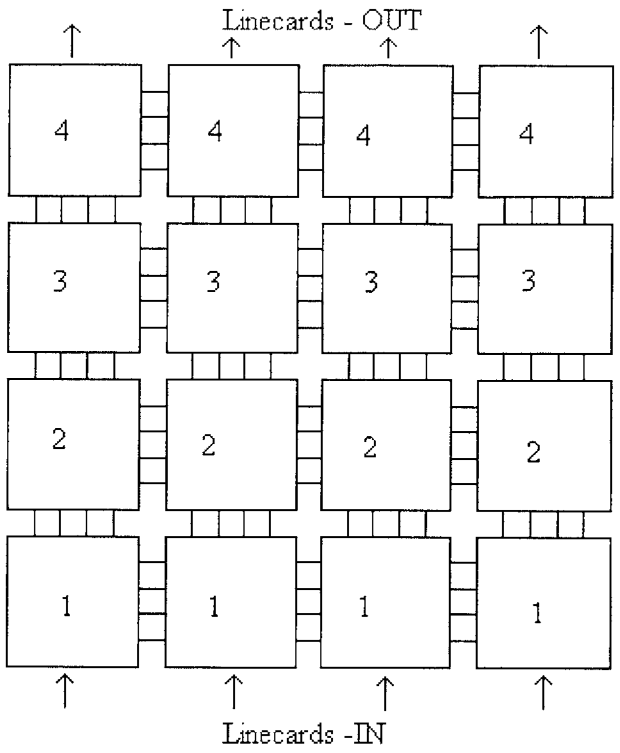


Figure 2-1. The Raw Network Processor – Stage Layout.
This diagram highlights the layout of the stages common to all versions of the Raw network processor. This diagram however is specific to versions II, III, and IV.

2.2 Stage Descriptions

This section outlines operation of the different stages common to all versions of the Raw network processor.

2.2.1 Stage 1: Table Lookup

The first pipeline stage performs the forwarding table lookup. The forwarding table is stored in a DRAM located adjacent to the row of Stage 1 tiles (for all versions except Version I this is off port 12 adjacent to tile 12).

Each Stage 1 tile maintains the address of where the next packet payload for its forwarding column will be stored in memory. When a Stage 1 tile is ready to receive the next packet header, it sends the payload memory address to its connected linecard over the first static network, which signals the linecard to begin sending the packet. The linecard then sends the first sixteen words (64 bytes) of the packet, which is the maximum length of an IP header, on the first static network. The Stage 1 tile's switch simultaneously routes these words into the tile processor and to the next stage. If it is fewer than sixteen header words, the header is padded with zero-valued words until it is sixteen words long. The header has a field that indicates the actual header length, so the padding words are discarded before the packet is sent to the egress linecard. By padding the headers to ensure a length of sixteen words, the switch code is simplified, as the switches can assume that they always route the same number of header words.

To perform the forwarding table lookup, the Stage 1 tile uses the destination IP address of the packet header. The destination IP address of the packet is the fifth data word of the header. This destination IP address is then used to lookup in the forwarding table to which output port to send the packet. The forwarding table layout used is that proposed by Gupta et al. in [3], specifically, their DIR-24-8-BASIC scheme. This design has two tables, both stored in DRAM, and makes a tradeoff to use more memory than necessary to store the routing entries in order to minimize the number of memory accesses for a lookup. The first table, called TBL24, stores route prefixes that are up to, and including, 24 bits long. This table has 2^{24} entries, containing the route prefixes 0.0.0 to 255.255.255. This table is always indexed by the first 24 bits of the address. If the routing prefix is fewer than 24 bits long, the route is duplicated in the table to span all possible indexes. For example, if the prefix is 18.222/16, then there will be 256 entries (all of which have the same destination route), spanning from 18.222.0 to 18.222.255. Although more compact representations are certainly possible, this implementation ensures that lookups for routing prefixes up to 24 bits long will take only a single memory access. The second table, called TBLlong, stores all the route prefixes that are longer than 24 bits. If a route prefix is longer than 24 bits, then 256 entries are allocated in TBLlong, and a pointer to this set of entries is stored in TBL24, as well as a flag indicating that the pointer needs to be dereferenced. The routing lookup result is found by using the 8 low-order bits of the destination address to index the 256 entries from TBLlong. Performing a lookup for an entry with a prefix length longer than 24 bits thus takes two memory accesses. The result from the forwarding table lookup is the output port to which the packet should be sent. The Stage 1 tile sends the output port result to the Stage 3 tile in same forwarding column. In addition to the output port, the tile also sends the address of the packet payload. Finally, the Stage 1 tile updates the memory address for the next packet payload, and sends it to its linecard indicating that it is ready to receive the next packet.

2.2.2 Stage 2: Compute Header Checksum

The second pipeline stage computes the IP packet header checksum. Stage 2 tiles receive the sixteen header words over the first static network from the Stage 1 tile before it. As with the first stage, the second stage switch simultaneously routes the header words into the tile processor and onto the next stage. The checksum is computed by summing the bytes of the header, and then using carries to compute the 1's complement sum. The result of the checksum is a Boolean value, indicating whether the checksum was valid. This result is sent to the Stage 3 tile in the same forwarding column.

2.2.3 Stage 3: Decrement TTL, Update Header, and Send to Output Queue or Discard

The third pipeline stage decides whether to discard or forward the packet and performs the actual forwarding of the header. The third stage receives the sixteen header words over the first static network from the second pipeline stage. It also receives the output port and payload memory address from Stage 1, and the checksum result from Stage 2. These messages each contain a header word that allows the Stage 3 tile to correctly identify the results, as they may be received in an unpredictable order. This stage then updates the time-to-live (TTL) of the packet, reading the TTL field, decrementing it, and writing the value back to the packet header. Because this action changes the bits in the header, it must also recompute the header checksum and update the checksum bits, this is accomplished using the method described in RFC 1624 [7]. Once the TTL has been updated, the packet header is ready to be routed to the next hop. Before performing the routing, this tile stage decides whether the packet should be discarded. For the packet to be routed the TTL must be greater than 1 and the checksum must be valid. Because the checksum result was computed in the second pipeline stage, and the Stage 3 tile only needs verify that the checksum passed. If either the checksum is invalid or the TTL has reached zero, then the tile discards the packet. Otherwise, the packet is valid and is routed to the fourth pipeline stage. The route for the header is determined based on the output port received from the first pipeline stage. Since the packet may be destined for any output port, the dynamic network is used to send the packet header to the appropriate Stage 4 tile output queue connected to the output linecard. Stage 3 sends this data as a dynamic network message that contains the sixteen packet header words and the packet payload address. The third stage represents the switching component of the router, in which the packet is moved from the input forwarding path to the output queue.

