The Raw Router: Gigabit Routing on a General-purpose Microprocessor

by

James William Anderson

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

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004 [June 2004]

© James William Anderson, MMIV. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUL 2 0 2004

LIBRARIES

Author .....
Department of Electrical Engineering and Computer Science
May 7, 2004

Certified by .....
Anant Agarwal
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by .....
Umar Saif
Research Scientist
Thesis Supervisor

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

ARCHIVES

The Raw Router: Gigabit Routing on a General-purpose Microprocessor

by

James William Anderson

Submitted to the Department of Electrical Engineering and Computer Science
on May 7, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Conventionally, high-speed routers are built using custom hardware, typically dubbed as network processors. A prominent example of such a network processor is the Intel IXP1200. Such a network processor typically takes years of effort in the design, fabrication and refinement of the custom hardware, and, worst, must be frequently redesigned to meet the oft-changing requirements of emerging network applications.

This thesis presents the design and implementation of a *software* gigabit network router on a general-purpose microprocessor using MIT's Raw microprocessor. The Raw processor, developed by the Computer Architecture Group at MIT, has sixteen RISC processors, arranged as a grid, that communicate through programmable switches and hardware network interconnects with single-cycle latencies. As opposed to previous high-speed network routers, the Raw router is built without using any custom hardware, and achieves its performance by carefully programming and orchestrating, in software, the interconnects within the Raw chip. Our Raw implementation uses stream-oriented abstractions and differs significantly from that of commercial network processors, which use memory-oriented semantics. Consequently, the Raw router is not only flexible in its architecture and easy to upgrade, our results show that the Raw Router is more than three times faster than a router built using Intel's custom-made IXP1200 network processor.

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, my thesis adviser, for his assistance, ideas, and energy. His vision for Raw has been a source of inspiration in my work.

I have greatly benefited from my collaboration with Umar Saif, my other thesis advisor, who has provided valuable ideas and feedback through the design of the router and the writing of this thesis.

Thanks also go to Tony Degangi, for his work on the initial implementations of the router.

Finally, special thanks go to my colleagues in the Computer Architecture Group for their suggestions, comments, and invaluable assistance, including Michael Taylor, Jason Miller, David Wentzlaff, Nathan Shnidman, and Volker Strumpen.

I dedicate my thesis to my parents, who gave me the opportunity to study at MIT.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rapid advancements in transmission link technologies have increased potential link capacity to the point that network routers, rather than cables, are the bottlenecks. To cope with the ever-greater processing demands high-bandwidth links place on routers, vendors use custom hardware to achieve the required performance. Recently, several vendors have developed *network processors*, programmable special-purpose microprocessors designed to exploit the parallelism inherent in network workloads. Such a network processor typically takes years of effort in the design, fabrication, and refinement of the custom hardware, and, worst, must be frequently redesigned to meet the oft-changing requirements of emerging network applications.

This thesis presents the design and implementation of a *software* gigabit network router on a general-purpose microprocessor: MIT's Raw microprocessor. The Raw processor, developed by the Computer Architecture Group at MIT, has sixteen RISC processors, arranged as a grid, that communicate through programmable switches and hardware network interconnects with single-cycle latencies. As opposed to previous high-speed network routers, the Raw router is built without using any custom hardware, and achieves its performance by carefully programming and orchestrating, in software, the interconnects within the Raw chip. Our Raw implementation uses stream-oriented abstractions and differs significantly from that of commercial network processors, which use memory-oriented semantics. Consequently, the Raw router is flexible in its architecture and easy to upgrade, but achieves these benefits without

sacrificing performance.

1.1 Thesis Overview

This chapter describes the major contributions of the Raw Router, as well as an overview of the router and the Raw microprocessor architecture. The remainder of the thesis is organized as follows. The next chapter discusses the router design in detail. Chapter 3 describes the implementation of the Raw router and presents the performance evaluation and analysis. Finally, Chapter 4 proposes future work and concludes.

1.2 Contributions

The primary contribution of this thesis is the design of a high-performance software router for a general-purpose parallel architecture. This design encompasses the placement of functionality on processing elements and the communication patterns between them. The careful orchestration of the flow of packets as they are processed and sent through the router is the key to achieving maximum forwarding rates and minimal latency. Additionally, the design maximizes the use of the architecture's internal bandwidth for the storage and retrieval of packet payloads while the headers are being processed.

1.3 Raw Router Overview

The Raw Router uses two Raw processors—one for the control-plane and one for the data-plane—and four linecards. The control-plane handles all aspects of router management, such as maintaining the forwarding tables, and processes special packets, such as IP packets with options. The data-plane performs the processing required by the IPv4 specifications [1] and actually switches the packets to the correct output ports.

The basic design of the the Raw Router data-plane is a four stage pipeline, with four independent forwarding paths. Each forwarding path processes packets for one linecard. In the 4x4 Raw tile grid, the forwarding paths are columns of tiles, and the pipeline stages are rows of tiles. All four forwarding columns process packets simultaneously and at the maximum possible rate—if one forwarding column is stalled, it will not effect the other forwarding columns.

The Raw Router has been implemented in C code for the tiles and assembly code for the switches. It meets all of the requirements specified for an IPv4 router.

1.4 The Raw Microprocessor

This section provides a brief overview of the Raw architecture.[1] A more detailed discussion of the architecture is available elsewhere [7, 8, 9].

Tiled Architecture

The Raw architecture supports an ISA that provides a parallel interface to the gate, pin, and wiring resources of the chip through suitable high level abstractions. As shown in Figure 1-1 the Raw processor divides the chip into an array of 16 identical, programmable tiles. A 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, two types of communication routers—-static and dynamic—and 32KB and 64KB of software-managed instruction caches for the processing pipeline and static router, respectively. Each tile is sized so that the amount of time for a signal to travel through a small amount of logic and across the tile is one clock cycle.

On-chip Networks

The tiles are interconnected by four 32-bit full-duplex on-chip networks, consisting of over 12,500 wires. Two of the networks are static (routes are specified at compile

[1]This section borrows heavily from the architecture overview in [10].

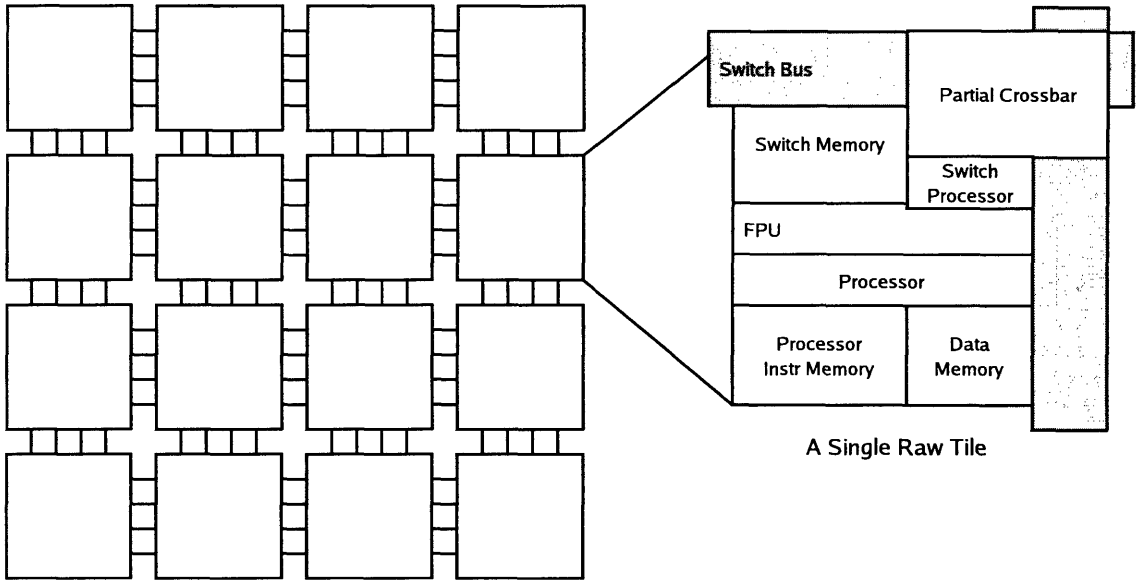

Figure 1-1: **The Raw Microprocessor.** The Raw microprocessor has 16 tile processors (a detailed view of a tile processor is also shown) arranged as a 4x4 grid.

time) and two are dynamic (routes are specified at run time). Each tile is connected only to its four 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 interconnect and its interface with the processing pipeline are its key innovative features. These on-chip networks are exposed to the software through the Raw ISA, thereby giving the programmer or compiler the ability to directly program the wiring resources of the processor, and to carefully orchestrate the transfer of data values between the computational portions of the tiles – much like the routing in an ASIC. 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 router in each tile contains a 64KB software-managed instruction cache and a pair of routing crossbars. Compiler generated routing instructions are 64 bits and encode a small command (e.g., conditional branch with/without decrement) and several routes – one for each crossbar output. These single-cycle routing instructions are one example of Raw's use of specialization. Because the router program memory is cached, there is no practical architectural limit on the number of simultaneous communication patterns that can be supported in a computation.

Raw's two dynamic networks support cache misses, interrupts, dynamic messages, and other asynchronous events. The two networks use dimension-ordered routing and are structurally identical. One network, the memory network, follows a deadlock-avoidance strategy to avoid end-point deadlock. It is used in a restricted manner by trusted clients such as data caches, DMA and I/O. The second network, the general network, is used by untrusted clients, and relies on a deadlock recovery strategy [4].

Chapter 2

Design

This chapter discusses the design and architecture of the Raw Router. Throughout the design and development, we have evaluated our working design to discover the bottlenecks, and we have made substantial revisions and improvements. This chapter presents three major versions: the final optimized design (Version III), the preliminary design (Version I), and the predecessor to the final version (Version II). The revisions primarily encompass improvements in the communication patterns and memory transactions to reduce contention in the Raw on-chip networks and the off-chip memories.

