Gigabit Routing on a Software-exposed Tiled-microprocessor

Umar Saif
umar@mit.edu

James W. Anderson
jwanderson@cs.ucsd.edu

Anthony Degangi
degangi@mit.edu

Anant Agarwal
agarwal@.mit.edu

MIT Laboratory for Computer Science and Artificial Intelligence

ABSTRACT

This paper investigates the suitability of emerging tiled-architectures, equipped with low-latency on-chip networks, for high-performance network routing. In this paper, we present the design, implementation and evaluation of a continuum of software-based routers on the MIT RAW microprocessor. The routers presented in this paper explore 1) several design choices for mapping the routing functions to the RAW tiles, 2) the role and behavior of RAW on-chip interconnects for transporting and switching packets, and 3) the placement of packet buffers and their interaction with the RAW on-chip networks. Our experiments evaluate the performance benefit of streaming on-chip networks for transporting packet payloads, effect of buffering on the linecards, and the cost of scaling our design. Our software-based routers on RAW can achieve a throughput of 15Gb/sec – an order of magnitude improvement over previous software routers on traditional general-purpose architectures and at least four times faster than Intel's IXP1200 Network Processor.

Categories and Subject Descriptors

C.2.6 [**Internetworking**]: Routers

General Terms

Design, Performance

Keywords

RAW router, programmable router, tiled architecture

1. INTRODUCTION

Programmability and high-throughput, two traditionally conflicting requirements, continuously challenge the network routing industry. In order to remain competitive in a market characterized by an ever-increasing demand for throughput and functionality, network routers must provide high-throughput without confounding programmability or scalability of the architecture. However, existing routing platforms fall short of addressing the twin requirements of performance and programmability. Network routers built using custom-designed hardware are typically expensive, offer limited programmability and must be frequently redesigned to keep up with changes in routing standards, throughput requirements, and algorithmic innovations. In contrast, general-purpose architectures offer a greater degree of flexibility for upgrading and extending the router functionality, but fail to provide the level of performance warranted by high-performance network routing.

The performance of a network router depends on the careful exploitation of packet-level parallelism and the efficient flow of packet streams in the router fabric. In a traditional general-purpose architecture, not only does an application have little control over how its parallel tasks are mapped to chip resources, the only mechanism of communication between an application's parallel tasks is via sequential memory read/write operations. Therefore, the parallelism of a general-purpose architecture is typically underutilized, leading to poor performance, and exhibits diminishing returns, due to the overhead of memory-based communication. Furthermore, since each IP packet is an independent data unit, the traditional performance gain from (temporal or spatial) caching in a memory-hierarchy is irrelevant when processing IP packet streams.

However, the emerging "all-purpose" tiled-architectures, equipped with low-latency on-chip interconnection networks, promise to combine the programmability of general-purpose architectures with the performance of custom-designed network processors. Purported as the natural architectural evolution in response to the increasing significance of wire-delay, tiled-architectures replicate and distribute the chip resources as a set of interconnected clusters, or tiles, and employ structured low-latency on-chip networks to route traffic between different tiles. Tiled-architectures like the MIT RAW microprocessor [4] expose the hardware resources to the software, affording a degree of flexibility to the software in how it utilizes the chip's pin resources. Taking into account the latency of inter-tile communication, the software fully controls the partitioning and lay out its parallel tasks on the chip tiles. Such architectures, therefore, promise to offer a flexible model for exploiting the parallelism inherent in packet-based network traffic, as well as an efficient mechanism for transporting packets within the chip.

This paper explores how traditional network routing functions, implemented by custom-designed hardware in network processors, may be mapped to the architectural primitives offered by the emerging "all-purpose" tiled-architectures to achieve high-performance.

In this paper, we evaluate the performance of MIT's RAW tiled-microprocessor for network routing. We chose the RAW microprocessor since its tiled-architecture resembles several emerging network processors, such as Cisco Toaster, and, more importantly, offers a number of interesting features for network routing. First, RAW's tiled-architecture, for instance as a 4x4 grid, offers several functional units which may be programmed to exploit the parallelism afforded by Internet packet streams. Second, RAW offers four on-chip low-latency networks—two statically-routed streaming networks and two dimension-ordered-routed dynamic networks (General Dynamic Network, GDN, and Memory Dynamic Network, MDN), routed by dedicated switch-processors—for rich inter-tile communication. Finally, RAW exposes the chip resources—functional units, pins and on-chip networks—to the software, affording the software a degree of flexibility over how routing functions are mapped to the RAW's tiled-architecture. A detailed description of the RAW microprocessor is given in [10].

Our design space for network routing on RAW's tiled-architecture is characterized by the following three properties that distinguish it from previous custom-designed network processors.

Processing: Instead of using custom-designed hardware to implement header processing functions, such as lookup and verification, our router architectures program the RAW tiles to perform these functions in software.

Switching: Traditional routing architectures use a dedicated switching artifact, such as a crossbar switch. However, in our software-router, the RAW on-chip networks are used as the switching fabric, effectively combining the switching and processing of packets on a single chip.

Buffering: Often dubbed a store-and-forward switch, a network router is essentially designed around a packet buffer that holds the incoming packets waiting to be processed. In custom-designed network routers, such packet buffers are central to the design of the router, and are typically accessed via special-purpose memory controllers over dedicated high-bandwidth interconnects. In the RAW microprocessor, however, packet buffering must be provisioned externally to the chip, by connecting SDRAMs to the RAW pins, and accessed over the RAW on-chip networks.