2.2.4 Stage 4: Queue Header for Output Linecard

The fourth pipeline stage queues packet headers and payload addresses for the output linecards; thus this is the only pipeline stage tied to the egress linecards, whereas all other stages have been tied to the ingress linecard. When the packet header queue is empty, a Stage 4 tile performs a blocking read from the dynamic network, waiting to receive a header and payload address over the dynamic network from any third stage tile. If the packet queue is neither empty nor full, then the tile will perform a non-blocking read from the dynamic network, periodically polling the network to see if another packet header has arrived. If a packet header has arrived, it is read from the network and appended to the queue. If the packet queue is full, then the router is experiencing network congestion on that output port. The router can be configured to have the tile either discard subsequent incoming packet headers until there is space in the queue, or to stall the earlier stages until the output queue has been sufficiently drained. The output linecard connected to a Stage 4 tile indicates that it is ready to receive the next packet by sending the tile an interrupt over the general dynamic network. The interrupt does not cause the tile to send the next packet immediately, but rather triggers the interrupt handler to set a flag that indicates that the linecard is ready for the next packet. This tile periodically checks this flag, if the flag is set, then the tile clears the flag and sends the first packet header and payload address in the queue over the general dynamic network to the output linecard.

2.2.5 Network Usage

The differences between the various versions of the Raw router lie in their use of the various on-chip networks to connect the various stages. Section 2.3 details how each of these various versions use these networks to connect the stages.

2.2.6 External Devices

All of the versions of the Raw network processor make similar use of external devices. Common to all versions are the input port and output port linecards as well as the DRAM forwarding table. The input port linecards are connected to the Stage 1 tiles and the output port linecards are connected to the Stage 4 tiles. However, how these linecards send the packet payload through the chip from the input port to the output port varies with each version. Each version makes use of different networks to accomplish this task, with most storing the packets temporarily in some external DRAM bank. Section 2.3 details each version's use of the external devices.

2.3 Version Descriptions

This section details the workings of the various versions of the Raw network processor. Figure 2-2 highlights the use of the different networks in the network processor. While the diagram is specific to Version III, the setup for all versions is similar.

2.3.1 Version I

This version is the initial attempt at using the Raw chip as a pipelined network processor capable of handling four ports.

This design sends messages concerning the validity and destination of a packet between the first three stages of the pipeline over the static network. Packet payloads are sent between the third and fourth stages of the pipeline over the general dynamic network. In this design, the first eight words of the packet header are sent into the pipeline, rather than all sixteen. The remaining words of the packet are sent directly to the external DRAM banks for storage until they are read by the output port linecard. Payload data is sent to the DRAM bank using the DMA protocol. The DMA protocol on Raw involves sending data as packets of eight data words and two state words, one the address and one a bit mask, over the memory dynamic network. The output port linecard reads the packet payload from the external memory bank also using DMA transfers. For reads, the DMA protocol involves sending a two word message containing an address and tag to the DRAM. The DRAM responds with a nine word reply containing the tag and the eight words at the memory address requested. Thus data is transferred between linecards and memory in eight word blocks. This version uses two DRAM banks located off tiles 1 and 13 (ports 1 and 11).

In order to accommodate the use of the DMA protocol, packets are sent into the chip from the right side and sent out from the left. Using this left to right orientation, however, creates a contentious bus when sending the dynamic messages between Stage 3 and Stage 4.

This bus is a result of the dimension ordered routing protocol used by the dynamic networks. The use of only two DRAM ports in this version also proved to be a limiting factor in the performance of sending large packets, since each DRAM is shared by two linecards, and the DRAM memory controller used is half-duplexed, thus only able to handle either a read or a write request at one time, not both. Furthermore, the use of the static network for sending communication messages between the first three pipeline stages proved inefficient since the switch processor had to be reset each time to do this at a significant clock cycle penalty.

2.3.2 Version II

This version is an update of the previous version designed to take advantage of the dimension ordered routing of the dynamic network as well as a different memory model. In this design the layout of the pipeline is the same as the first, but rotated 90 degrees. Now packets are sent in from the bottom of the chip and exit from the top. Communication between the first three stages of the pipeline is done using the general dynamic network so as not to have to reset the switch processor. The pipeline was rotated so that the flow of data between the third stage and fourth stage of the pipeline, as well as between memory and the output port linecard, can take advantage of the X then Y direction of routing over the dynamic networks. To allow for this, the design also changed the memory to using a dual ported DRAM. This DRAM allows for requests and responses on both static and memory networks. Using this DRAM increases performance by allowing it to simultaneously read incoming packet payloads from the static network and write out outgoing packet payloads to the memory network for use by the linecards. This version uses four DRAM banks, located off ports 4, 6, 13, and 15, much like the version in Figure 2-2. As in the figure, the input port linecards each are connected directly to a DRAM port using the static network. This layout enables payloads to be sent directly to the DRAMs from the cards without interfering with each other. This layout also takes advantage of the dimension ordered routing of

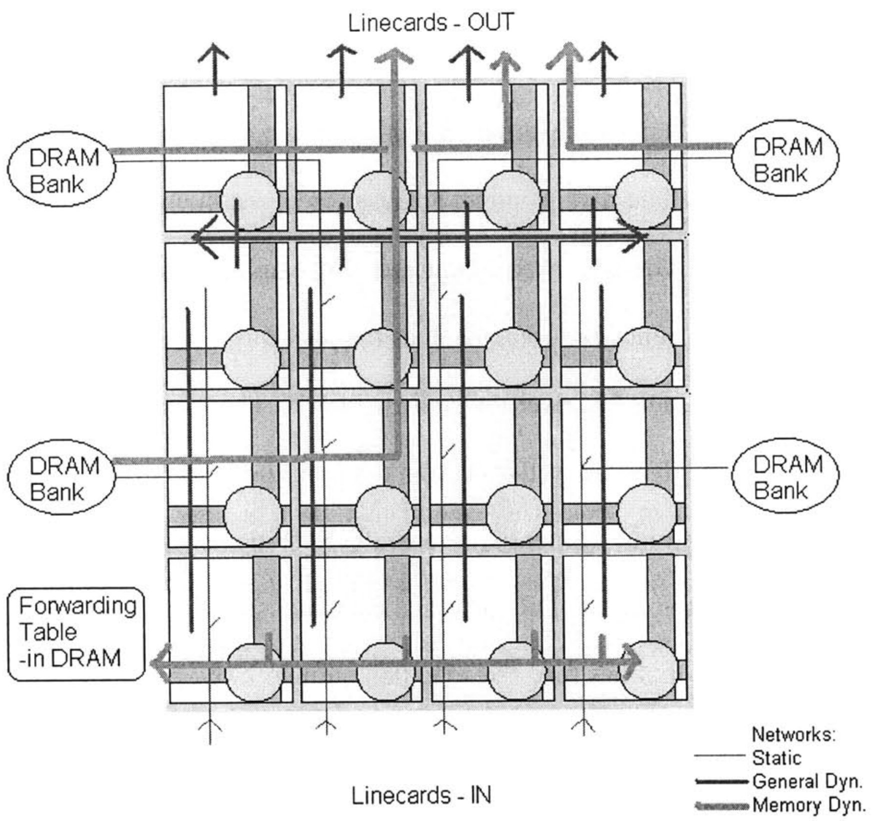


Figure 2-2. The Raw Network Processor – Network Usage.
This diagram highlights the use of the various networks in the Raw network processor. While specific to Version III, the other versions take advantage of many of the same design choices.

