Shared Memory Multiprocessor Architectures for Software IP Routers

Yan Luo, Laxmi Narayan Bhuyan, *Fellow, IEEE,* and Xi Chen

The authors are with Computer Science and Engineering, University of California, Riverside, CA 92521. {yluo,bhuyan,xichen}@cs.ucr.edu

This research has been supported in part by NSF grant CCR-0105676 and CCR-0196102.

Abstract

In this paper, we propose new shared memory multiprocessor architectures and evaluate their performance for future Internet Protocol (IP) routers based on Symmetric Multi-Processor (SMP) and Cache Coherent Non-Uniform Memory Access (CC-NUMA) paradigms. We also propose a benchmark application suite, *RouterBench*, which consists of four categories of applications representing key functions on the time-critical path of packet processing in routers. An execution driven simulation environment is created to evaluate SMP and CC-NUMA router architectures using this *RouterBench*. The execution driven simulation can produce accurate cycle-level execution time prediction and reveal the impact of various architectural parameters on the performance of routers. We ported the FUNET trace and its routing table for use in our experiments. We find that the CC-NUMA architecture provides an excellent scalability for design of high performance IP routers. Results also show that the CC-NUMA architecture can sustain good lookup performance even at a high frequency of route updates.

Index Terms

Multiprocessor Architecture, Shared Memory, IP Router, Network Applications, Route Updates.

I. Introduction

As link speed continues to grow exponentially, packet processing at network switches and routers is becoming a bottleneck. Assuming an average packet size of 1000 bits, a multi-gigabit router has to process several million packets per second. This implies that the average packet processing time has to be less than one microsecond, which is impossible for a single processor to achieve.

Different approaches are proposed and deployed to improve the performance of Internet Protocol (IP) routers, whose performance metric has been mainly the throughput or packets per second. Application Specific Integrated Circuits (ASICs) are usually preferred in high-end core routers due to their high performance while tolerating their inflexibility and lack of programmability. The emerging network processors provide programmability and flexibility together with high performance, but are usually targeted to edge routers. Therefore, general purpose multiprocessors are preferred for high-end core routers because of their high performance, programmability and scalability [7], [27]. Packet processing in these core routers is accomplished through software, hence they are called software IP routers.

Modern commercial routers such as BBN Multi-Gigabit Router [27] and Cisco 12000 Series Gigabit Switch Router [7] employ multiprocessors to process packets. BBN's MGR [27] incorporates a number of centralized Forwarding Engines (FEs) that are reponsible for processing packets coming from the line cards (LCs). LCs are network interfaces where packets arrive and depart. Cisco 12000 series multi-gigabit routers [7] adopt a distributed architecture, in which each LC has a dedicated FE associated with it. As these commercial routers are deployed, some questions arise for both industrial and academic community. Among them, the following two questions are addressed in this paper:

- How to evaluate the software packet processing capability of a multiprocessor router architecture?
- What types of multiprocessor architectures yield the best performance for core routers?

The aim of this paper is to evaluate different multiprocessor architectures for software routers, and to develop techniques to improve performance. Execution time or packet processing latency in a router is considered as the main performance metric in this paper instead of throughput. Thus we employ execution driven simulation to predict the performance of IP routers. The advantage of execution driven simulation is that it can produce accurate cycle level execution time and reveal the impact of architectural parameters on router performance. We extend Augmint [22] to perform multiprocessor architecture simulation. We choose Augmint because it runs fast due to its simple processor architecture model, and it enables us to focus on exploring parallelism at the packet level. The impact of packet-level parallelism on the router performance is known to be more profound compared to architectural extensions like instruction-level parallelism (ILP), branch prediction or speculation [4].

In order to evaluate the router architectures, we propose a benchmark called *RouterBench*, which consists of four categories of applications that reside on the time-critical path of packet processing in a core router. Our benchmark applications are similar to those of the Click router [16]. We quantitatively analyze the computation requirements of the key functions in *RouterBench*, and evaluate their execution time on the shared memory multiprocessor router architectures.