For our evaluation, we implement a basic IPv4 network router, compliant with RFC 1812, in software. Our routing software implements four basic functions: address lookup, header verification, header (TTL) update and checksum recompute. We use the (software implementation of) two-tiered forwarding table layout proposed by Gupta et al. in [5] for lookup, specifically the popular DIR-24-8-BASIC scheme. This scheme, typically implemented in hardware, trades space for lookup efficiency and provides a fair reference point for comparison with hardware-based router implementations. The header checksum is computed by taking one's complement of the header bytes, while we implement the standard incremental checksum update algorithm, described RFC 1141, for updating the outgoing header.

The rest of the paper presents the design, implementation, and evaluation of a continuum of router architectures that explore: a) various design choices for mapping the routing functions to the RAW tiles, b) the role and behavior of RAW on-chip interconnects for transporting and switching packets, and c) the architecture of packet buffers and their interaction with the RAW on-chip networks.

2. EVALUATION METHODOLOGY

To evaluate our software-based router architectures, we use a validated cycle-accurate simulator of the RAW chip. The use of a simulator, instead of actual hardware, permitted us the flexibility to explore alternative motherboard configurations and simulated linecards. [4] further describes this simulator. Our simulated input linecards read their input from files containing packets in byte streams and have a configurable rate at which they send packets to the RAW processor. If the rate is greater than the saturation point of RAW, then the linecards simply 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.

In our experimental setup, the input linecards are programmed to send UDP packets to the routing engine at various rates. We used two types of input packet data: randomly generated and real packet traces from the Internet. The randomly generated packet data comprises 4000 packets 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. The captured Internet trace files (obtained from http://lever.ucla.edu/ddos/traces) each contain 10,000 packets. For these real traces, the forwarding table is initialized with randomly assigned 24-bit entry prefixes, which reflect actual routing table entries. For the real packet traces, we ignore the timing data included in the traces, and configured the linecards to send the packets at the maximum possible rate.

Our evaluation measures the forwarding rate, throughput, and latency of the router. The forwarding rate, perhaps the most important measure of a router's performance, is evaluated by measuring the rate at which a router can forward 64-byte packets (minimum packet size on the Internet without MTU discovery) 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. Plotting forwarding rate versus input rate indicates both the maximum loss-free forwarding rate (MLFFR) and the behavior of the router when overloaded. An ideal router would emit every input packet regardless of input rate, corresponding to the line $y = x$.

The throughput, on the other hand, is measured in gigabits per second using 1500-byte packets – maximum size Ethernet frames on the Internet. With maximum-sized frames, the throughput evaluation is dominated by the capacity of the internal bandwidth of the router i.e. switching and buffering capacity of the router.

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.

The evaluation presented in this paper is intended to be self-contained in that we establish our own baseline case instead of simply comparing with previous work. However, as a point of reference, we also compare the performance of our router with the hugely popular Intel IXP1200 Network Processor. For our comparison, we use 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 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.

Importantly, since the architectures presented in this paper are focused on evaluating the suitability of emerging tiled-architecture and on-chip interconnects for routing network packets, our evaluation is focused on what typically constitutes a data plane or the forwarding engine in a network router, rather than "control-plane" issues such as quality of service, active queue management, scheduling, signaling and route discovery.

3. SOFTWARE ROUTING ON RAW

In this section we describe the evolution of our router architecture through a series of advancements to optimize the processing, switching and buffering of packets on the RAW's software-exposed tiled-architecture.

3.1 Parallelism: RAWRouter Version I

Our first design establishes a baseline case for the performance of network routing on the RAW microprocessor. In the first version, we focus primarily on the performance afforded by the parallelism of the RAW's tiled-architecture, with little attention to optimize the switching and buffering of packets in the router.

In our baseline case, we use a 16-tile RAW grid (4 x 4) to serve four network ports. The routing software establishes four parallel processing paths, each independently mapped to a network port to enable parallel processing across incoming traffic streams. When programmed for IPv4-forwarding, each processing path is in turn divided into a four-stage pipeline to exploit parallelism between IP header processing operations. In this version, the RAW static network serves the dual purpose of streaming packets between the linecards and the packet buffers and as a broadcast channel for feeding the three stages of the header-processing pipeline.

Figure 1 shows our first design on the RAW 4 x 4 grid. The four-stage pipeline is mapped to four principal operations in IPv4 forwarding: address lookup, header validation, header update, and header forwarding. The pipeline stages are organized in "vertical" columns of tiles, with four forwarding rows. The packets are streamed (broadcast) to the first three stages of the pipeline over the RAW static-network-1. The first stage performs the route lookup by cache missing to a forwarding table in an external DRAM located adjacent to the column. The DIR-24-8-BASIC scheme used for lookup in our implementation 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 second pipeline stage computes the IP packet header checksum. 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.

The third pipeline stage, in turn, decides whether to discard or forward the packet and performs the actual routing of the header. Additionally, the third stage tile receives the output port and payload memory address from Stage 1. Finally, the third pipeline stage updates the time-to-live (TTL) of the packet. 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.