the memory dynamic networks since reply messages from the DRAMs are sent first in the X direction to the appropriate linecard then directly up to Stage 4. Since the DRAM banks are located on two different rows, it is possible for each output port linecard to read from a DRAM bank simultaneously without forming any contentious busses. Both the input and output port linecards and memory interact using the Raw cache miss protocol instead of DMA in this version. The Raw cache miss protocol is similar to the DMA protocol, in that it communicates with memory eight words at a time, but is slightly more efficient in that the write requests do not include a bit mask, since all eight words are written. Also in this version the first 16 words of the packet header are read into the pipeline and the rest of the packet is sent to external memory.

2.3.3 Version III

This version is an incremental build on the second. Pictured in Figure 2-2, this version takes advantage of the internal network buffers when sending packets between the third and fourth stages of the pipeline. The main change, however, is in the memory model. Instead of using dual ported DRAM, this design takes advantage of streaming DRAM. This DRAM is able to process requests from different networks, much as the previous, but the protocol allows for words to be streamed into the DRAM over the static network. The streaming DRAM works by receiving a message over the dynamic network from the input port linecard containing the address and number of words to be sequentially read off the static network. As words are streamed across the static network, the DRAM reads them and stores them sequentially starting at the specified address. When it has read the specified number of words, the transaction is complete. This DRAM also has the ability to simultaneously handle requests from other networks, such as requests from the output port linecards for data, while servicing the streaming transaction. Using this memory model greatly improves performance for large packets over the previous versions. In Version 3, the output port line card also sends requests to the memory in batches so as to reduce the latency in receiving the packet payloads from memory.

2.3.4 Version IV

This version represents a minor change to the previous to take advantage of the fact that modern routers employ linecards which themselves contain significant buffering and processing resources. As such, this version is identical to the previous except it removes payload buffer DRAMs. Instead, payload data is stored on the input port linecard until requested directly by the output port linecard. To accomplish this, the input port associates an ID number to a specific packet as well as the port from which the packet was received. This information is encoded as the word which was previously the address. The output port linecard then receives this word with the packet header, decodes it, and sends the requests for the payload of the specified packet to the appropriate input port linecard. To avoid clogging the memory networks of the chip with the payload data of large packet, this transaction is accomplished using the Raw cache miss protocol. Thus the output port linecard sends a request for eight words and receives a reply of eight words of data. The limiting factor in this version proved to be the creation of a bus on the bottom row of the memory network of payload data being sent from the input to output linecards. This bus is a result of the dimension ordered routing used on the network. However, removing the buffering in the external DRAM banks proved to provide a great enough performance benefit to offset the effects of this bus.

2.4 Scalability

Ensuring that the Raw network processor be able to scale to handle varying number of linecards was of great importance in the choice of design. All four versions are able to handle anywhere between one and four linecards trivially. To handle fewer than four linecards, simply do not connect one of the input pipelines to an input port linecard or do not connect one of the output tiles to an output port linecard. While it may seem possible to scale the processor to greater than four linecards by simply increasing the number of tiles in the Raw matrix, adding an extra pipeline for each additional linecard, thus taking advantage of the inherent scalability of the tiled architecture of the chip, this approach does not work for all of the versions discussed. In fact, only Versions I and IV can be scaled beyond four linecards. Versions II and III can not be scaled as a result of their use of the static networks for sending payload data to the off-chip DRAMs. For these two versions, the static network pipes were carefully chosen so as to provide direct streams of data from the linecard to the DRAM. Under the rules governing the static crossbar operation, all operands must met before executing a routing instruction, thus, given the random nature of internet traffic, it is not possible to cross the static network paths to the

DRAMs. Furthermore, it is not possible to layout similar non-crossing paths for greater than four linecards using our design. Versions I and IV, by contrast, use only the memory dynamic network for transferring payload data between the in and out port linecards, and as a result are free to be scaled to arbitrary sizes. An attempt was made at a scalable version of the Version III router. This version uses only the dynamic networks—no static networks were used. While this version proves to be scalable, this benefit comes at the cost of degraded performance, as discussed in Section 3.6.

2.5 Buffering

Traditional routers are based on one of several buffering disciplines. These include: shared buffering, in which all incoming packets from the various linecards are stored in a global pool of memory from which they are dequeued and processed, the IXP2400 for example uses this design; input buffering, in which the buffering is done at every input port; output buffering in which the buffering is done at every output port; and internal buffering, in which the switching fabric itself is able to buffer packets internally in the face of switching contention. Each of these schemes has its own advantages and drawbacks as well as implications for the overall design of the system. In theory output buffered routers are preferable since they guarantee 100% throughput even when there is contention for output links; these routers require a non-blocking switch fabric and a memory bandwidth of N*R (where N is the number of linecards and R is the rate of each). Shared memory architectures must be designed around a sophisticated memory controller capable of delivering a memory bandwidth of $N^2 * R$. Input buffered routers, by contrast, offer simpler designs than the latter two, but are only capable of yielding a theoretical maximum throughput of 58% due to head-of-line blocking. To achieve a 100% throughput rate, input buffered switches must use technique known as Virtual Output Queuing whereby packets at each input are sorted into queues for each output, then a global scheduling algorithm resolves contention by using a variant of a bi-partite graph matching algorithm to decide the order of transfers from these virtual output queues. This technique, however, requires that the global scheduling algorithm can instantaneously look at all of the input queues to generate a non-blocking switch schedule.