Route updates may happen frequently due to IGP, BGP [28] or other routing protocols. The routing update messages cause changes to the routing table structures, stall the route lookup procedure, and degrade the lookup performance. Hence, we also study the impact of such route updates on the lookup time of multiprocessor routers.

In both BBN's MGR [27] and Cisco's 12000 series routers [7], the common data structure, such as routing table, is replicated in all the processors resulting in a wastage of memory. Also, updating multiple copies of a large data structure is time consuming and needs special hardware support. Shared memory architectures enable multiple processors to share one copy of data structure, e.g. routing table, which not only saves memory space, but also eases updating. The scalability and high performance of shared memory multiprocessor architectures make them salient candidates for router architectures. Hence, we propose to design and test two shared memory architectures in this paper.

The contribution of this paper is the following:

- We propose two shared memory multiprocessor architectures for IP routers based on SMP and CC-NUMA organizations.
- We construct an execution driven multiprocessor simulation framework for evaluating these router architectures consisting of forwarding engines (FEs), line cards (LCs) and a switching fabric.
- We develop a set of benchmark applications that reside on the critical path of packet processing in the routers. This *RouterBench* is used to evaluate the multiprocessor architectures under study.
- We study the impact of route updates on the performance of routing table lookup in a multiprocessor router architecture.

This paper is organized as follows. The related research is presented in Section II. We describe the SMP and CC-NUMA router architectures, and introduce a simulation framework in Section III. In section IV, we identify the key functions that a router performs for processing a packet, and propose *RouterBench*, which is used in the simulation study of multiprocessor router architectures. The simulation results are presented in Section V. In Section VI, we study the impact of route updates on the routing table lookup performance of the CC-NUMA based router. Finally, Section VII concludes the paper.

II. Related Work

Wolf and Turner proposed the design of a scalable, high-performance active router [34], where multiple network processors with cache and memory on a single application specific integrated circuit (ASIC) were used. However, a general purpose multiprocessor architecture has advantages over ASIC or specialized processor because it avoids long development process, incurs less design cost, and is programmable. In addition, general purpose processors usually have on-chip caches, which can store frequently-used data structures locally. Hence, BBN MGR [27] is based on DEC Alpha 21164 general purpose processors.

The performance evaluation of a router should be carried out with appropriate benchmark applications. There exist a few network benchmarks for network processor design and evaluation like CommBench [32], NetBench [18] and PNI benchmark [9], etc. CommBench includes a set of header processing applications in traditional networks and payload processing applications in active networks. NetBench contains micro-level, packet-level and application-level benchmarks. PNI benchmark covers packet header applications such as packet classification and fowarding, and packet data processing applications such as IPSec. Although these benchmarks cover various applications in the network realm, none of them specifically addresses all the workload in an IP router. RFC 1812 is the classic document that states the requirements of IP Version 4 routers [2]. Karlin et al. [15] summarized the main functions of routers as classification, fowarding and scheduling. Kohler et al. [16] built a functional software router - Click Modular Router where the key functions of an IP router are implemented.

Most of the work in performance evaluation of routers emphasizes the delay in packet transmission, and ignores the packet processing latency. Chan et al. presented a modeling framework to optimize the cost and performance of routers with multiple forwarding engines [5]. They assumed that route lookup time was exponentially distributed, which did not reveal the real packet processing delay. Papagiannaki et al. [26] measured the single-hop delay from an operational backbone network router, but did not specify the processor computation time, queuing delay, and other contributions to the total delay. Chen et al. [6] studied the buffer and queue management techniques to optimize the Click software router [16] on off-the-shelf multiprocesor PC hardware. In both [6] and [26], the implementations are based on existing hardware, which lack study on investigating the impact of various architecture parameters (such as CPU power, memory size), and on tuning these parameters to optimize the router performance.