The destination of the outgoing message at Stage 3 is determined by the output port received over the static 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 packet header and the packet payload address.

The fourth pipeline stage queues packet headers and payload addresses for the output line-cards. 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 a 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 check 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 the 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.

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 (GDN). 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 2 and 14, for buffering 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.

Figure 3 and 4 show the performance of Version I by comparing its Maximum Loss Free Forwarding Rate (MLFFR) with IXP1200. This basic version performs approximately three times better than IXP1200. Importantly, the throughput of this version is simply derived from mapping the IPv4 header processing to the tile-level parallelism of the RAW microprocessor; with little attention to optimize on-chip transport and buffering of packets, this version characterizes a base-line configuration for exploiting the parallelism of the RAW microprocessor for routing IP packets.

In fact, the performance of this simple version suffers from several shortcomings in the way it transports and buffers incoming packets. First, with two memory banks mapped to four input ports, this version is based on a shared-memory architecture. As a consequence, there is contention for the two packet buffers as well as on the static-network that maps the incoming streams to the off-chip memory. Second, the packet buffer in this version is half-duplex DRAM, limited to performing either a read or a write request, but not both simultaneously. Third, since RAW's dynamic networks are dimension-order routed, with the x dimension first, the data paths in this version suffer from contention on both the Memory-Dynamic-Network (MDN) and the General-Dynamic-Network. The third stage tiles must contend for the GDN when sending a packet header to the fourth-stage

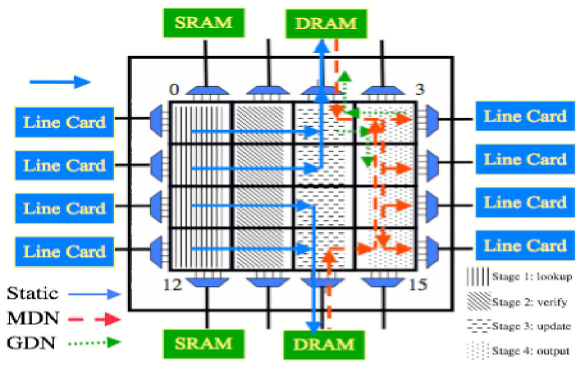

Figure 1: Architecture of RAW Router Version I

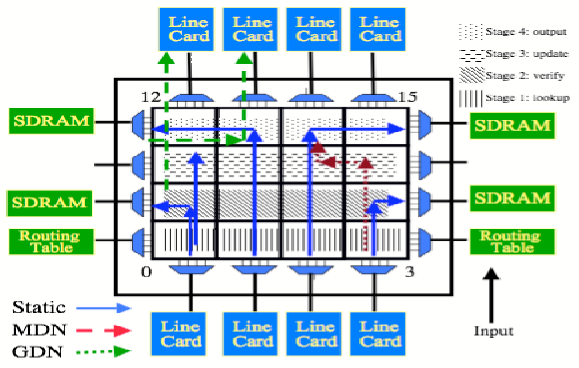

Figure 2: Architecture of RAW Router Version II

tile serving the destination network port. Worse, there is severe contention on the MDN, as output linecards use the memory network to DMA the packet payloads from the packet buffers.

3.2 Buffering and Switching: RAWRouter Version II

The second version (shown in figure 2) of the RAW router is focused on exploiting RAW's software interconnects and pins to optimize the switching and buffering of packets in the routing fabric.

In order to reduce the contention for packet buffers, this version is based on an input-buffering architecture; instead of sharing two packet-buffers between four network ports, this version employs four memory banks, connected to tiles 4, 7, 12, and 15, each statically mapped to an input network port via the static network. Additionally, this version uses full-duplex SDRAMs for buffering packets, permitting simultaneous read and write operations. Finally, both the input and output port linecards and memory interact using the RAW cache miss protocol instead of DMA used in the first 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.

To reduce the contention on the memory and general dynamic networks, this version provisions the parallelism of RAW's tiled-architecture such that it can take advantage of the dimension-ordered-routing of the RAW dynamic networks. Conceptually, the packet processing pipelines in this version are rotated 90 degrees from Version I, so that they are oriented as shown in figure 2. In this orientation, the results from the third stage are sent across the third stage dynamic network instead of causing contention on GDN in the fourth stage. The placement of packet buffers on two different rows, each independently mapped to an input port, not only eliminates the contention on the static network when storing packets, but the corresponding data paths between the packet buffers and the output ports on the memory network are also free of contention (unless two or more line-cards are reading from the same packet-buffer). "Out of band communication" between the first three stages of the pipeline—transmission of destination address from Stage 1 to Stage 3, and the result of the verification step from Stage 2 to Stage 3—is done using the general dynamic network so as not to have to reset the switch processor on the static network.

Figure 5 and 6 compare the performance of Version II with Version I and IXP1200.

Version II performs 10% better than Verion I for 64-byte packets, while our major improvement in the buffering architecture yields close to 30% increase in throughput with 1500-byte packets.

3.3 Optimizing Memory Access: Version III

Version II is designed to optimize the flow of packets inside the routing fabric and incurs the overhead of external memory access only for packets larger than 64 bytes. However, since the transfer of packets from a line-card to the packet buffer is accomplished by cache-missing, larger packets incur the overhead of several memory transactions, each transferring an eight word (32 byte) cache line.