In contrast to the traditional designs of using just one of these methods, the Raw network processor uses a hybrid combination of input buffering, output buffering, and internal buffering. This hybrid design was chosen so as to fully exploit the architectural features of the tiled chip to achieve maximum performance. In all versions of the processor some input buffering is assumed, as all linecards have at least some buffer space to handle the non-deterministic nature of internet traffic. For large packets payloads are buffered either in external memory or in memory on the linecard. Furthermore, packet headers are buffered in Stage 4 before being sent to the output port linecards, thus constituting the output buffered portion of the hybrid design. The innovative feature of the design, however, is that it takes advantage of the internal buffering available to Raw inter-tile communication networks, thus constituting the internal buffered portion of the hybrid design. Packet headers are sent between Stages 3 and 4 over the general dynamic network as message of sixteen header words, an address word, and a header encoding the destination tile. Fortunately, Raw's dynamic networks are built with internal buffers. Each tile has four words of buffer space for messages traveling in each of the north, south, east, west directions as well as into the tile processor. In addition each tile has sixteen words of buffer space for messages traveling out of the tile processor to the dynamic network. Thus if a message is blocked on its way to its destination by an earlier message, the tile can continue sending the message before being forced to stall by filling up these internal network buffers. By taking advantage of this internal buffering, the design enables all four of the Stage 3 tiles to simultaneously send headers to any set of the Stage 4 tiles with all extra data words being absorbed by the internal network buffers, thus not forcing any of the input pipeline stages to stall.

Chapter 3

Performance Evaluation

This section discusses the implementation of the Raw network processor as well as the performance evaluation and analysis.

3.1 Implementation

The Raw network processor was developed and tested using a validated cycle-accurate simulator of the Raw chip. Details of the simulator are given in [9]. The simulator was used instead of the actual hardware version of the chip because the only existing motherboard which houses the Raw processor is very limited in how devices can be attached to it. Thus by using the simulator there was greater freedom to place external devices around the Raw processor. Also using the simulator provided much greater flexibility in testing the processor using linecards of different capabilities than would have existed were actual hardware devices used. Using simulated linecards allowed the focus to be placed on the design of the processor on the Raw chip itself, without worrying about the interfaces to the linecards.

The Raw network processor was written in C code for the tile processors and assembly code for the switch processors. The code is about 1300 lines of C and 500 lines of switch assembly. The linecards were written in 800 lines of a C-like language supported by the Raw simulator for external devices.

3.2 Experimental Methodology

As described above, the Raw network processor was designed and tested entirely in the Raw simulation environment using simulated linecards and memory. The simulated input linecards read their input from files containing the packets in byte streams. These linecards have a configurable rate at which they send their next packet to the Raw processor. If the rate is greater than the saturation point of Raw, then the linecards will stall until the Raw processor is able to accept the next packet. The output linecards write every byte they receive from the Raw processor to trace files, which are validated in post-processing scripts to ensure that the router is routing correctly. For the timing results, the Raw microprocessor is clocked at 425MHz. The input linecards send UDP packets at specified rates.

The Raw network processor was tested on two types of input packet files: randomly generated and captured traces[2]. The randomly generated packet files are traces of 4000 packets of a specified size with 128 different source and destination addresses. Each of these 128 source and destination addresses was randomly assigned to an input and output port, respectively. Each evaluation was performed using the same configuration four times with four different random traces, and averaged the results. There was no difference in performance when using traces longer than 4000 packets or using more than 128 addresses. For the evaluations, the forwarding table was initialized with 32-bit entry prefixes, that is, each address has its own entry in the forwarding table. The 32-bit prefixes also require two memory references for each table lookup, which is the worst-case performance. The captured trace files each contain 10,000 packets. For the live traces, the forwarding table was initialized with randomly assigned 24-bit entry prefixes, which more accurately reflect actual routing table entries. The results for the live packet traces were also averaged from two different traces, each containing 10,000 packets. These traces represent worst case scenarios, as both traces have over 90% of the packets destined for one output port. For the live packet traces, the timing data included in the traces was ignored. Instead, the linecards were configured to send the packets at the maximum rate that the router would accept them.

[2] The packet traces were obtained from http://lever.cs.ucla.edu/ddos/traces/

The evaluation measured the forwarding rate, throughput, and latency of the router. The forwarding rate was measured using 64-byte packets, which are minimum-sized TCP packets. Throughput was measured by using 1500-byte packets, which is the maximum size of an Ethernet packet. Finally, latency was measured to determine how much time the router delays the packet in the forwarding process. Latency was measured as the time it takes from the first word of a packet to arrive at the router to the time it takes for the first word of a packet to exit the router. The latencies for both large and small packets were measured.

3.3 Forwarding Rate

The maximum loss-free forwarding rate (MLFFR) of the various versions of the Raw network processor was evaluated by measuring the rate at which a router can forward 64-byte packets over a range of input rates. Minimum-size packets stress the router harder than larger packets; the CPU and several other bottleneck resources are consumed in proportion to the number of packets forwarded, not in proportion to bandwidth. Plotting forwarding rate versus input rate indicates both the MLFFR and the behavior of the router under overload. An ideal router would emit every input packet regardless of input rate, corresponding to the line y = x.

Figure 3-1 compares the forwarding performance of the various versions of the Raw network processor. As the figure shows, the Raw network processor is able to achieve a MLFFR of 10 million packets per second. The bottleneck for minimum-sized packets is either the routing table lookup or computing the header checksum; this is discussed in more detail in Section 3.5.

It is interesting to note is how well the various versions come to approach the null router based on Version III, not shown in the graph. The null router has a MLFFR of 15.5 million packets per second; it does no actual processing, it merely serves as reference for the maximum possible LFFR of the pipelined design. For the null router the checksum and TTL are not computed, the results are always successful, and the output port is always the same as the input