Bhuyan et al. [4] were the first to measure the packet processing time and architectural impact through an execution driven simulation. They extended the RSIM [25] simulator to evaluate the execution of a routing table lookup algorithm on a multiprocessor router. In this paper, we further extend the previous research [4] to a complete simulation framework and router workload. We extend Augmint [22] to perform multiprocessor architecture simulation because RSIM spends too much simulation time on architectural detail inside the CPU, such as instruction level parallelism (ILP), branch prediction, speculative execution, etc. It was shown that the impact of these parameters is not that significant compared to multiprocessing [4]. Augmint assumes simple processor architectures, but enables us to focus on exploring packet-level parallelism.

III. Multiprocessor Architecture for Routers

A. Existing Multiprocessor Router Architectures

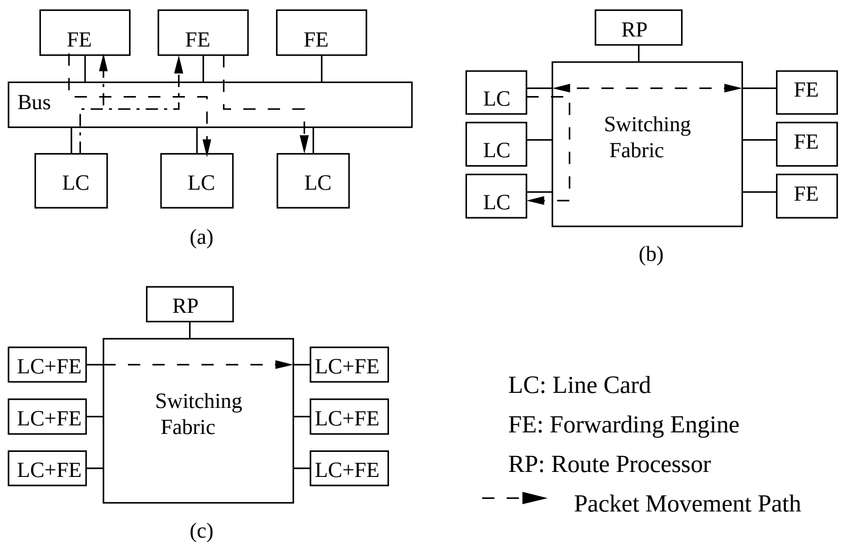

Fig. 1. Router Architectures. (a) multiple FEs and LCs with shared bus, (b) paralle router architecture with separate LCs and FEs, (c) distributed router architecture with combined LC and FE.

A basic router architecture consists of line cards, a router processor and a backplane switch. Current trend in the router designs is to use multiple packet processing units instead of a single centralized one. High-performance routers can be divided into three organizations shown in Fig. 1. The architecture of Fig. 1(a) connects multiple FEs and LCs via a high-speed bus. Packet headers can be sent to one of the FEs from an LC to perform route lookup, packet classification and header updates, etc. Such a router is inexpensive in cost, however, since the LCs forward packets to outbound LCs via bus also, the bus could be a bottleneck. The multiprocessor Click router in [6] is based on such an architecture. A parallel architecture in Fig. 1(b) replaces the shared bus with a switching fabric connecting all FEs and LCs. Moreover, a separate Route Processor is incorporated to run Internet routing protocols (such as BGP), accounting and other administrative tasks, which are not time-critical to packet processing. The FEs are responsible for routing table lookup, packet classification and packet header updates, which are time-critical operations. BBN's Multi-Gigabit Router[27] architecture conforms to Fig. 1(b). Fig. 1(c) depicts a distributed architecture, where each LC has an FE locally, and all router tasks such as routing table lookup are completed in the local FE. An example of such an architecture is Cisco 12000 series multi-gigabit router, in which each LC has a dedicated layer three forwarding engine.