Version III maintains the design of Version II but reduces the overhead of transferring large packets between the line-cards and the external memory. To accomplish this, Version III employs *streaming memory transactions* to transfer large packet payloads from the line-cards to the SDRAM packet buffers. 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 read or write an arbitrary given number of data words starting from a given address. The memory controller is then able to read or write one word of data per cycle using the pre-configured paths on the RAW static network. With the streaming memory transaction, packets need not be fragmented into smaller fragments when written to the packet buffer.

In our implementation of the streaming memory interface, the first tile in the packet-processing pipeline manages the packet memory as a circular buffer. When interrupted by the input linecard, the first tile configures the memory controller with the address of the next available memory location, as well as the length of the packet payload, aligned to a cache-line boundary. 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 every word that it routes, and when the counter reaches zero, the switch processor branches to begin routing the next packet header or payload.

The optimized memory interface depends on the streaming property of the RAW static network. However, for transfer of packets from the memory to the outgoing line-card, memory requests must be routed on a dynamic network. Because the output linecards cannot know which SDRAMs will hold the packet payloads until they receive the header, static routes cannot be pre-configured for streaming memory transfers. The overhead of reconfiguring the static network for every packet payload is too expensive,

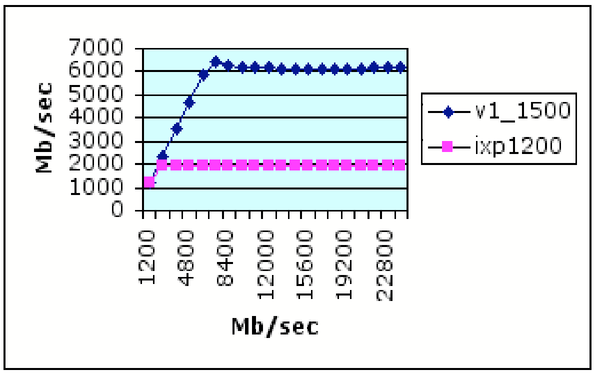

Figure 3: Throughput comparison of RAW Version I and IXP1200 with 1500-byte Packets

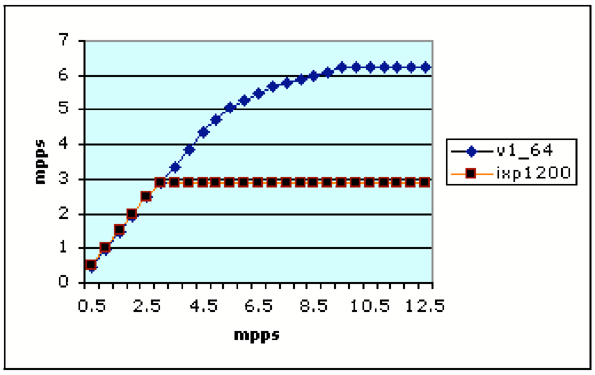

Figure 4: Comparison of RAW Version I and IXP1200 Forwarding Rates with 64-byte packets

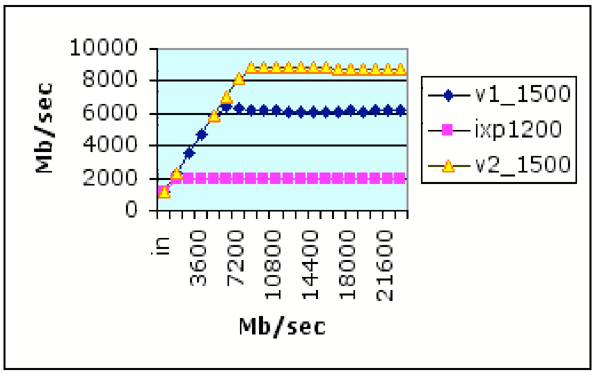

Figure 5: Throughput Comparison of RAW Version I, II and IXP1200 with 1500-byte packets

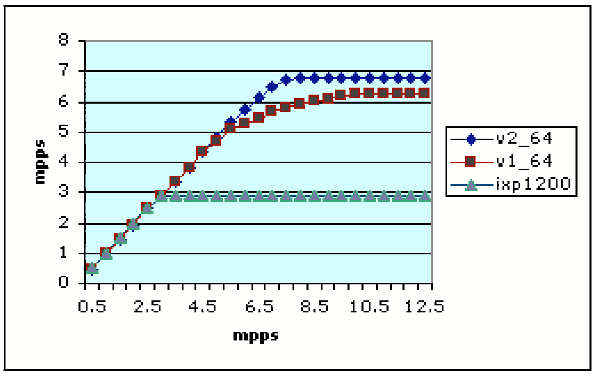

Figure 6: Comparison of RAW Version I, II and IXP1200 Forwarding Rates with 64-byte packets

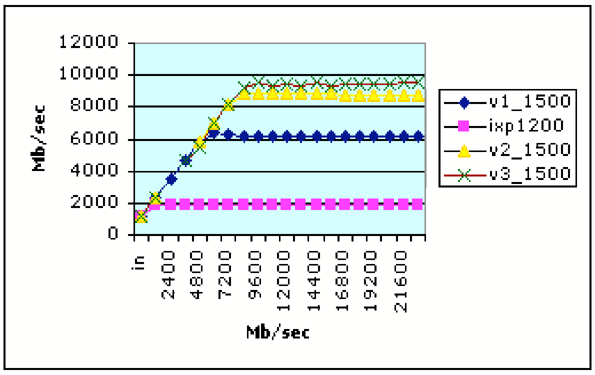

Figure 7: Throughput Comparison of RAW Version I, II, III and IXP1200 with 1500-byte packets