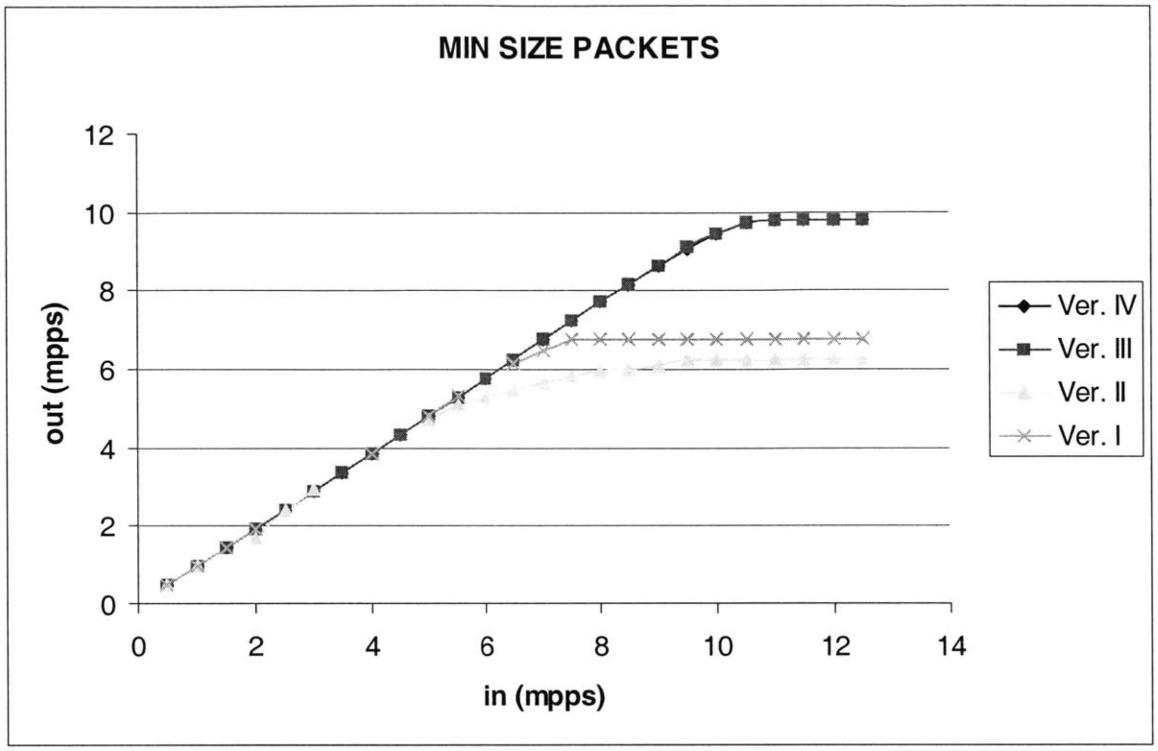


Figure 3-1. Raw Network Processor Forwarding Rates.
The forwarding rate is plotted as a function of the input rate for 64-byte packets for all versions.

port. Also interesting to note is that the maximum possible rate of forwarding packets straight through the Raw processor at one word per cycle per port is 106.25 million packets per second. By comparison, Intel's published results for an eight port gigabit Ethernet router made using two of its IXP2400 each running at 600Mhz indicate a MLFFR of 12 million packets per second [6]. Here it must be noted that while this router is running twice as many ports, it is using two network processors running at over 140% the speed of the Raw chip.

3.4 Throughput

While small packets measure the processing capabilities of a router, large packets serve as an important test of the router's internal bandwidth and its ability to move data between input and output ports. Figure 3-2 shows the performance of the various Raw network processor

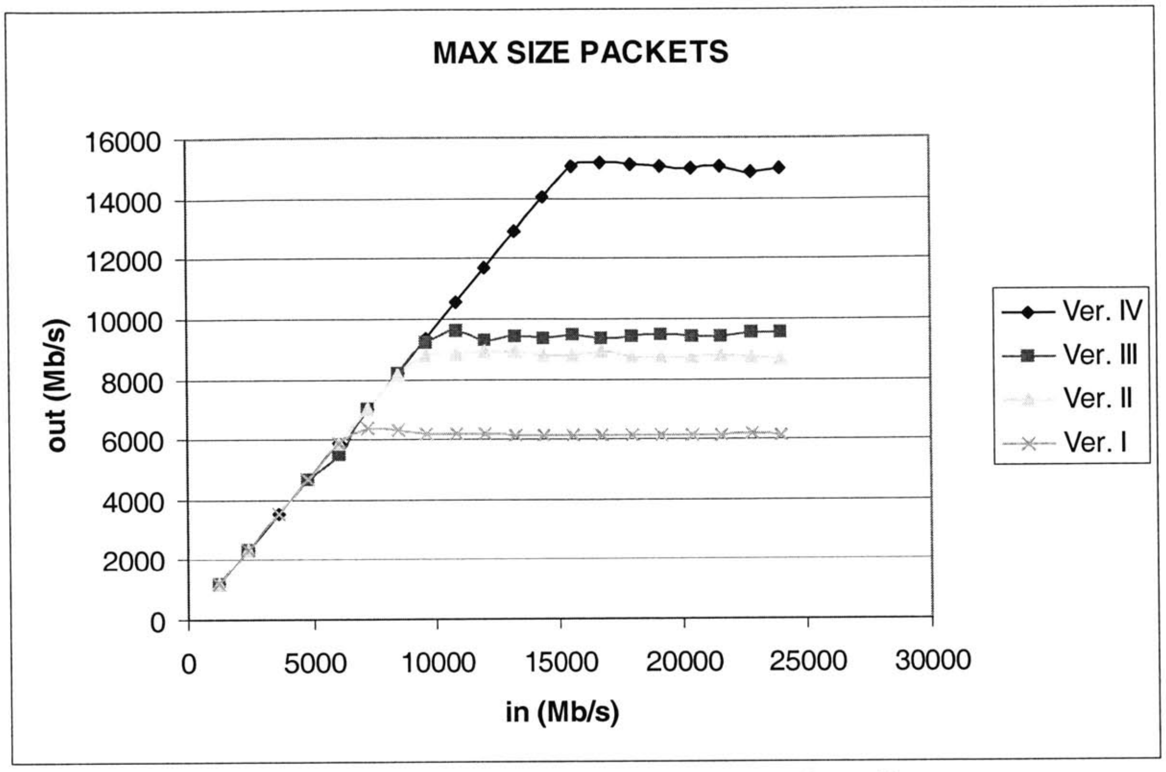


Figure 3-2. Raw Network Processor Throughput Rates.
The throughput rate is plotted as a function of the input rate for 1500-byte packets for all versions.

versions tested on 1500 byte packets. As can be seen in the figure, the Raw network router is able to achieve a maximum sustained performance of 14.4 Gb/s using Version IV, which stores packet payloads on the linecard. For less capable linecards the router is able to achieve a maximum performance of 8.4 Gb/s using Version III. The bottleneck for all versions of the router is the transferring of packet payload data over the memory network in small chunks. In Version III, payload data must also first be sent to the DRAMs, which adds additional time.

Figure 3-2 also demonstrates the performance improvements effected by the various enhancements to subsequent versions of the router. The improvement from Version I to Version II is largely due to the elimination of contention on the memory networks. The further gain between Version II and Version III comes from using more efficient memory transactions, namely streaming memory transactions for the input payloads and pipelined memory requests for the outputs. The performance improvement of Version IV over Version III is largely due to the fact that since payload data is stored on the linecard, time does not need to be wasted streaming it to DRAM. Interestingly, the IXP2400 based router described in the previous section is only able to route packets at a rate of 8 Gb/s [6].

3.5 Latency

Another important metric of router performance is latency. Latency measures how long it takes for a packet to pass through the router. Even if a router is capable of maintaining a high sustained throughput and forwarding rate, it is of no use if it takes unreasonably long for a packet to pass through it. If the router has high latency and delays a packet for too long, then packets can arrive at the receiver out-of-order. Out of order packets can cause problems at the application layer, but should also be avoided because they can cause TCP to conclude that packets are being dropped, which will lead to spurious retransmissions and may lead to congestion.