2.1 Raw Router Overview

The Raw Router uses two Raw processors—one for the control-plane and one for the data-plane—and four linecards. The control-plane handles all aspects of router management, such as maintaining the forwarding tables, and processes special packets, such as IP packets with options. The data-plane actually processes the packets and switches them to the correct output ports. The input linecards are connected to the southern edge of the control-plane. All data packets flow through the control-plane directly to the data-plane, which is connected to the northern edge of the control-plane. The output linecards are connected to the northern edge of the data-plane. Figure 2-1 shows this configuration.

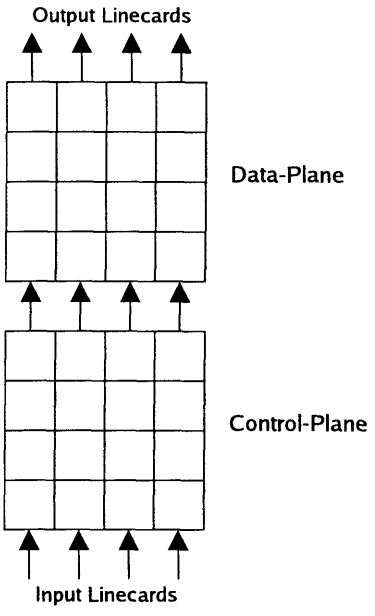

Figure 2-1: **Raw Router Overview.** The Raw Router is designed to use two Raw processors, one for the control-plane and one for the data-plane. The design has four independent forwarding columns, one for each linecard.

2.2 Control-Plane

The control-plane is responsible for all aspects of router management. The Raw Router uses a dedicated Raw processor for the control-plane. Router management entails running routing protocols, such as BGP [6] and OSPF [5], to maintain the forwarding tables. The control-plane must process all packets destined to the router itself, such as control packets from other routers containing instructions for updating or maintaining routes. The Raw Router also uses the control-plane to process special packets prior to forwarding them, such as IP packets with options.

The architecture of the control-plane is divided into four rows, although not all rows are used to process a packet. Figure 2-2 shows an overview of the stages. The input linecards are connected to the input ports of tiles 12 through 15, in the bottom row. As soon as a linecard receives a packet, it writes it to the first static network for its routing column.

2.2.1 Demultiplexer

The first row of tiles that the packet passes through is a demultiplexer. The switches for these tiles route the packet both north and into the processor. The demultiplexer reads the packet from the static network and stores it in a buffer in the processor data-cache. The demultiplexer looks at the destination address and port to determine if the packet is to be forwarded and needs special processing or if it is destined for the router itself. If either of these conditions is true, then the demultiplexer sends the packet using the general dynamic network to the appropriate processing tile. The demultiplexer can determine the correct processing tile based on the destination port of the packet—for example, BGP listens on port 179. Based on this port, or the IP header if the packet has options, the demultiplexer consults a table that indicates the router's current control configuration and which processing tiles are running which services. If the packet does not require special processing or is not destined for the router, then the demultiplexer can discard the packet.

2.2.2 Processing Tiles

The control plane has eight tiles that are able to run different routing protocols and packet processing routines. The switches on all of these tiles are configured to route all data to the north. The processes to be run on these tiles are completely customizable. The tiles register their services with the demultiplexer tiles, and then receive their appropriate packets over the dynamic network. If a processing tile needs to send a packet—for instance, a BGP `UPDATE` or `KEEPALIVE` message—then the processing tile can send the new packet to any of the queuing tiles over the general dynamic network. Similarly, if the processing tile receives a packet that is destined for another host, such as an IP packet with options, then the processed packet is also sent to a queuing tile using the dynamic network.

2.2.3 Queuing Tiles

The topmost row of tiles queue packets for the data-plane. The packet queue is kept in the processor data-cache, to ensure fast reads and writes. Each queuing tile receives every packet sent to its input port on the router. Before enqueuing a packet, the tile performs the same check as the demultiplexer on the packet header. If the packet is destined for the router itself or requires further processing, then the tile discards the packet, since the demultiplexer had already sent it to the appropriate processing tile. Periodically, the queuing tiles poll the general dynamic network to see if any of the processing tiles have packets to be sent.

The queuing tiles dequeue packets and send them north over the first static network into the data-plane. They also establish the streaming memory transactions with the packet payload memory buffers, as described in section 2.3.5.

Figure 2-3 summarizes the communication patterns used by the control-plane. This diagram illustrates that for the common case—a normal IPv4 packet destined for another host—the packet is sent straight through the control-plane on the first static network. Forwarding path ensures that the control-plane adds minimal latency for common packets and that it does not negatively impact the throughput of the router.

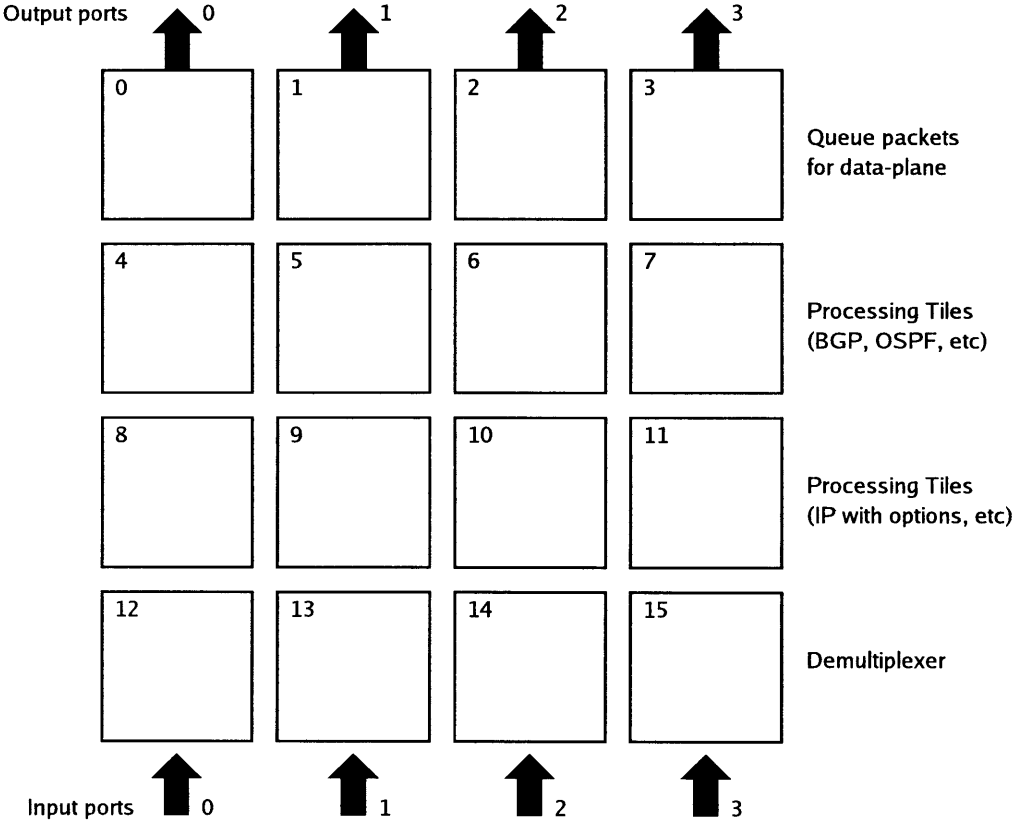

Figure 2-2: **Raw Router Control-Plane Overview.** The Raw Router control-plane is designed with four independent forwarding columns—one for each linecard. The first row of tiles demultiplexes control packets and sends them to the appropriate processing tile, in the second or third row. The last row queues packets for the data-plane.

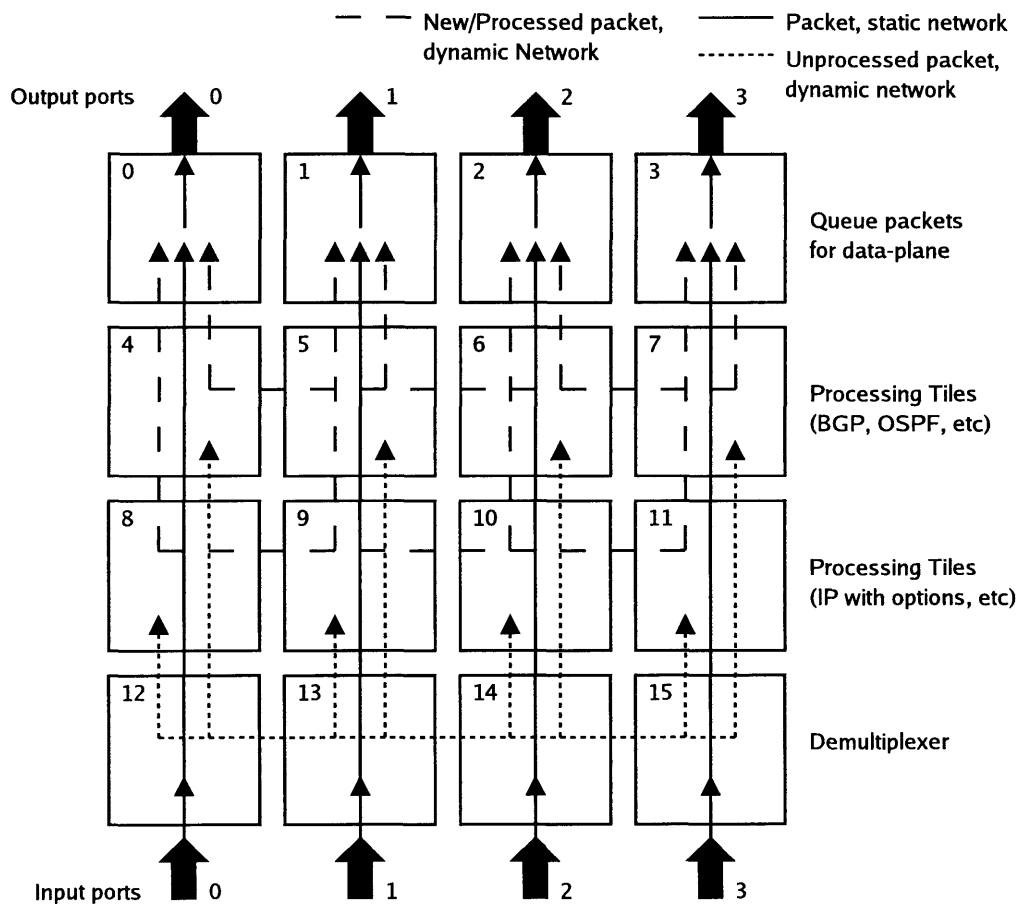

Figure 2-3: **Raw Router Control-Plane Communication.** All packets are forwarded through the control-plane using the first static network to the queuing tiles. The queuing tiles decide to queue or discard the packet, based on the destination address. Queued packets are sent to the data-plane. The demultiplexing tiles send packets over the dynamic network to processing tiles. If the processing tiles need to send new or processed packets, they are sent to a queuing tile using the dynamic network.