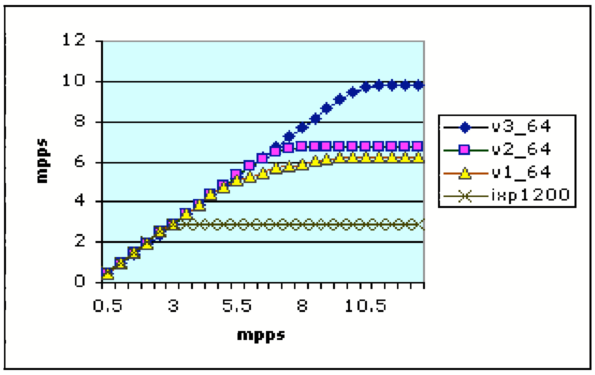

Figure 8: Comparison of RAW Version I,II, III and IXP1200 Forwarding Rates with 64-byte packets

confounding the use of streaming interface. Instead, the output linecards use the standard memory interface, which supports transactions of eight word cache-line-sized blocks. Although the amount of memory per request is fixed at 32 bytes, the memory controllers in Version III supports 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. However, even with the use of pipelined requests, the retrieval of packet payloads to the output linecards is the bottleneck step in this router design for packets with large payloads.

Finally, instead of incurring the cost of accessing an external packet buffer for every incoming packet, packets are buffered on the external memory only when "needed". Specifically, for packets smaller than 64 bytes in size, constituting more than 40% of the Internet traffic, the RAW router simply streams the packets through the pipeline of processing tiles. The choice of 64-byte as a minimum threshold value for buffering a packet is significant in two respects. First, a 64-byte frame is large enough to encompass a maximum size IP header, including optional fields. This ensures that only packet payloads are stored in external buffers, while headers are processed at line-speed. Second, a 64-byte frame permits the limited amount of on-chip queuing at the fourth stage to be used for resolving transient contentions for outgoing links. Packets smaller than 64 bytes in size are padded with additional bits such that the static network routers responsible for reading the packet header into the pipeline stages are statically programmed to route 64 bytes into the tile, avoiding the complexity to read the header length field and count-down the number of bytes to be read into the tile.

Figures 7 and 8 compare the performance of Version III with Version I and II, as well as with IXP1200. The design improvements in Version III yield a further improvement of 10% for 1500-byte packets and close to 30% for 64-byte packets (since only those packets that are larger than 64-bytes incur the overhead of memory access).

3.4 Buffering on Line-cards: Version IV

The three versions presented so far in the paper make minimal assumptions about the processing and buffering capacity of the line-cards using the router fabric. While this permits interoperability with a wide-range of line-cards found in the edge-routing domain, most high-end line-cards are embedded with sufficient memory to buffer traffic at line-speed i.e. 250msec of buffering at the throughput supported by the line-card. With such line-cards, there is no need to route incoming traffic streams to external buffers. In the case of the RAW router, buffering on the line-cards avoids the cost of accessing off-chip memory, as well as the congestion on the RAW on-chip networks caused by transporting the packets to and from the external buffers. The first 64-bytes of each packet are simply streamed through the router packet-processing pipelines, as is done in previous versions, while the rest of the packet is transferred from the input to the output line-card using the RAW general dynamic network.

Figures 9 and 10 compare the performance of this version with earlier versions that buffer packets on the off-chip memory, as well as with IXP1200 as a reference point. By avoiding the cost of external memory access altogether, this version achieves a throughput of 15Gb/sec for 1500-byte packets, an improvement of 36% over Version III. The forwarding rate of the router with 64-byte packets, however, is not affected since these packets are not buffered in Version III either.

4. SCALABILITY: VERSION V

In the router architectures presented above, each input port is served by a dedicated pipeline. This need not be the case when fewer than four line-cards are connected to the router. Our routing software detects the number of line-cards at boot-up time and establishes an appropriate mapping between packet-processing pipelines and the incoming traffic streams. To accomplish this, the static router of the tile connected to the input linecard is programmed to switch incoming packets both towards north, to feed the its own pipeline, and either east or west depending on which other pipeline is not connected to a linecard. Correspondingly, the static router of a first-stage tile not connected to a line-card is programmed to read packets from another first-row tile rather than from a line-card. This scheme permits the processing capacity of the router to be mapped to a variable number of line-cards, much like in network processors such as the IXP1200 and Cisco Toaster, albeit without bottlenecking the design with the use of a centralized register or a multiplexing unit. Figure 12 shows the forwarding rate of the RAW router (Version III) when connected to only two line-cards as opposed to four line-cards. With each incoming traffic stream processed by two pipelines, the two-port version can process minimum-sized packets at a rate 20% faster than the four-linecard setup. The maximum throughput achieved with two linecards, 12Gb/sec, represents the processing capacity of the RAW fabric when packets are not stored on external memory. For large packets the cost of storing and retrieving packet payloads from the memory dominates the performance (shown in figure 11).