Table 3-1 details the latency measured for both 64-byte and 1500-byte packets at the MLFFR for the various versions of the Raw network router. The large difference in latency between the small and large sized packets demonstrates the impact the inter-tile communication networks have on the performance of the overall design, since the payload data is sent over some combination of these networks in all versions. Table 3-2 lists the number of cycles it takes to perform the various tasks for each stage. For Versions III and IV, it takes ~160 cycles to process a packet over the first three stages of the pipeline, thus it takes ~160 cycles from the time the packet header arrives at the pipeline to when it is finished being sent to Stage 4. This time is constant regardless of the size of the packet, so all of the remaining latency time is spent transferring the payload data between the linecards. As Table 3-1 shows, the various improvements in the inter-tile network usage among the different versions help to lower this

	64 byte packets		1500 byte packets	
Version	MLLFR (mpps)	Latency (cycles)	MLFFR (mpps)	Latency (cycles)
I	7.2	537	0.5	4335
II	9.8	223	0.65	2625
III	10	216	0.7	2450
IV	10	216	1.2	1027

Table 3-1. MLFFR and associated Latency times for the various versions of the Raw Network Processor.

Stage	Task	Cycles
1	Forwarding Table Lookup ≤ 24 bit prefix	29/~100
1	Forwarding Table Lookup > 24 bit prefix	36/~150
2	IP header checksum	78
3	Update TTL and Checksum	26
4	Dequeue and send to output linecard	22

Table 3-2. Raw Network Processor Task Times. The time taken to perform the different packet header processing steps. The first value for the forwarding table lookup is the time if the result is already in the processor data cache; the second value represents a cache miss.

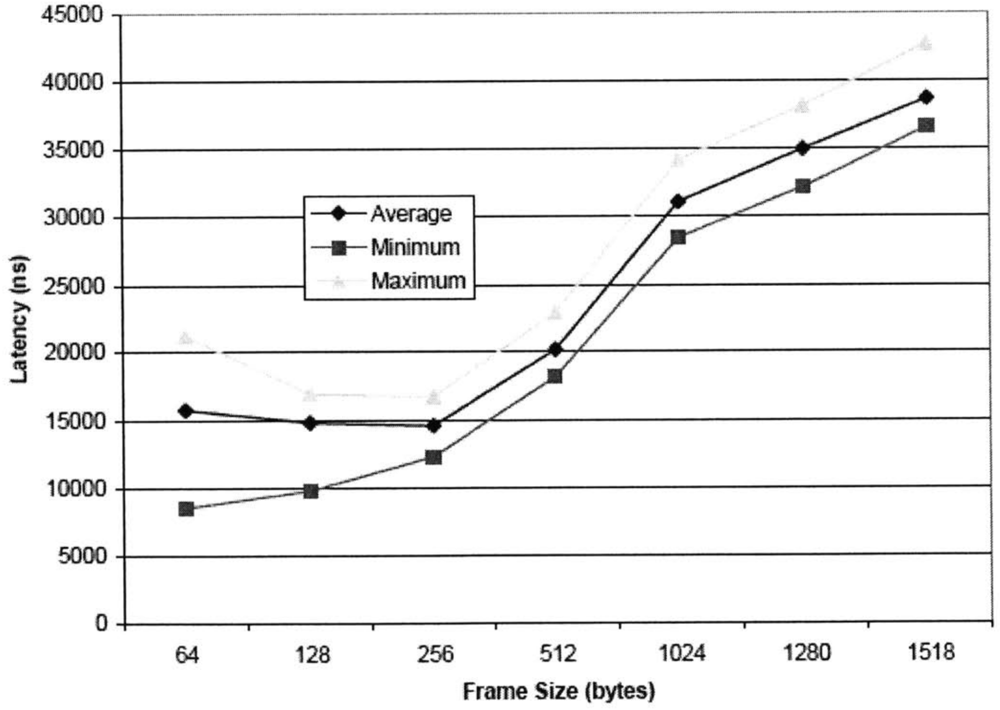


Figure 3-3. Intel IXP2400 Latency.
This plot shows the latency associated with packets of varying sizes using the Intel IXP2400 based router [6].

latency. As a comparison Figure 3-3 plots the latency of the IXP2400 based router described earlier for packets of varying sizes [6].

3.6 Scalability Performance Measurements

Along with the four versions of the Raw network processor described earlier, two additional variations of the design were also tested. These two variations served to test the performance of the router when scaled to different numbers of linecards. Figures 3-4 and 3-5 plot the forwarding rates of a variation of the Version III router using only two ports against the full four port version. Note that the two port version is able to achieve 20% speedup in forwarding minimum sized packets, yet yields the same MLFFR for the 1500-byte packets. The speedup for small packets is a result of each linecard having access to two pipelines to process its packets; the identical results for large packets are due to the effects of using the memory network to move the packet payload data.

Figures 3-6 and 3-7 similarly plot the forwarding rates and throughputs of an entirely dynamic network based version of Version III. This version was constructed to both demonstrate the performance advantages of using the stream-oriented static networks and to determine the performance hit taken by making a scalable variation of Version III. Since this version uses only dynamic networks, it is not subject to the static network crossing constraint of Version III, and thus can be scaled arbitrarily. Figure 3-6 highlights the performance advantage of using the static network to stream packet headers into the tiles, since all of the data for minimum sized packets is sent directly to the tiles. Figure 3-7 also shows the advantage of using the static networks, except here for sending the payload data to the DRAMs. This figure highlights the 10% performance penalty paid for making the Version III design scalable.

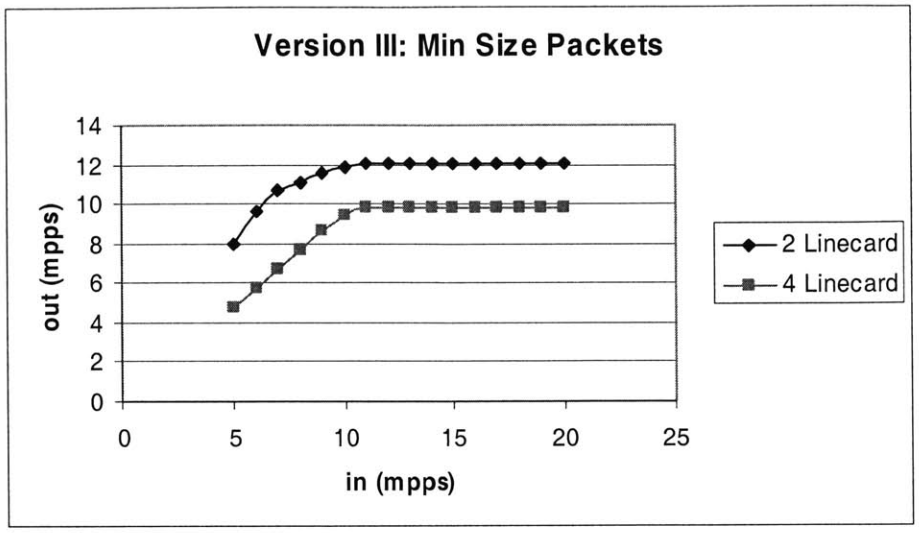


Figure 3-4. Forwarding Rate of Version III, comparing the 2 and 4 linecard versions on 64-byte packets, to prove the scalability of the router.