2.3 Data-Plane Version III

This section presents a detailed discussion of the final version of the router data-plane. The design is presented in chronological order by the four processing stages that each packet passes through, followed by the mechanisms handling the packet payload input and output.

The basic design of the the Raw Router data-plane is a four stage pipeline, with four independent forwarding paths. Each forwarding path processes packets for one linecard. In the 4x4 Raw tile grid, the forwarding paths are columns of tiles, and the pipeline stages are rows of tiles. Figure 2-4 shows the layout of the input and output ports with respect to the Raw processor.

All four forwarding columns process packets simultaneously and at the maximum possible rate—if one forwarding column is stalled, it will not effect the other forwarding columns. There are only four physical linecards, but they have separate input and output queues that operate independently.

2.3.1 Stage 1: Table Lookup

The first pipeline stage performs the forwarding table lookup. There are two copies of the forwarding tables, which are stored in SDRAMs connected to tiles 12 and 15. One copy of the forwarding table is designated as the *current* copy, and the other is the *pending* copy. The Stage 1 tiles maintain a pointer indicating the current table, and all lookups are done from this table.

When the router control processor determines that a forwarding table update is necessary, for example in response to a BGP `UPDATE` message, it updates the *pending* table. Once the update has been written to memory, the control processor notifies the Stage 1 tiles to change their table pointers to point to the updated table and to invalidate their caches. As the updated *pending* table is now the *current*, the former *current* table becomes the new *pending* table and will receive the next update. However, the control processor also must propagate the last update to the new pending table, so that it is synchronized with the new current table.

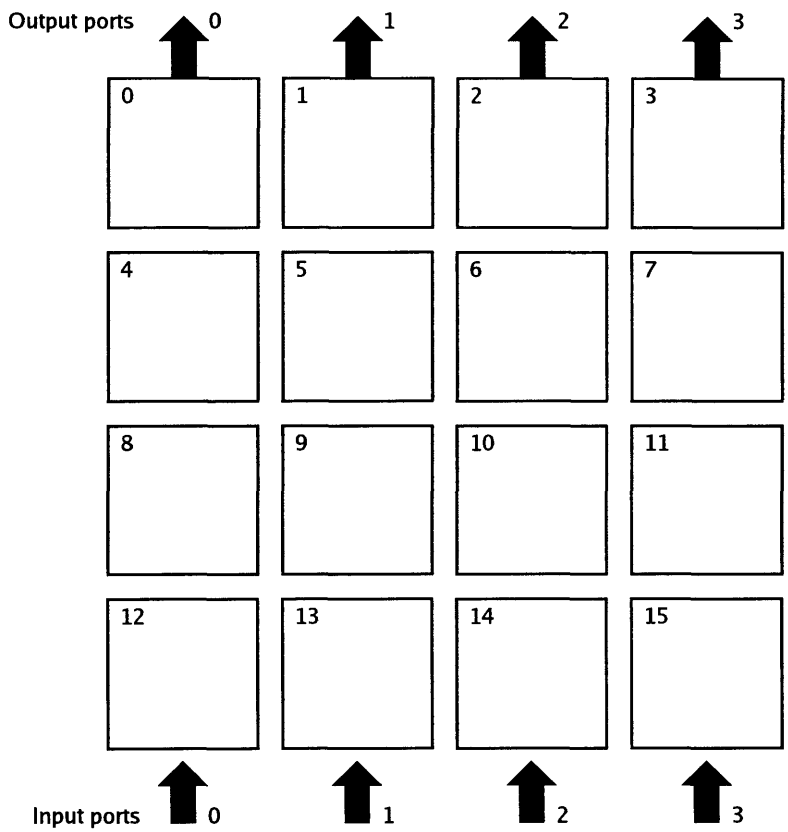

Figure 2-4: **Raw Router Data-Plane Overview.** The Raw Router data-plane is designed with four independent forwarding columns—one for each linecard—each of which has four pipeline stages.

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 input queue in the control-plane over the first static network, which signals that the control-plane queue should begin sending the packet. The control-plane queuing tile 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. (See Section 2.3.5 for a description of how the packet payload is handled.) The Stage 1 tile's switch simultaneously routes

these words into the processor and north to the next stage. If there are less 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 next hop. 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.

The destination IP address of the packet is the fifth data word of the header. This destination IP address is the value used to perform the table lookup.

We use a two-tiered forwarding table layout proposed by Gupta et al. in [2], specifically, their *DIR-24-8-BASIC* scheme. This design has two tables, both stored in SDRAM, and makes a trade-off 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 less than 24 bits long, then the route is duplicated in the table to span all possible indexes. For example, if the prefix is 192.168/16, then there will be 256 entries (all of which have the same destination route), spanning from 192.168.0 to 192.168.255. 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 over the dynamic network

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. Figure 2-5 shows the communication for the first pipeline Stage. Finally, the Stage 1 tile updates the memory address for the next packet payload, and sends it to the control-plane queuing tile, indicating that it is ready to receive the next packet.

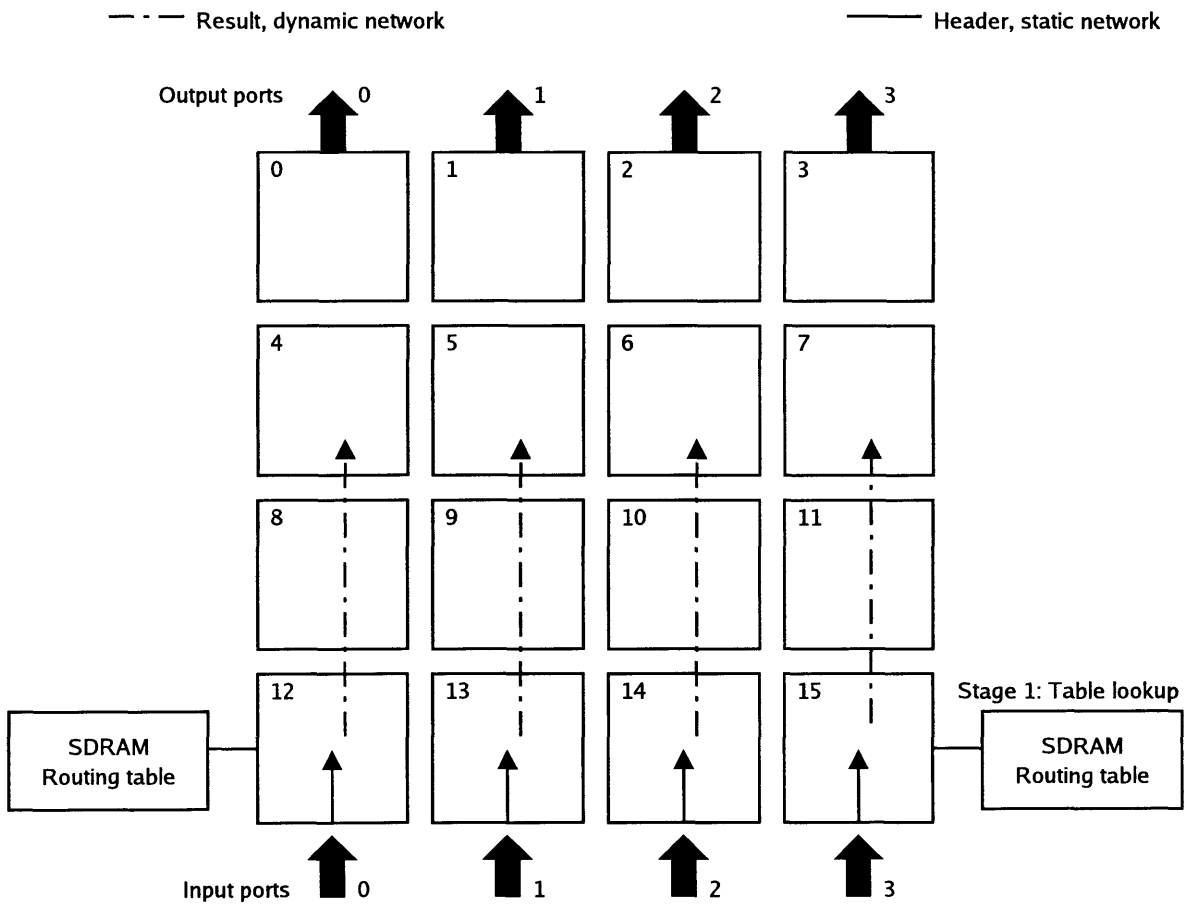

Figure 2-5: **Forwarding Stage 1: Table Lookup.** The first pipeline stage tile performs one or two memory accesses, depending on the route prefix length, in a forwarding table to determine the output port for the next hop. The output port, along with the packet payload memory address, is sent to Stage 3.

2.3.2 Stage 2: Compute Header Checksum

The second pipeline stage computes the IP packet header checksum. A second stage tile receives 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 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 over the dynamic network to the Stage 3 tile in the same forwarding column. Figure 2-6 shows the communication for the second pipeline stage.