However, scaling the router beyond 4 line-cards presents an interesting challenge on the RAW microprocessor. In all previous versions, the performance of the router is enhanced by the availability of the RAW static networks to stream incoming packets to the external packet buffers, as well as a broadcasting channel to feed the header-processing pipeline. However, routing on the RAW static network requires that 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` $cSi \rightarrow csti, cWi \rightarrow cNo$, requires input data to be available on both cSi and cWi before executing the instruction. This design has the negative side effect that, given the non-deterministic arrival rate of Internet traffic, static networks cannot be made to “cross” when streaming packets from the linecards to the DRAMs. Therefore, in all previous versions, the routing functions, packet buffers and line-card are laid out on the RAW matrix such that the static networks do not ’cross’. However, such a layout cannot be extended beyond a 4x4 tile-matrix on the RAW microprocessor.

Therefore, scalability of our router architecture beyond a 16-tile matrix must depend solely on the use of the RAW dynamic networks for routing data on the chip. This adversely effects the performance of the router in two respects: 1) Packet Duplication. Incoming packets cannot be broadcast to the RAW tiles – incoming packets must be replicated and independently routed to each stage in the header-processing pipeline. 2) Packet Fragmentation: Dynamic networks must be used to transport packet payloads from the linecards to the external DRAM packet buffers.

In order to understand the performance implications of a dynamic-networks-only router, we re-implemented Version III (dubbed Version III.d) without using static networks (Version IV assumes buffering on the linecards and hence does not rely on the static network to stream packets from the linecards to the packet buffers). Figures 13 and 14 show the performance of Version III.d. The throughput measured with minimum-sized packets, shown in figure 14, illustrates the performance degradation caused by the absence of the static network as a broadcast channel to feed the three stages of the header-processing pipeline. The 30% throughput degradation shown in figure 14 illustrates the overhead of replicating and routing the incoming packets independently to each of the tiles in the header-processing pipeline on the dynamic network.

Figure 13 illustrates the performance degradation in Version III.d for maximum-sized Ethernet frames. The 10% throughput degradation characterizes the performance penalty of routing packet payloads, from the line-cards to the DRAM buffers, as 8-word chunks on the dynamic network, instead of streaming the payload on a precomputed path on the static network. However, since the cost of packet duplication and routing is amortized over large packets, the performance penalty for maximum-sized packets is lesser than that for minimum-sized packets.

5. REAL AND IDEAL PERFORMANCE

Our evaluation so far has focused on either 64-byte minimum-sized TCP packets or 1500-byte maximum-sized Ethernet frames. By studying these two extremes, we discovered several bottlenecks and explored a range of design refinements. However, an important measure of the performance of the router is how well

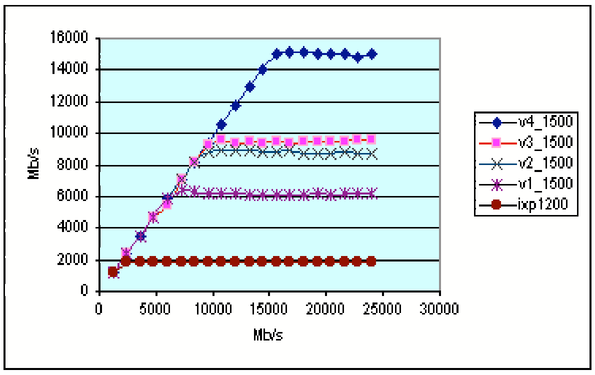

Figure 9: Throughput Comparison of RAW Version I, II, III, IV and IXP1200 with 1500-byte packets

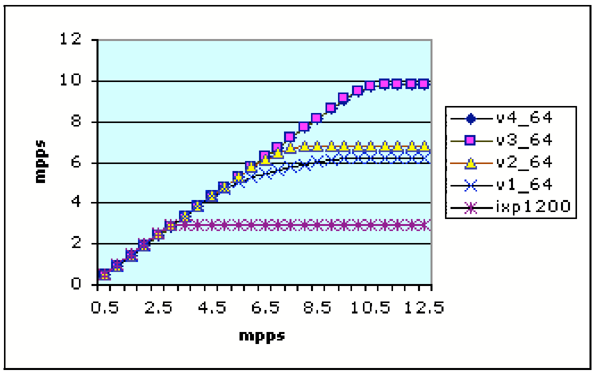

Figure 10: Comparison of RAW Version I, II, III, IV and IXP1200 Forwarding Rates with 64-byte packets

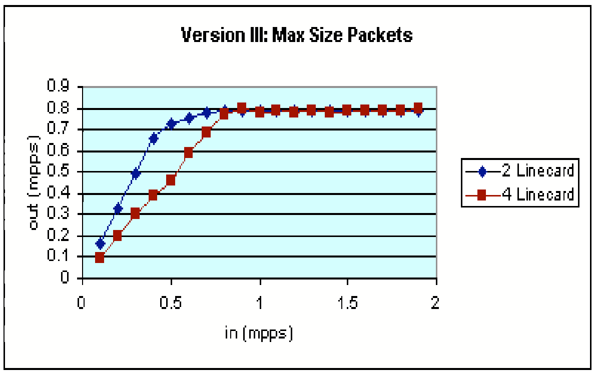

Figure 11: Throughput Comparison of RAW Version III with 2 and 4 Linecards using 1500-bytes packets