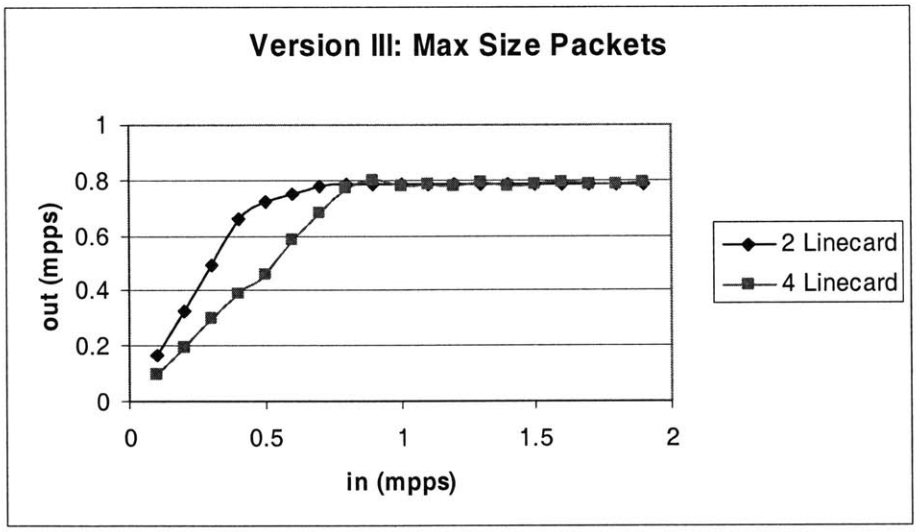


Figure 3-5. Forwarding Rate of Version III, comparing the 2 and 4 linecard versions on 1500-byte packets, to prove the scalability of the router.

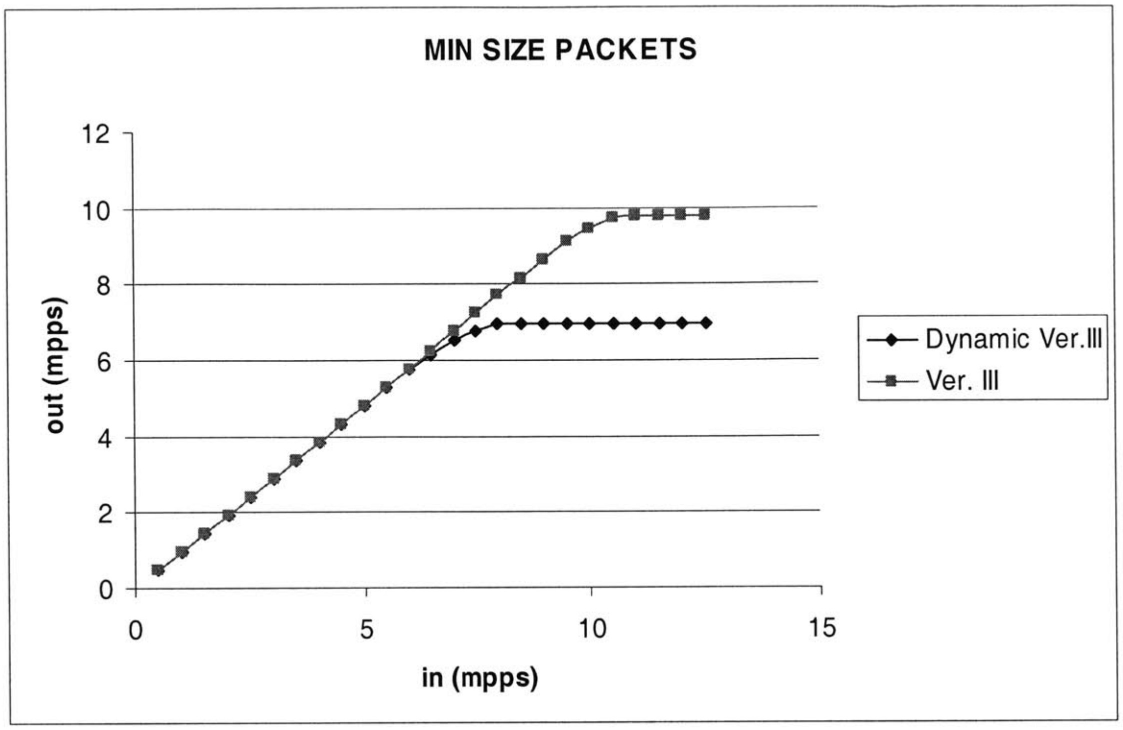


Figure 3-6. Forwarding Rate of Version III compared with the entirely dynamic network based Version III. Note the advantage the use of the static networks provides.