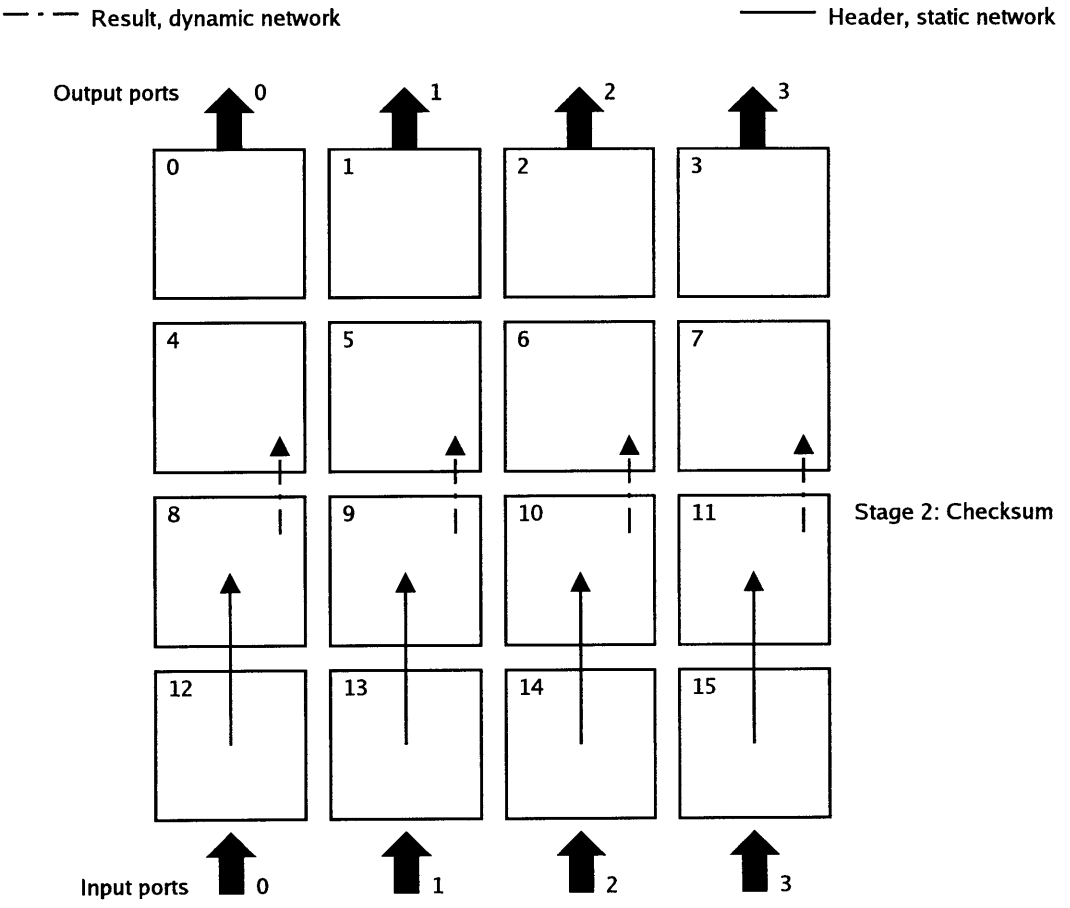

Figure 2-6: **Forwarding Stage 2: Compute Header Checksum.** The second pipeline stage tile computes the header checksum. The result, which is a boolean value indicating whether the checksum was valid, is sent to Stage 3.

2.3.3 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 routing of the header. The third stage receives the sixteen header words over the first static network from the second pipeline stage. However, unlike the first two stages, a third stage tile does not forward the header directly on to the fourth stage tile in its forwarding column. Additionally, the third stage tile receives the output port and payload memory address from Stage 1, and the checksum result from Stage 2, both over the dynamic network. 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.

The third pipeline stage updates the time-to-live (TTL) of the packet. A Stage 3 tile reads the TTL field, decrements it, and writes the value back to the packet header. Because it has changed the bits in the header, it must also recompute the header checksum and update the checksum bits.

Once the TTL has been updated, the packet header is ready to be routed to the next hop. Before performing the routing, the third pipeline 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 over the dynamic network from the first pipeline stage. The Stage 3 tile creates a new dynamic message destined to the Stage 4 tile output queue connected to the output port. This message contains the sixteen packet header words and the packet payload address. Figure 2-7 shows the communication patterns for the third pipeline stage. 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.

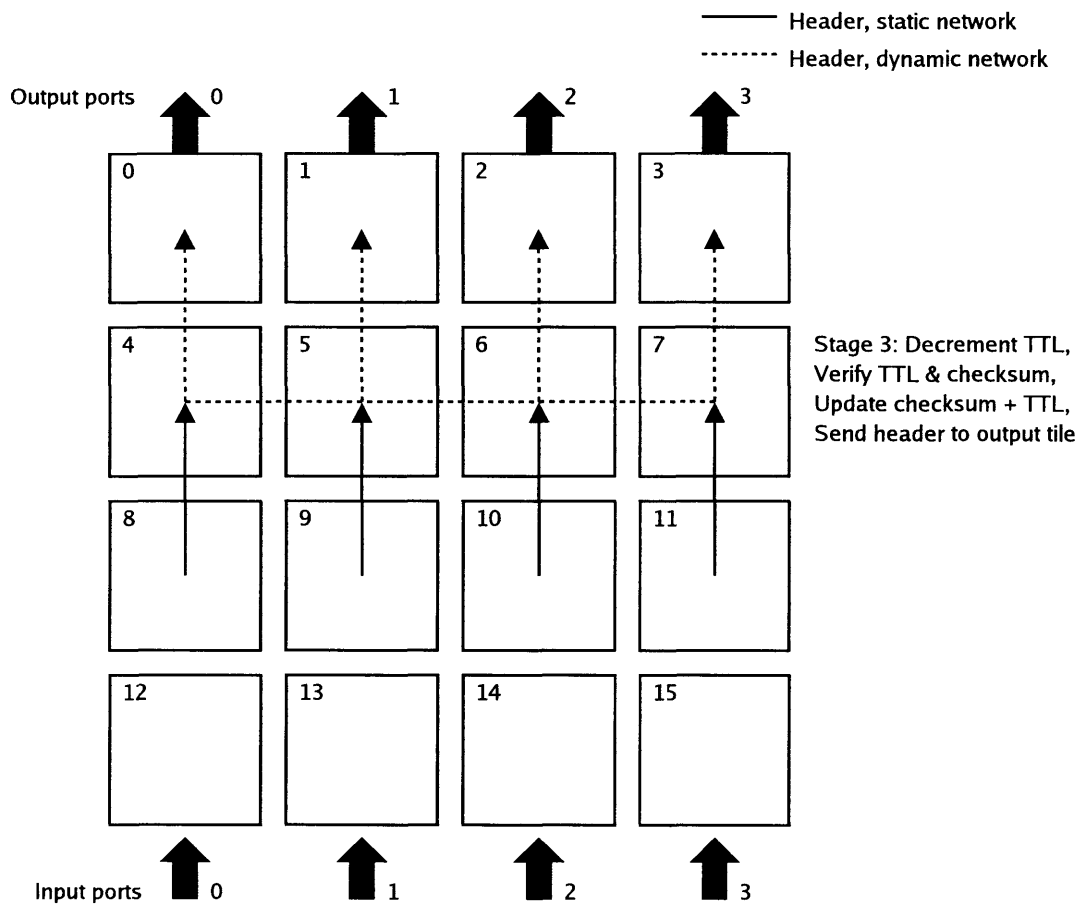

Figure 2-7: **Forwarding Stage 3: Decrement TTL, Update Header, Send to Output Queue or Discard.** The third pipeline stage tile decrements the time-to-live (TTL) field, recomputes the checksum, and updates the header with the new TTL and checksum. At this point, the header is ready to be sent to the next hop. The Stage 3 tile sends the processed packet header, along with the packet payload memory address, over the general dynamic network to the output queue based on the lookup result from Stage 1, or it discards the packet if the header checksum was invalid or the TTL reached 0.

2.3.4 Stage 4: Queue Header for Output Linecard

The fourth pipeline stage queues packet headers and payload addresses for the output linecards. When the packet header queue is empty, a stage four 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 so, this packet header 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 forwarding column until attempting to send to the output queue.

The output linecard connected to a stage four tile indicates that it is ready to receive the next packet by sending the tile an interrupt. The interrupt does not cause the tile to send the next packet immediately, but rather the interrupt handler sets a flag that indicates that the linecard is ready for the next packet. The stage four 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. Figure 2-8 shows the communication for the fourth pipeline stage (the linecard interrupt is not shown).

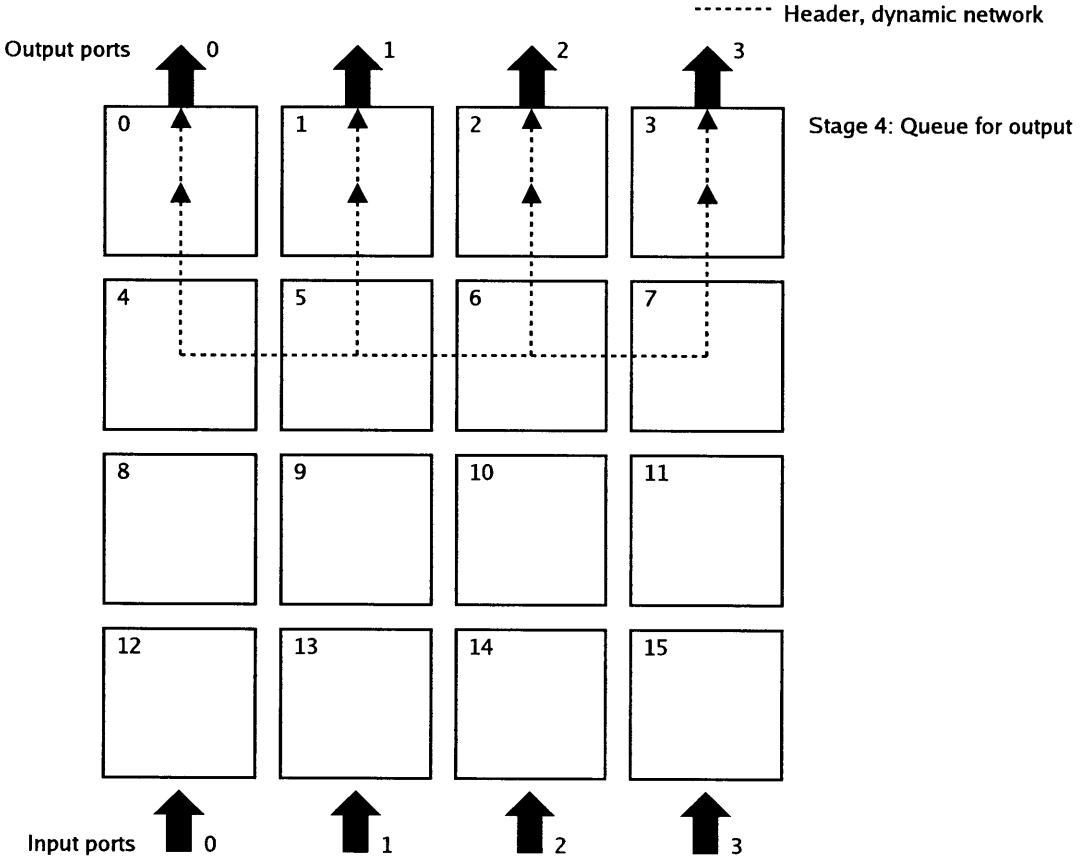

Figure 2-8: **Queue Header for Output Linecard.** The fourth pipeline stage receives packet headers from the third stage, and buffers them in a queue. When the corresponding linecard is ready to send the next packet, it interrupts the tile. The tile then dequeues a header and sends it to the linecard using the general dynamic network.