There are two additional decisions to be made when designing a high performance multiprocessor router. The first one is the programmability. A programmable architecture has increasing advantages over ASICs because new protocols and router functions can be incorporated. So we consider general-purpose processors as forwarding engines. The second one is the memory usage. The routers can either have all data structures replicated for each processing element, or have a shared data structure. The obvious benefit of the replicated one is the simultaneous memory access by all the FEs. However, as size of the data structures (such as tree structure for routing table lookup) grows and number of processing elements increases, the memory requirement may easily exceed beyond limit. For example, Cisco 12000 one-port 10-Gigabit Ethernet Line Card has 256MB routing table memory [8], which turns out to be 4GB for a router with only 16 LCs. Another drawback of replicated data structures is the difficulty to update them. For example, whenever there is an update on the routing table, every copy of the data structure for routing table lookup will need to be updated simultaneously. This will increase the hardware complexity or degrade the performance significantly if done sequentially. So a shared memory architecture seems to be a good candidate. A distributed shared memory (DSM) organization can be adopted to provide simultaneous memory access from different processors. This is possible because different processors will access different parts of the shared data structure at any particular time.

Shared memory architectures can be divided into two categories, namely, Symmetric Multi-Processors (SMP) and Cache-Coherent Non-Uniform Memory Access (CC-NUMA) multiprocessors. Their organizations are briefly described later in this section. In shared memory architectures, a global memory space is shared by all the processors in the system. Each processor has a local cache, which can store recently accessed shared data blocks. In order to study the sharing

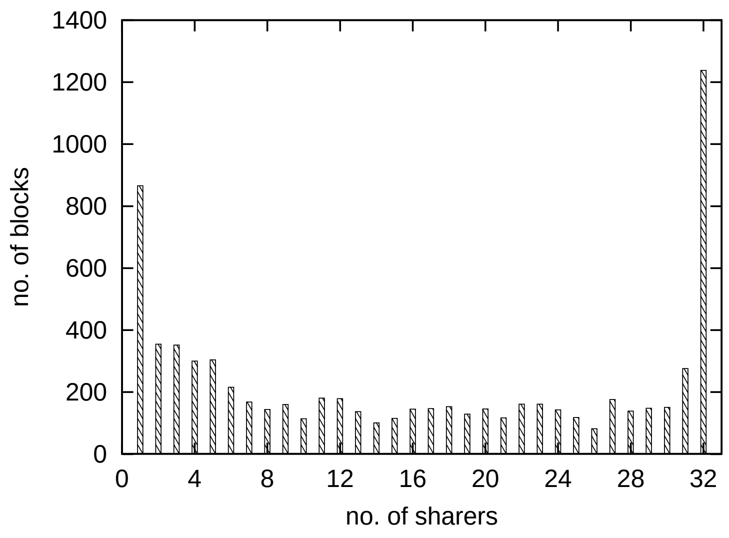

Fig. 2. Sharing on SMP.

behavior in routers, we executed Radix Tree Routing(RTR) [31] table lookup algorithm on a simulated SMP architecture with 32 processors. RTR is a classic route table lookup algorithm for layer three (IP) packet forwarding in routers, which performs route lookup based on a radix tree data structure. We used publicly available FUNET trace and its routing table [24] in this experiment.

Fig. 2 illustrates the sharing behavior of RTR, where x-axis denotes the number of processors sharing a certain data block, and y-axis is the number of data blocks that *x* processors share. It can be observed that, the number of data blocks shared by all processors is very high, which means that the degree of sharing in RTR lookup is substantial. These highly shared blocks are actually radix tree root node and its near descendants. They are shared most frequently because every lookup has to begin from these nodes located at the top of the radix tree. As RTR lookup traverses the tree, route lookup on packets with same destination IP address will follow the same path until the lookup finishes whereas other lookups may branch to different ways after some point. A continuous stream of packets with same destination IP will be seen in the packet stream because two communicating parties will maintain a connection and send data packets for a period of time. However, these packets may be sent to different FEs for processing. The corresponding FEs will follow the same path on the radix tree, and thus the tree nodes along the path will