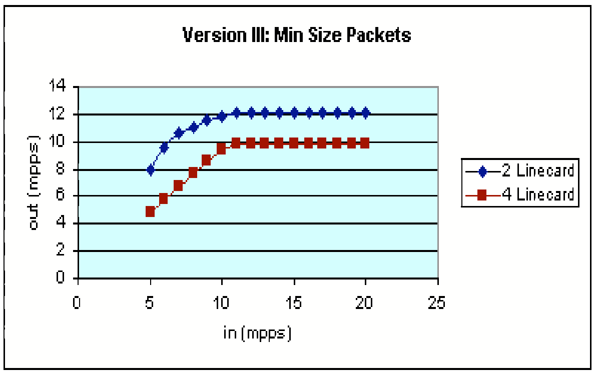

Figure 12: RAW Version III Forwarding Rate with 2 and 4 Linecards using 64-bytes packets

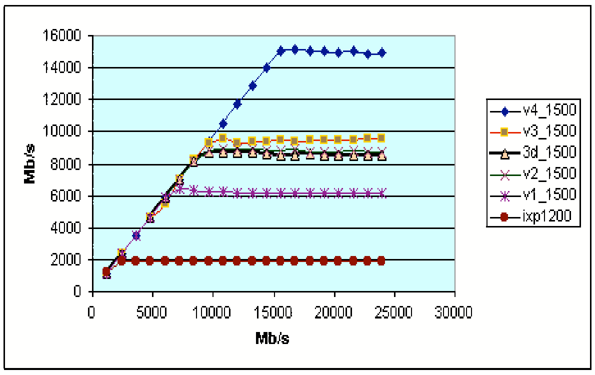

Figure 13: Throughput Comparison of RAW Version III and III.d with 1500-bytes packets

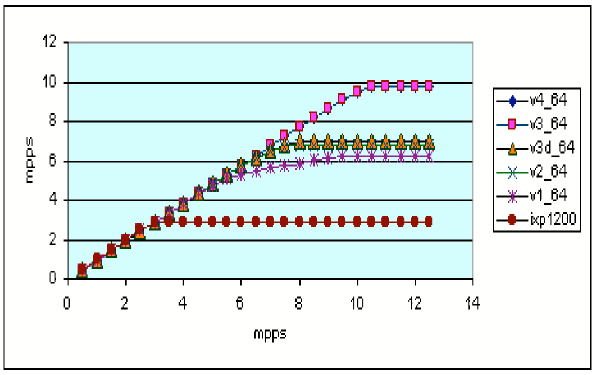

Figure 14: Comparison of RAW Version III and III.d Forwarding Rates with 64-bytes packets

it performs with variable-sized packets in a real Internet traffic trace. Figure 15 shows the throughput of the four basic versions of our router with variable-sized packets in a real Internet traffic trace (http://lever.ucla.edu/ddos/traces) containing 10,000 packets. The average packet size of the trace is around 128 bytes — equivalent to the average packet size on the Internet. The peak achieved throughput of our router is 8.9 Gb/sec with external buffering and 14 Gb/sec when buffering is done on linecards. With variable-sized packets, the peak throughput of each version is slightly lower than the throughput with 64-byte packets due to the overhead of memory transactions.

Furthermore, so far we have only compared the performance of our router with IXP1200. With its popularity, IXP1200 provides a good reference point to put the performance of our architecture in context. However, in order to understand "how good" is our design, we compare the performance of our router architecture with an ideal "null router", that simply forwards an incoming packet to the nearest outgoing port (without any processing). The throughput of the "null router" is plotted in figure 15. The "null router" achieves a peak throughput of 15.5 Gb/sec, approx 37% better than our optimized design with external buffering and only about 1% better than the architecture which assumes buffering on the

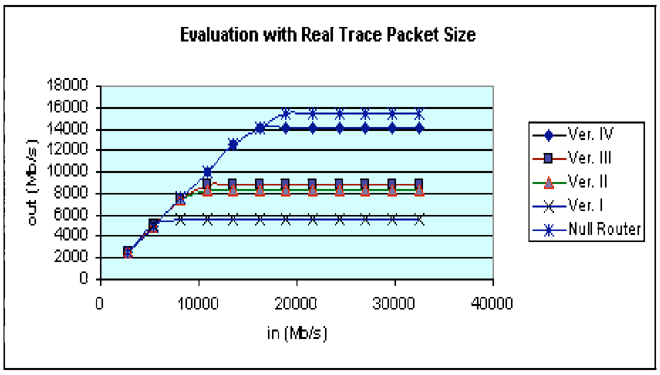

Figure 15: Throughput Comparison of RAW router Version I, II, III and IV with a "Null Router" using variable-sized packets in a captured Internet Traffic Trace

linecards. The performance difference between the "null router" and our architectures reveals the inefficiency our design resulting from the overhead of accessing external buffers and contention on the dynamic network when retrieving the packet payloads.

5.1 Latency Evaluation

Table 1 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 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 [7].

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 be 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. 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.

Table 2 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

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 1: RAW Latency. Latency of the RAW router for 64 and 1500 byte packets.

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 — typical for Internet traffic often comprised of packet-trains. 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.

6. RELATED WORK

Previous work on software-based high-performance network routing has largely focused on custom-design network processors and traditional general-purpose architectures.