2.3.5 Payload In

The packet payloads are sent directly from the control-plane queuing tiles to SDRAMs. As described in Section 2.3.1, each tile in the first pipeline stage sends a memory address for the packet payload to its control queuing tile. The queuing tile then initiates a streaming memory transaction with its dedicated SDRAM using the first static network. Streaming memory is a memory interface supported by the Raw memory controllers, in which the processor or a device sets up a transaction with the memory, indicating that it wishes to write an arbitrary given number of data words to a given address. The memory controller is then able to write one word of data per cycle for the transaction. Streaming transactions are used to support efficiently writing large chunks of data using the static network. The use of streaming memory for the packet payloads is a distinguishing feature of the Raw Router. Most routers process packets in flits, or smaller sized blocks of data. While section 2.3.6 describes how Raw uses these to retrieve the packet payload, the packet does not need to be fragmented into smaller blocks when it is written to the buffer. Figure 2-9 shows the placement of these payload SDRAMs and the static routes.

The control queuing tile sends the address given to it by the Stage 1 tile, as well as the length of the packet payload, aligned to a cache-line boundary. The first static network is configured so that it will route the payload to the correct memory. As the payload length passes through each network switch, the switch processor loads this value into a register. This register is then used to count the payload words—the switch decrements the value for ever word that it routes, and when the counter reaches zero, the switch branches to begin routing the next packet header or payload.

Once the streaming transaction has been initiated, the control queuing tile writes the entire packet payload over the first static network. The static network routes the payload to the SDRAM, and the memory controller stores the data. The memory is allocated as a circular buffer, which is large enough to store the maximum number of packets that router can process simultaneously.

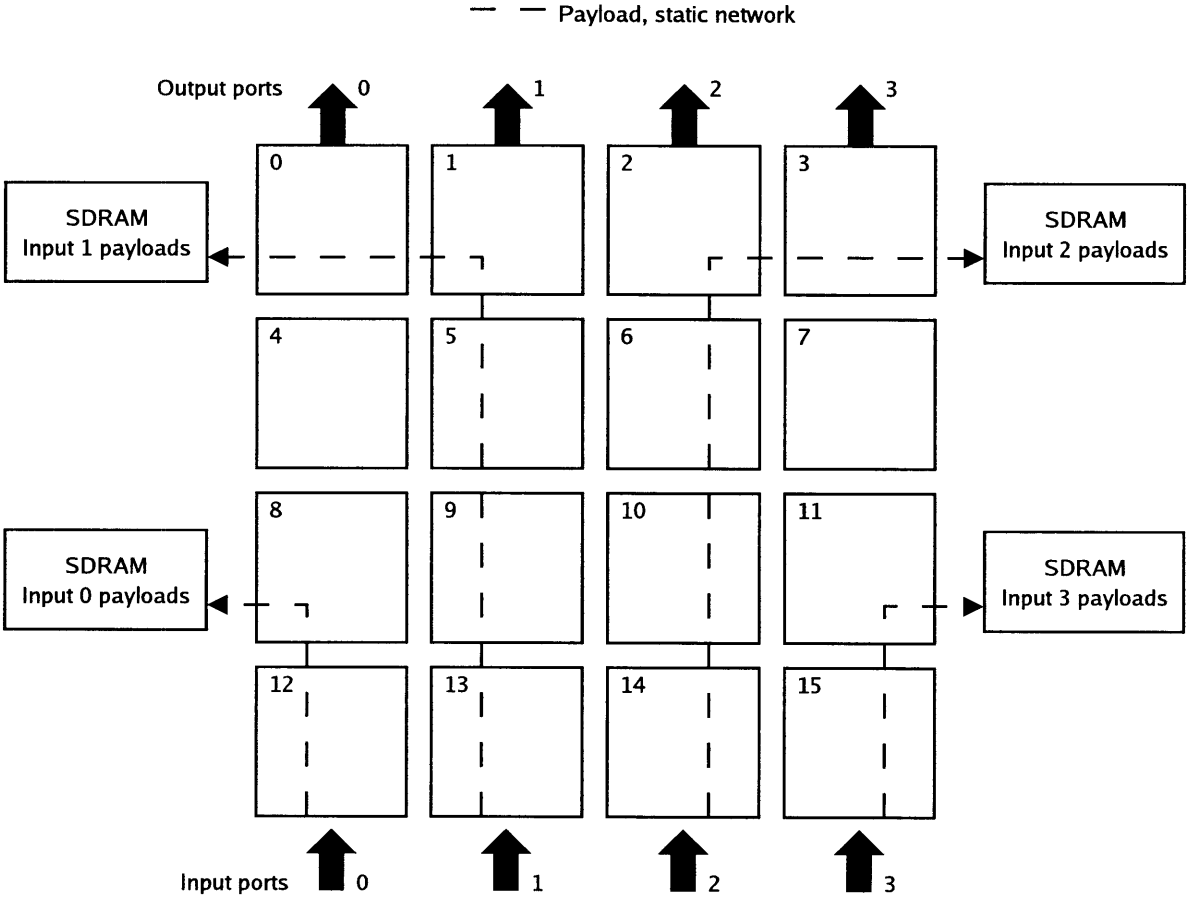

Figure 2-9: **Payload In.** The packet payloads are sent directly from the input control-plane queuing tiles over the first static network to four SDRAMs using the streaming DRAM interface. These routes are designed so that there is neither network nor memory-bandwidth contention.

2.3.6 Payload Out

The output linecards receive packet headers, along with the payload addresses, from their fourth stage queuing tiles over the general dynamic network. The output linecards directly retrieve the packet payload from memory using the memory dynamic network. The memory controller interface only supports streaming access to memory when the static network is used. Given that the output linecards cannot know which SDRAMs will hold the packet payloads until they receive the header, static routes cannot be pre-configured, as is done for the input payloads. The overhead of reconfiguring the static network for every packet payload is too expensive, so the streaming interface is not used. Instead, the output linecards use the standard memory interface, which supports transactions of eight word (64-byte) cache-line-sized blocks. Although the amount of memory per request is fixed at 64-bytes, the memory controllers support pipelined requests, with up to four outstanding memory requests. The output linecards thus send pipelined requests at the maximum possible rate. These requests and their responses are sent over the memory dynamic network.

Even with the use of pipelined requests, the retrieval of packet payloads to the output linecards is the bottleneck step in the router for packets with large payloads. The delay of this bottleneck is the longest if multiple linecards need to receive their payloads from the same memory controller simultaneously. Figure 2-10 shows the communication patterns used by the linecards for loading the payload from memory. As seen in this figure, the memory network acts as two busses, between the memories attached to tiles 0 and 3, and tiles 8 and 11.