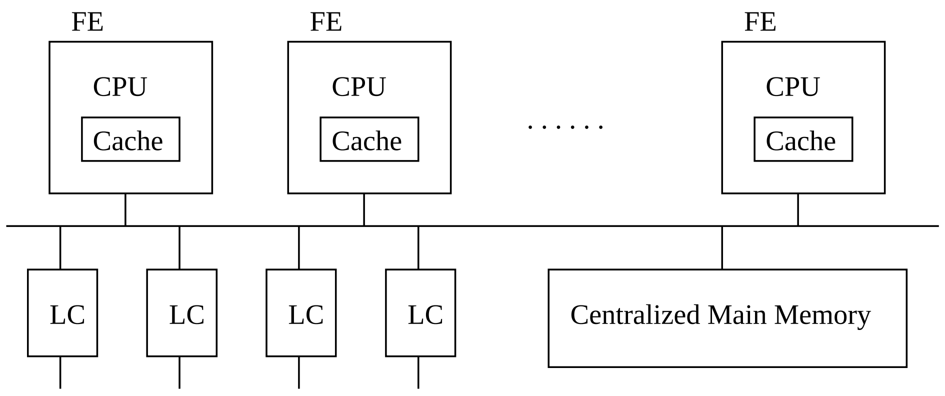

Fig. 3. SMP Router Architecture

be shared by these FEs. This interesting observation motivates us to propose multiprocessor architectures with shared memory for IP routers. We examine the suitability of both SMP and CC-NUMA multiprocessor architectures in the following sections.

B. SMP Based IP Router

In SMP router architectures, the FEs and LCs are connected through a high speed bus. The centralized shared memory module is also attached to the bus as depicted in Fig. 3. This architecture is same as the multiprocessor Click router [6].

An SMP-based router architecture has the following features:

- Each FE has cache memory of one or multiple levels. The SMP router architecture uses uniform/centralized shared memory, and broadcast/bus-based snoopy cache coherence protocol [3], [13].
- Centralized shared memory stores routing table and other data structures, which are shared by all the FEs.
- An LC has an on-card memory buffer, where incoming packets (both header and payload) are stored.
- An FE accesses these memory spaces using memory mapped I/O operation, processes the packet header, and writes the port number of the outbound LC into the packet header.
- LCs transfer the packets to outbound LCs via the same shared bus.

The FEs perform header checking, classification, routing table lookup etc. Routing table lookup

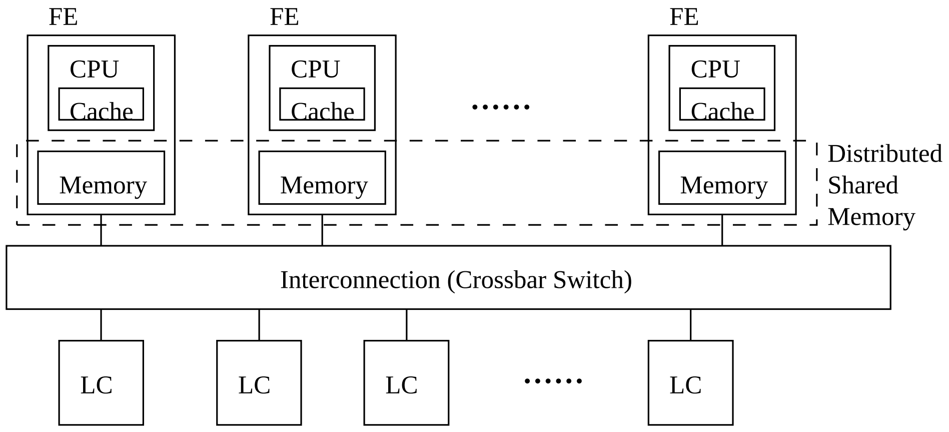

Fig. 4. CC-NUMA Router Architecture.

process involves loading radix tree nodes from centralized main memory to cache, upon which RTR algorithm is performed. The lookup result and updated packet header are written back (again via bus) to the origin LC. Intuitively the potential problem could be the bottleneck of the centralized memory and bandwidth limit of the bus. The FEs have to compete for the bus to access shared data structures in centralized main memory, and the LCs compete for the bus to transfer packets consisting of both headers and payloads.