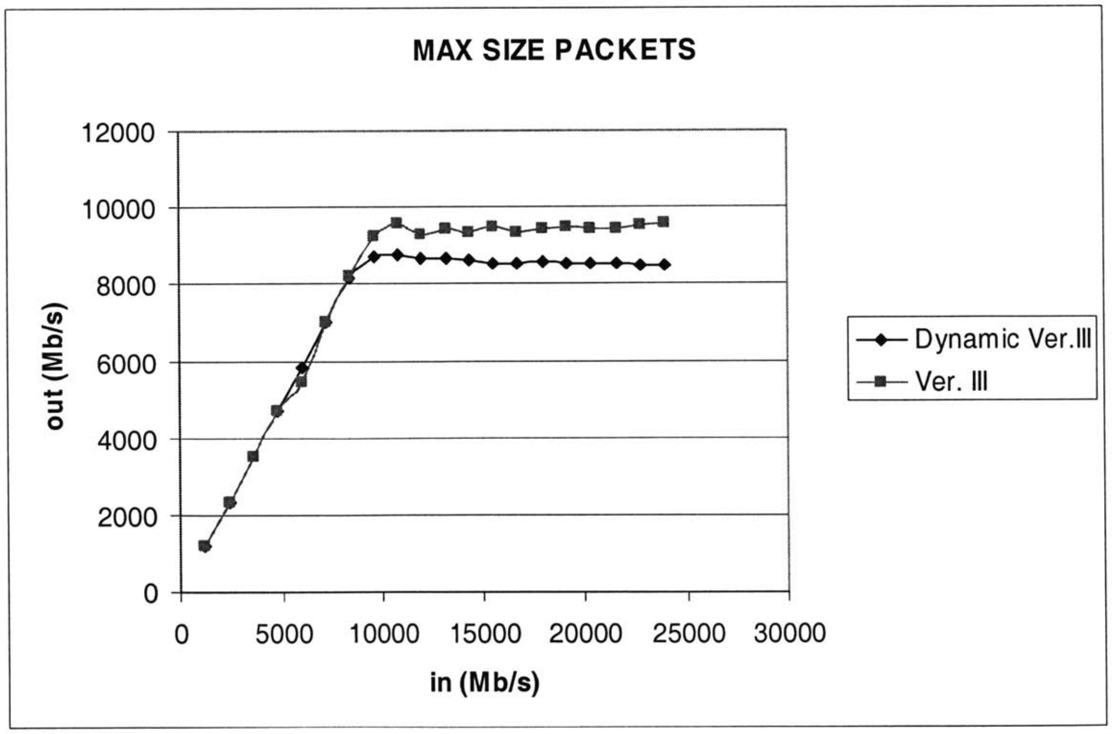


Figure 3-7. Throughput of Version III compared with the entirely dynamic network based Version III. Note the advantage the use of the static networks provides.

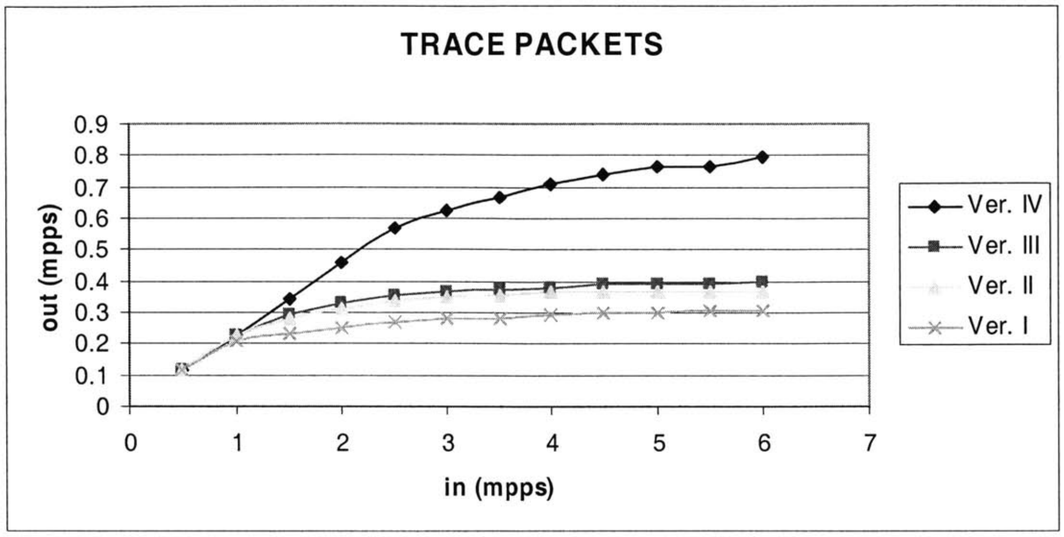


Figure 3-8. Forwarding Rate of all versions, using captured internet traces.

Chapter 4

Suggestions for Design Modifications and Conclusion

4.1 Suggestions for Architectural Design Modifications

The work on using the Raw microprocessor as a network processor has yielded insight into possible areas for architectural improvement for future versions of the Raw chip as well as tiled, general purpose microprocessors in general. The most significant is the addition of a non-stalling crossing routing instruction for the static network switches. One of the key architectural contributions of the Raw microprocessor is its introduction of programmable static networks for managing stream-oriented data flow throughout the chip. Yet, as described in Section 1.5, all operands for instructions to the static switch must be met before the routing instruction is executed. For example an instruction such as NOP ROUTE $cSi->$csti, $cWi->$cNo, requires input data to be available on both $cSi and $cWi before executing the instruction. This design has the negative side effect of making it extremely difficult to cross static network streams of data unless the arrival time of the data is known beforehand. In using the Raw processor as a network processor, an embedded application itself, the static networks are utilized to stream data from the linecards to the DRAMs. As discussed in Section 2.4, it is the fact that the static network pipelines can not be crossed that made it impossible to scale the Version II and Version III network processors. If it were in fact possible to cross these pipelines or control the two static networks independently, this problem would have been averted. The ability to cross static networks paths or control the static networks independently has more far reaching implications than just using the Raw chip as a network processor; such capabilities would make the Raw chip much more flexible for all embedded application uses by enabling the designer to layout fixed, crossable streaming data paths throughout the microprocessor.

4.2 Conclusion

This thesis describes the design and implementation of a scalable, flexible, high performance network processor built using a tiled, general purpose microprocessor. By taking advantage of the inherently parallel nature of the tasks involved in internet packet routing and mapping these tasks on to the innately parallel structure of the tiled microprocessor, the Raw network processor is able to achieve performance that matches or exceeds that of commercially available, custom designed network processors, but required significantly less development time and cost. This can be seen from the comparisons to the Intel IXP2400 in the previous chapter. Yet, these results were achieved while maintaining the flexibility to add new features and the scalability to accommodate varying numbers of linecards, by implementing the processor entirely in software and taking careful use of the stream-oriented interconnects and exposed pin resources of the tiled microprocessor. We hope that these encouraging results will motivate further research into using tiled, general purpose architectures for implementing network processors.

Bibliography

[1] Anderson, James. The Raw Router: Gigabit Routing on a General-purpose Microprocessor. Masters Thesis, MIT, 2004.

[2] Baker, F.. RFC 1812 – Requirements for IP Version 4 Routers. In *Internet Request For Comments*, number 1812, Jun 1995.

[3] Gupta, Lin, and McKeown. Routing Lookups in Hardware at Memory Access Speeds. In *INFOCOM (3)*, pages 1240-1247, 1998.

[4] Intel Corp. Intel IXP2400 Network Processor, Data Sheet. Technical Memo, Intel Corp, Feb 2004.

[5] Intel Corp. Intel IXP2400 Network Processor, Product Brief. Technical Memo, Intel Corp, 2003.

[6] Meng, David. IXP2400 Intel Network Processor IPv4 Forwarding Benchmark Full Disclosure Report for Gigabit Ethernet. Technical Memo, Intel Corp., Mar 2003.

[7] Rijsinghani, Anil. RFC 1624 – Computation of the Internet Checksum Via Incremental Update. In *Internet Request For Comments*, number 1812, May 1994.

[8] Taylor, Michael Bedford. The Raw Processor Specification. Technical Memo, CSAIL/Laboratory for Computer Science, MIT, 2004.

[9] Taylor, et al. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Stream. In *ISCA*, 2004.

[10] Taylor, et al. Baring it all to Software: Raw Machines. *IEEE Computer*, pages 86-93, Sept 1997.

[illegible]