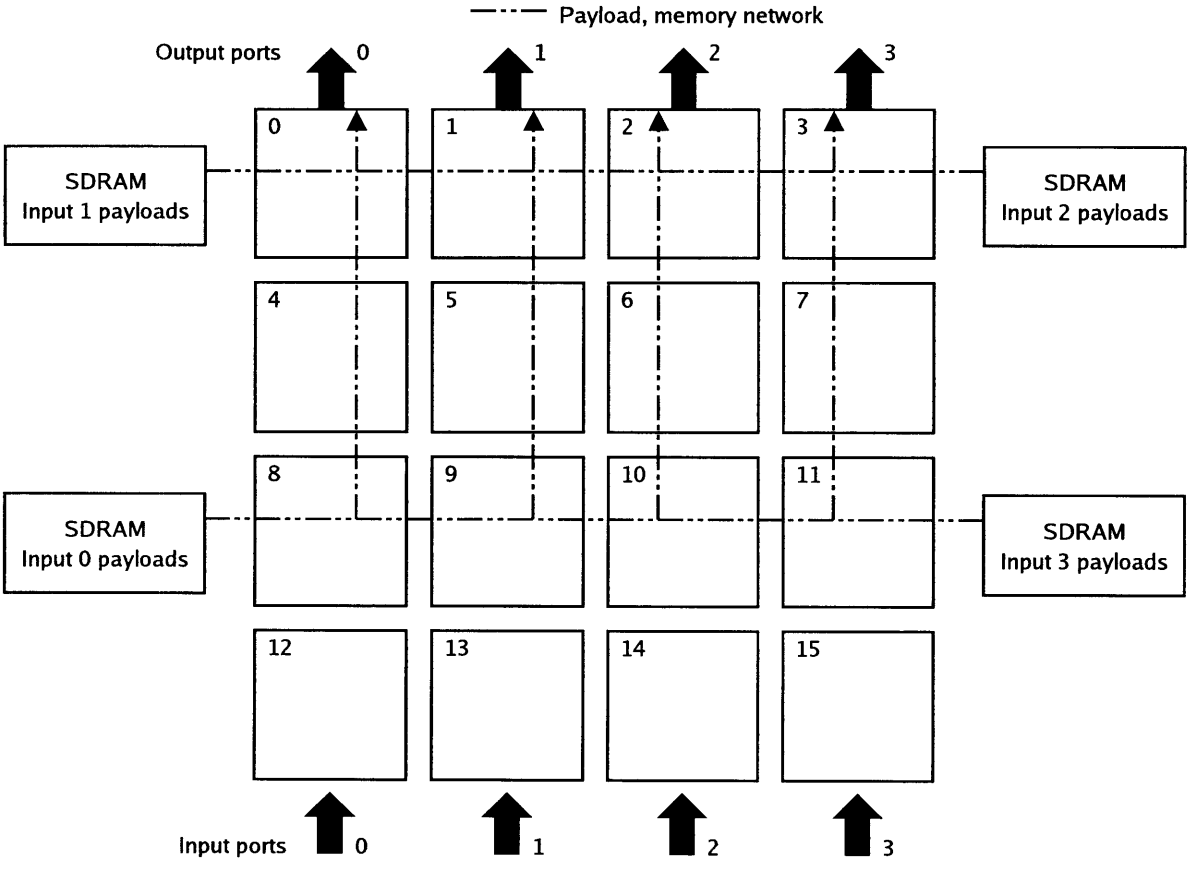

Figure 2-10: **Payload Out.** The output linecards request the packet payloads from memory using standard memory transactions over the memory dynamic network.

2.4 Data-Plane Version I

This section summarizes the design of the initial version of the Raw router data-plane. The first design also has the same four stage pipeline: lookup, validation, update, and forwarding. The pipeline stages were organized in *vertical* columns of tiles, with four forwarding *rows*. That is, the basic design from figure 2-4 is rotated 90 degrees counter-clockwise. The packets are streamed to the first three stages of the pipeline over the first static network. The first stage performs the route lookup by cache missing to a forwarding table in an external DRAM located adjacent to the column. The other stages are essentially the same as in Version III.

The Version I design sends messages with results of the checksum validation and destination port over the static network. Packet payloads are sent between the third and fourth stages of the pipeline using the general dynamic network. In this design, the first eight words of the packets are sent into the pipeline, and the remaining packet data is sent directly to two external DRAMs, attached to tiles 1 and 13, for storage until the packet is ready to be sent by the linecard. The output linecards retrieve the packet payloads using a DMA protocol.

The orientation for this design was chosen so that we could use the standard Raw DMA protocol for handling the packet payloads.

There were several problems with the first version of the design. First, there was contention for the two memory banks that stored the packet payloads. These two DRAMs did not have enough bandwidth to support reading and writing all of the packet payloads simultaneously. The memory controllers were half-duplexed, such that they were limited to performing either a read or a write request, but not both simultaneously. The second major problem in the Version I design was that two busses were formed along the general dynamic network in the fourth stage and the memory dynamic network in the third stage. Raw's dynamic networks are dimension-order routed, with the x dimension first. Thus, the fourth stage tiles had contention trying to send the packet headers to the correct output queue. Similarly, there was severe contention along the third stage, as both the input and output linecards were

sending memory requests and attempting to read responses along the memory network connecting these tiles.

2.5 Data-Plane Version II

The section summarizes the major differences between Version I and Version II of the Raw router design. One of the major changes in the second version was rotating the design 90 degrees, so that it is oriented as shown in figure 2-4. Now, the results from the third stage are sent across the third stage dynamic network routers, as shown in figure 2-7, instead of causing contention for the dynamic network in the fourth stage.

The memory contention was addressed with several changes. First, memory was changed to use SDRAMs, which are full-duplexed, allowing them to read and write a cache-line simultaneously. Second, the number of memories was doubled, employing a total of four, connected to the same tiles as in Version III. The placement of these SDRAMs eliminates the contention for writing the packet payloads from the input linecards.

Other changes concern the use of the static network. Instead of using the static network to send the results from the first two stages, these results are sent using the general dynamic network. Instead of only sending the first eight words of the packet into the pipeline, this design sends the first sixteen (64-bytes), which is the maximum size of an IP header.

The second version of the router is very similar to Version III. The main throughput bottleneck is the use of the DMA protocol, which limits requests to a single eight word cache-line at a time. These requests must be completed serially, in that a second read cannot be issued until the first is satisfied.

2.6 Full Version III Router Data-Plane

The major changes between Version II and Version III were to increase the throughput for large packets by using more efficient memory transactions. For the input packets,

Version III uses a streaming memory transaction instead of the DMA protocol. The throughput for output packets was increased by using pipelined memory requests. The forwarding rate was also increased by eliminating several unnecessary delays within the data-plane. For instance, in Version II, Stage 3 would send a notification word to Stage 1 once it had forwarded the current header. Stage 1 would block until it received this notification before requesting the next packet from the input port. This communication is unnecessary, as the static network will simply cause the processors to stall if Stage 3 is congested.

Figure 2-11 shows all of the communication patterns for the data-plane in Version III. The design and orchestration of these communication patterns and the placement of functionality within the data-plane are two of the major contributions of this thesis. In the final version of the router, we have eliminated communication contention between different stages of the router, except for the sending of the headers from the third to fourth stage. The streamlined communication patterns allow the router to achieve its high performance.

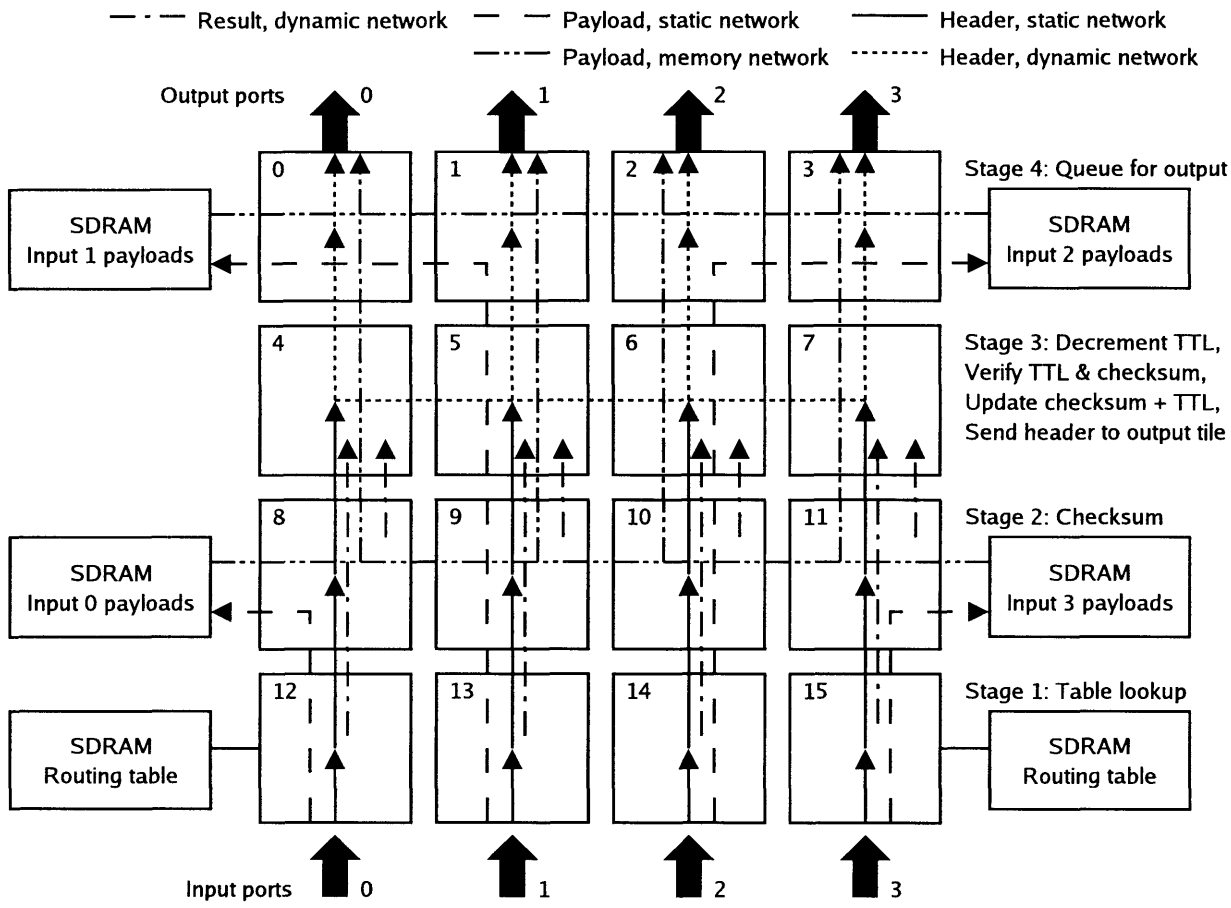

Figure 2-11: **Complete Router Data-Plane.**

Chapter 3

Evaluation

This chapter discusses our implementation of the Raw router and presents our performance evaluation and analysis. The evaluation compares the forwarding rates and throughputs of all three versions of the router and the Intel IXP1200. Finally, we analyze the latency and a breakdown of the time required to process each packet.

3.1 Implementation

The Raw Router was developed using a validated cycle-accurate simulator of the Raw chip. We chose to use the simulator to develop our prototype router, as opposed to the actual Raw hardware, for two reasons. First, the simulator gives us complete flexibility for the placement of devices and memories around the Raw processor. The fabricated Raw board has restrictions on which Raw pins are connected to memories and which pins are connected to external devices. This is why the original Version I design was oriented so packets flow horizontally—this orientation accommodated the placement of memories supported by the actual hardware. However, since our goal was to explore different router architectures and we wanted to see how fast we could push the router, we decided to change our design to use memories that are not supported by the current Raw hardware board, but could be supported in a future revision.