C. CC-NUMA Based IP Router

The architecture shown in Fig. 1(b) avoids the bus problem by putting a crossbar network between the LCs and FEs. A crossbar switch allows all one-to-one connections between the LCs and FEs. Although BBN's MGR has an organization similar to Fig. 1(b), the memory of FEs is not shared, so the routing table is replicated in all the FEs. We propose a CC-NUMA architecture to eliminate the replication by using a shared memory paradigm.

A Cache Coherent Non-Uniform Memory Access (CC-NUMA) router architecture, shown in Fig. 4, has the following features.

- Each FE has cache memory that can be of multiple levels. The cache coherence is maintained through a directory-based organization [3], [13]. It also has a local memory module, which is part of the global shared memory space.
- Each LC has an on-card memory buffer, which stores the packet (both header and payload).

An FE can access any memory remotely via crossbar. Ideally, an FE should be located at the LC, as considered in [4]. However, we chose the organization in Fig. 4 similar to the BBN's MGR.

- All FEs and LCs are connected via a crossbar switch fabric, which allows simultaneous multiple connections for high bandwidth.

A CC-NUMA router works as follows. Each FE gets a new packet header from an LC, performs header checking, classification, and routing table lookup etc. Routing table lookup involves loading radix tree nodes from distributed memory to local cache. The packet header is updated using the lookup result, and it is written back to the origined LC. The LCs then initiate packet transfers via the crossbar.

D. Multiprocessor Router Architecture Simulation

To evaluate the performance of the above router architectures, we develop an execution-driven simulator. Augmint [22] is initially a simulation tool for Intel CISC processors, only simulating the instruction execution on the processor side. Its processor simulation is relatively simple, however, it enables us to focus on the multiprocessing of packets. Augmint provides us the flexibility to add memory module and bus/crossbar module for SMP/CC-NUMA architecture in the backend. We construct a cache memory module with 32-byte cache blocks and 32KB cache size. It is two-way set-associative, and the cache replacement policy is LRU. We extended Augmint to implement a snoopy cache coherence protocol for SMP and a full-map directory-based protocol for CC-NUMA architecture [3], [13]. Memory management policy in CC-NUMA is page-based round robin, so the radix tree data structure is distributed uniformly across the memory modules. We simulate FEs and LCs as independent components as in a real router, and they interact through the interconnection (bus or crossbar switch).

IV. RouterBench

To evaluate router architecture performance with appropriate benchmark applications, we need to identify the key functions of routers. RFC 1812 [2], "Requirements for IP Version 4 Routers", states that "an IP router can be distinguished from other sorts of packet switching devices in that a router examines the IP protocol header as part of the switching process."

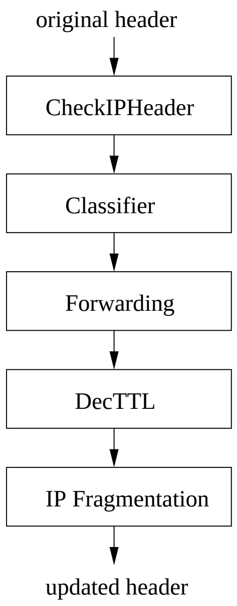

Fig. 5. Packet processing fbw.

The processing of IP layer involves IP header validation, routing table lookup, decrementing time-to-live (TTL), fragmentation etc [2], [16], [15]. Profiling on an example Click sofware router [16] also shows that these tasks take considerable amount of time to execute: header validation, classification and decrementing TTL take 12.3%, 9.2% and 3% of the total execution time, respectively. In addition, packet classification is becoming a mandatory task to support QoS in routers [14]. In general, each packet header will go through the processing shown in Fig. 5. To capture these key router functions, we propose *RouterBench*, which consists of the following four categories of applications.