Spalink et al [9] describe an extensible software router for the Intel IXP1200 Network Processor. However, their router is based on a shared memory architecture, in which micro-engines are statically partitioned as either input or output processing engines and communicate using serialized reads and writes to shared DRAM. The router architecture presented in [9] also does not exploit the parallelism available within different header processing tasks; an incoming packet is processed by a single micro-engine rather than a pipeline of processors. Spalink et al [9] report a peak IPv4 forwarding rate of 3.47 Mpps, compared to 9.75 Mpps achieved by our software routers.

In [7] Chen et al evaluate the performance of Click — a modular, software router — on Pentium Symmetric Multi-Processor (SMP) using a 4-way PIII SMP clocked at 500 MHz and 1GB RAM. Chen et al [1] report a peak MLFFR of only 0.5 Mpps. Interestingly, Chen et al report that their SMP setup did not yield any noticeable performance improvement beyond 4 processors, primarily due the overhead of synchronized memory access by the parallel processors.

Even the general-purpose single-chip parallel architectures, which offer parallelism akin to custom-designed network processors, fail to rival the performance of custom-designed network processors. Crowley et al [3] compare the performance of three different general-purpose single-chip parallel architectures — 8 processor single-issue CMP, 4 processor 2-way CMP, and 8-way SMT — all clocked at 500 MHz, and report a peak throughput of 1.28 Gb/sec with the 8-way SMT and only 0.92 Gb/sec with the 4 processor 2-way CMP. Crowley et al also found that increasing the number of contexts yielded diminishing returns at best due to the overhead of synchronized access to shared memory and context-switching.

The work presented in this paper follows from an initial investigation of RAW's tiled-architecture as a high-bandwidth switching fabric [2]. In this paper we investigate the suitability of RAW as a general-purpose network router architecture, including switching, buffering and processing of IP packets. We also study the scalability and role of RAW on-chip networks for achieving high-throughput for a variety of network loads.

7. CONCLUSION AND FUTURE WORK

The evaluation of the router architectures presented in this paper reveal the following key insights:

- The replicated core architecture coupled with low-latency on-chip interconnects provide a much more attractive routing platform than traditional memory-oriented architectures like x86.

- The software-level control over chip resources plays an important role for high-performance routing; the performance of our routers was primarily derived by how much control the software had on the layout of routing functions and the role and behavior of the on-chip networks. For instance,

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 2: 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.

the performance of our network router improved by 30% by changing the layout of the routing functions on the chip to reduce the contention on the on-chip networks. Similarly, the performance of the router improved by over 15% by employing the RAW static network and general dynamic network (GDN) for memory transactions, rather than using the default RAW memory dynamic network (MDN).

- A fixed routing policy, like Dimension-Ordered-Routing (DOR) in RAW, may lead to on-chip contention, with a performance penalty of 30-35%, if the routing functions are not carefully distributed to avoid overlapping data paths.

- While dynamically-routed on-chip networks are imperative for switching incoming packets to appropriate output ports, the presence of streaming, static network may lead to a performance gain of 10-30% when used for tile-broadcast and memory access.

- For large packets, streaming memory transaction, such as streaming DDR, improve the performance of the router by approximately 10% over using traditional Direct-Memory-Access (DMA) over dynamically routed networks.

- Buffering on line-cards avoids external memory transactions, leading to a performance gain of more than 35% over architectures that require external buffering.

- Given the non-deterministic nature of packet arrival times, the scalability of our router is adversely affected by the operand-centric design of the RAW static network which requires that all operands are met before data may be switched between tiles.

The work presented in this paper has focused on evaluating the packet processing and forwarding capacity of the RAW microprocessor. Hence, we have not discussed issues relating to the control-plane of the router. Our future work will focus on scheduling policies, quality of service issues, queue management strategies and evaluation of high-level network services like NAT, firewalls and layer-7 switching on the RAW microprocessor. We are currently exploring decentralized scheduling architectures such as a buffered-crossbar [8] and two-staged switch [6] to orchestrate the flow of packets within the routing fabric.

8. REFERENCES

[1] Benjie Chen and Robert Morris. Flexible control of parallelism in a multiprocessor pc router. In *Proceedings of the 2001 USENIX Annual Technical Conference (USENIX '01)*, pages 333–346, Boston, Massachusetts, June 2001.

[2] Gleb A. Chuvpilo and Saman Amarasinghe. High-Bandwidth Packet Switching on the Raw General-purpose Architecture. In *2003 ICPP*, 2003.

[3] Patrick Crowley, Marc E. Fluczynski, and Jean-Loup Baer. On the Performance of Multithreaded Architectures for Network Interfaces. Technical Memo, UW, 2000.

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

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

[6] Isaac Keslassy and Nick McKeown. Maintaining packet order in two-stage switches. In *IEEE INFOCOM*, June 2002.

[7] 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.

[8] Nick McKeown Shang-Tse Chuang, Sundar Iyer. Practical algorithms for performance guarantees in buffered crossbars. In *IEEE INFOCOM*, March 2005.

[9] Tammo Spalink, Scott Karlin, Larry L. Peterson, and Yitzchak Gottlieb. Building a robust software-based router using network processors. In *Symposium on Operating Systems Principles*, pages 216–229, 2001.

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