The second advantage of the simulator is that it supports simulation of arbitrary devices attached to the Raw processor. We used this device simulation support to

implement our high-performance linecards. Although the Raw board does have a PCI bus, PCI devices are not yet supported. Using simulated linecards allowed us to focus on the design of the router itself, without worrying about the interfaces to the linecards. Furthermore, by using simulated linecards we were able to explore the performance advantages of linecards that are aware of features supported by streaming processor architectures, such as using a streaming memory protocol instead of DMA transactions.

3.1.1 Control-plane

In our initial implementation of the Raw Router, the control-plane is not used. For this implementation, the input linecards are connected directly to the southern-edge of tiles 12-15 of the data-plane. The forwarding tables are statically initialized when the router boots.

3.1.2 Data-plane

As the control plane has not been used in this implementation, the forwarding tables are statically-initialized. Because the router cannot change its routing table without the control plane, only a single forwarding table is used, as opposed to the two tables—one of which accepts the next update—as described in section 2.3.1.

The Raw router data-plane was written in C code for the tile processors and assembly code for the switch processors. The entire data-plane 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.

Aside from the differences explicitly mentioned above, the implementation conforms to the design presented in Chapter 2.

3.2 Experimental Methodology

The evaluation for this thesis makes use of a validated cycle-accurate simulator of the Raw chip. Using the validated simulator as opposed to actual hardware allows us to explore alternative motherboard configurations and use simulated linecards. [10] further describes this simulator. Our 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 our timing results, we use a Raw microprocessor clock speed of 425MHz.

The input linecards send UDP packets at specified rates. We used two types of input packet files: randomly generated and captured traces. 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 are randomly assigned to an input and output port, respectively. For each evaluation, we ran the same configuration four times with four different random traces, and averaged the results. We found that there was no difference in performance when using traces longer than 4000 packets or using more than 128 addresses. The forwarding table is initialized with 32-bit entry prefixes, that is, each address has its own entry in the forwarding table. 32-bit prefixes also require two memory references for each table lookup, which is the worst-case performance.

The captured trace files [1] each contain 10,000 packets. For the live traces, the forwarding table is initialized with randomly assigned 24-bit entry prefixes, which more accurately reflect actual routing table entries. The results for the live packet traces are also averaged from four different traces, each containing 10,000 packets. For the live packet traces, we ignored the timing data included in the traces, and

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

configured the linecards to send the packets at the maximum rate that the router would accept them.

Our evaluation measures the forwarding rate, throughput, and latency of the router. The forwarding rate is measured using 64-byte packets, which are minimum-sized TCP packets. Small packets place more demand on a router's CPU and other bottleneck resources than large packets, because the CPU is used in proportion to the number of packets forwarded, not in proportion to bandwidth. However, the throughput is also an important measurement, because it stresses other router resources, such as internal bandwidth. Throughput is measured by using 1500-byte packets, which are the maximum size of an Ethernet packet. Finally, latency is measured to determine how much time the router delays the packet in the forwarding process. The latencies for both large and small packets are measured. 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.

3.3 Forwarding Rates

Perhaps the most important performance measurement of a router is its forwarding rate. The forwarding performance is 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 maximum loss-free forwarding rate (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 performance of the Raw router and of the Intel IXP1200 network processor for IP routing. The Intel IXP1200 results were collected using

the IXP1200 simulator included with Intel IXP Developer Workbench. We used the example router provided by Intel, which has two 1Gbit linecards. To measure the performance of the IXP1200, we used the sample 64-byte packet streams provided with the IXP simulator. The IXP simulator did not allow us to set the rate at which packets are sent to the router, so we plot the saturation point—which is 2,900,000 packets per second— and interpolate the rest of the curve.

For the Raw Router, we used 64-byte packet traces and measured the output rate as the offered load was increased in increments of 100,000 packets. The maximum loss-free forwarding rate of Raw is 9,455,000 packets per second. Raw's peak forwarding rate, attainable with bursty senders, is 9,535,000 packets per second. When the offered load exceeded Raw's forwarding capabilities, the linecard would stall until the forwarding column was able to receive the next packet. The bottleneck for minimum-sized packets is either the routing table lookup or computing the header checksum, and is discussed in more detail in section 3.5.

Figure 3-2 shows the performance of several variations of the Raw Router. The "null" line shows the performance of the router which does no processing. The checksum and TTL are not computed and the results are always successful, and the output port is always the same as the input port. The MLFFR for the null router is 15.5 million packets per second and provides a baseline for the maximum possible forwarding rate with no processing or switching overhead. The "IPv4 version III" line shows the performance of the final, optimized implementation of the router, and is the same plot included in figure 3-1.

Two other points of comparison are routers built with a Pentium III. One measurement is for the Click modular router [3]. The other measurement is for Linux IP routing modified to poll the network interfaces cards instead of taking interrupts on a packet arrival. Click has a MLFFR of 333,000 packets per second, and polling Linux has a MLFFR of 284,000 packets per second. Both of these results are taken from [3], which reports that the results were measured using a 700MHz Pentium III and 100Mbit PCI Ethernet controllers. Table 3.1 provides a summary of the maximum loss-free forwarding rates for the various routers and versions of the Raw router.

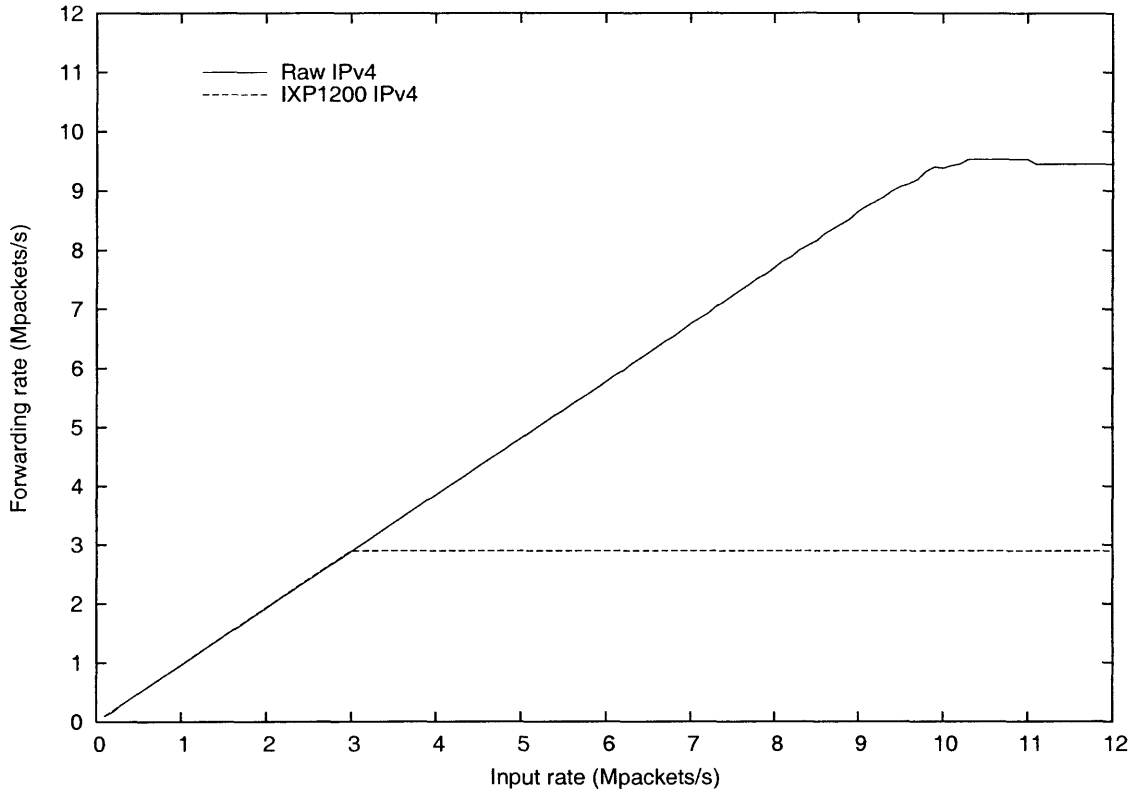

Figure 3-1: **Raw and IXP1200 Forwarding Rates Comparsion.** Forwarding rate as a function of input rate for the Raw and IXP1200 routers (64-byte packets).

3.4 Throughput

Although small packets best measure the processing capabilities of a router, the performance for routing large packets is another important metric. Large packets stress a router's internal bandwidth and its ability to move data from its input ports to its output ports.

Table 3.2 shows the throughput when routing 1500-byte packets for the optimal and "null" versions of Raw router, as well as the IXP1200 router with two 1Gb linecards. The Raw IP router has a sustained throughput of 13.74Gb/s. The bottleneck of the router for large packets is the transferring of the packet payload into and out of the SDRAM buffers. The throughput for the captured packet traces is 9.34Gb/s. The throughput for the captured traces is expected to be lower than the peak throughput, since the router must consume resources processing many small

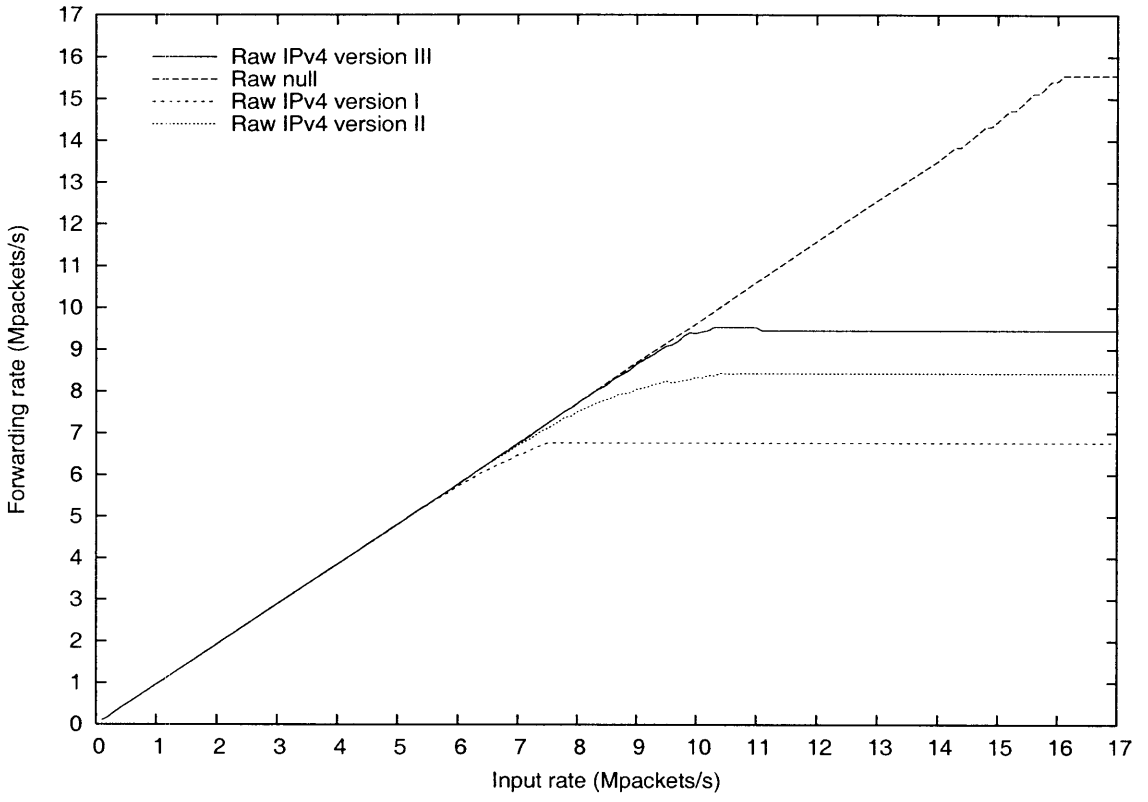

Figure 3-2: **Raw Router Forwarding Rates.** Forwarding rate as a function of input rate for the several implementations of the Raw router (64-byte packets).

packets.

Table 3.2 also illustrates the substantial improvements made to the throughput with the various enhancements over Version I and Version II. The improvement from Version I to Version II is largely due to the elimination of contention on the memory networks. The further gain of over 5Gb/s 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.

3.5 Latency

Table 3.3 shows the latency, in both cycles on the Raw processor and the corresponding time in nanoseconds, for the Raw IP router and the "null" implementation. These

Router	**Forwarding Rate** (Mpackets/s)
Raw null	15.5
Raw IPv4 version III	9.45
Raw IPv4 version II	8.4
Raw IPv4 version I	6.7
IXP1200 IPv4	2.9
Linux Click IPv4	0.33
Linux polling IPv4	0.28

Table 3.1: **Forwarding Rates Summary.** A summary of the maximum loss-free forwarding rates of 64-byte packets for several routers.

Router	**Throughput** (Gb/s)
Raw null	20.82
Raw IPv4 version III	13.74
Raw IPv4 version II	8.66
Raw IPv4 version I	5.96
IXP1200 IPv4	1.9

Table 3.2: **Throughput Comparsion.** A measure of sustained throughput for 1500-byte packets.

measurements were made from Raw cycle count at the beginning of each forwarding stage, and summing the results for all the stages. For 64-byte packets, the Raw IP router adds only 293ns of latency. As a comparison, the Click router has a latency of 2798ns for a 64-byte packet [3].

The difference between the latency for 64-byte and 1500-byte packets shows that moving the packet payload to and from memory is the router bottleneck for large packets. These results show that there is additional overhead for processing the packet payload in addition to just moving the data. There is a difference of 1436 bytes that need to moved into and out of memory. 1436 bytes equals 359 4-byte words. Because it takes one cycle to read and write each word, it should take an additional 718 cycles to read and write these extra bytes. However, the 5394 cycles needed to process a 1500-byte packet is much more than 718 cycles more than the 690 required for a 64-byte packet. The extra 3986 cycles represent overhead from the memory transactions, the memory latency, and contention on the memory network.

Router	Packet size	Cycles	Time(ns)
Raw null	64	416	177
Raw IPv4	64	690	293
Raw null	1500	3490	1483
Raw IPv4	1500	5394	2292

Table 3.3: **Raw Latency.** Latency of the Raw router for 64 and 1500 byte packets.

In particular, the memory dynamic network used by the output linecards to retrieve the packet payload accounts for the majority of this overhead latency, because it is not able to use the streaming memory interface used by the input linecards.

Stage	Task	Cycles	Time(ns)
1	Forwarding table lookup $\leq$ 24bit prefix	29/ $\sim$ 100	12/ $\sim$ 42
1	Forwarding table lookup $>$ 24bit prefix	36/ $\sim$ 150	15/ $\sim$ 64
2	IP header checksum	78	33
3	Update TTL and checksum	26	11
4	Dequeue and send to output linecard	22	9

Table 3.4: **Raw 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. The time taken to satisfy a cache miss varies based on the tile's distance from the memory; the averaged value is shown.

Table 3.4 further breaks down the cost of forwarding a packet through the Raw router. This table shows that for small packets, the bottleneck stage is either the table lookup or IP header checksum, depending on whether the cache-line(s) containing the result for the table lookup are already in the data-cache. The address prefix length for the lookup does not impact the latency if the relevant portions of the forwarding table are already cached. However, if the table entries are not cached, then the lookup takes fifty percent longer for a long prefix, and is also the bottleneck stage.

Chapter 4

Future Work and Conclusion

This chapter discusses future research possibilities for further increasing the performance and functionality of the Raw router, and then concludes.

4.1 Future Work

Prefetch Payloads in Stage 4

The bottleneck for large packets is reading the payload from memory using the output linecards. While the linecard is reading the payload, its Stage 4 queuing tile is blocked, simply polling the dynamic network for new headers from Stage 3. Instead, the Stage 4 tile could begin to prefetch the payload for the next packet in the queue into a local buffer in the processor data-cache. Then, when the linecard is ready for the next packet, the Stage 4 tile can send it the *entire* packet, not just the header.

The hard part of this optimization is ensuring that a Stage 4 tile does not slow down the router by interfering with a read being performed by another linecard to the same memory. Otherwise, the tile would create contention at the memory controller. The tile would have to ensure that no other tiles or linecards are currently performing a read from the desired memory before beginning the prefetch. Similarly, the tile should be interrupted if either its linecard completes sending its packet before the prefetch is completed or if another linecard wants to begin a read from the same

memory controller.

Concurrent Packet Processing in Output Linecards

If the output linecard is able to read from the memory network and either the general dynamic network or the second static network in the same cycle, then it would be possible to have the output linecards process large and small packets concurrently. While the output linecard is busy reading a large packet payload on the memory network, the queuing tile could send it payload-less packets from its queue using either the general dynamic network or the second static network. In addition to increasing the forwarding rate for mixed sized packets, this would greatly reduce the latency for small packets queued behind several large packets.

Note that this optimization is largely incompatible with the previous one proposing prefetching packet payloads.

Stage 4 Queuing and Dropping Packets

Evaluations should be done to determine how long the Stage 4 queues should be to guarantee an acceptable amount of latency. Currently, the router does not drop packets when it is congested—if the queues in Stage 4 become full, then the network buffers will fill and eventually cause the processors to stall. However, dropping is implemented in Stage 4—it is just not enabled. Further evaluations can determine the effect on performance and latency if the router drops packets when the Stage 4 queues become full.

Stage 3 Queuing

Another optimization that could potentially increase the forwarding rate and throughput, and possibly the latency for some packets at the expense of others, would be to add queuing to Stage 3. If a single Stage 4 tile's queue is full, then whenever a Stage 3 tile attempts to send a header to that queue, that forwarding column will stall—even if the other queues are empty. Instead of blindly attempting this blocking

write, the Stage 3 tile could poll the dynamic network status register. If it sees that the buffer for the direction in which it needs to send is not empty, then it would know that there is possibly congestion on the dynamic network, and it could enqueue that header. Each Stage 3 could maintain four queues, one for each output port. It could then potentially send subsequent headers to an output port for which there is no contention. However, it should eventually drain the headers from the queue, regardless of whether the buffers are empty or not.

This idea should definitely be explored along with dropping packets from the Stage 4 queues, which addresses the same issue.

Control-plane

The control-plane has not been implemented. An initial version of BGP was made to run on Raw, but much work remains to integrate it into the router.

Filters - QoS

Various filters can be added, either in the data-plane or in another Raw tile, to implement other functionality, such as Quality of Service (QoS) guarantees and statistics gathering.

More than 4 Linecards

Design a router using Raw able to support more than four linecards. A naive approach would be to simply put two four-port routers side by side. This will not work, however, because the packet payloads would be in severe contention for such a design.

4.2 Conclusion

This thesis describes the design and implementation of a gigabit network router on a general-purpose parallel architecture microprocessor. The Raw router is built without any custom hardware and achieves fast forwarding rates and high throughput through

the careful programming and management, in software, of the interconnects within the Raw chip.

Our results demonstrate the great potential of parallel architectures for high-performance routers. The Raw router is more than three times faster than a router built using Intel's custom-made IXP1200 network processor. When compared to a software router built with a Pentium III, Raw is able to forward over 30 times as many packets per second. We hope that the Raw router will encourage further exploration of flexible high-performance software routers built using parallel architectures.

Bibliography

[1] F. Baker. RFC 1812 - Requirements for IP Version 4 Routers. In *Internet Request For Comments*, number 1812, jun 1995.

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

[3] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[4] John Kubiatowicz. *Integrated Shared-Memory and Message-Passing Communication in the Alewife Multiprocessor.* PhD thesis, MIT, 1998.

[5] J. Moy. RFC 2328 - OSPF Version 2. In *Internet Request For Comments*, number 2328, apr 1998.

[6] Y. Rekhter. RFC 1771 - A Border Gateway Protocol 4 (BGP-4). In *Internet Request For Comments*, number 1771, mar 1995.

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

[8] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Jae-Wook Lee, Paul Johnson, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. The Raw Microprocessor:

A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro*, pages 25–35, Mar 2002.

[9] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures. In *2003 HPCA*, pages 341–353, 2003.

[10] Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffman, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *ISCA*, 2004.