- Classification
- Forwarding
- Queuing
- Miscellaneous

These operations reside on the critical path of the passing packets, which are time-critical, and are key factors of the router overall performance. Therefore we believe that the *RouterBench* specifically identifies the workload of IP routers and is suitable for evaluating their performance. At this stage, we do not consider other non-time-critical functionalities, such as BGP routing, logging, and administrative tasks.

The source codes of the *RouterBench* applications are available for public download at [17].

A. Classification

Packet classification is the primary task for identifying flows, filtering data traffic, QoS [14] , etc. Packets can themselves carry an explicit service classification field, such as Type of Service (TOS) in IP header [1], [23]. A general classification is done with an access list that allows specification of a packet's source and destination address, protocol, and the protocol port address for TCP or UDP. In some systems such as Click [16], a more generic configuration mechanism allows any field of the packet header to be identified using specification of an offset and a field length.

We take the general classification mechanism as our benchmark application, which uses a rule list and can check any field of the packet header. In theory there can be unlimited number of rules in a rule list. Since there is no typical rule list described in the literature, we list a set of rules that a typical router would configure based on an access list from a campus level router. There are totally 336 rules declared for the inbound traffic. An example of a rule is "dst host 138.23.168.40 and tcp port 80", which classifies all the incoming HTTP packets destinated to a server with IP address of 138.23.168.40. Figure 6 shows the cumulative distribution function (CDF) of packet counts on top rules based on the data in our campus level router. The x-axis is the rank of a rule, and y-axis is the cumulative distribution of traffic that are matched. we find that only 31 rules out of 336 of them classified 99% of the traffic, and many of the rules were never encountered by any packet. We thus use the top 31 rules in our classifier application. Our classification algorithm, ported from Click router [16], is based on the "hierachical trie" algorithm described in [11].

Hierachical trie structure is an extension of the trie data structures. A trie structure is constructed by examining one field (for example, "source IP" field) of the classification rules such that the trie consists of all the different values in this field. Each trie node contains an {*offset, prefix, mask* } tuple to represent the value in the field. *Offset* specifies the bits to be examined. Those bits are ANDed with *mask* and then compared with *prefix* to deterimine if there is a match or not. The left and right children of a node can be either another trie node or a leaf node, which indicates that the final classification decision can be made here. Then, the hierachical

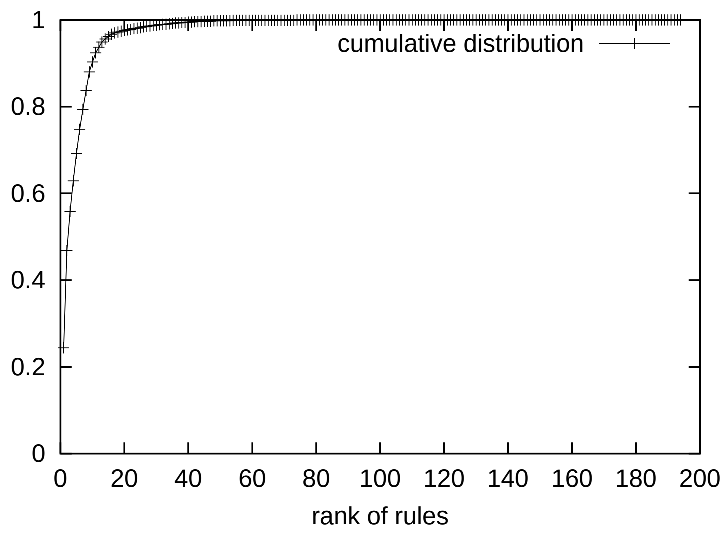

Fig. 6. Cumulative distribution of traffic matched on top rules.

trie is constructed by connecting all the tries based on the rule list. Classification of an incoming packet proceeds from the root node, determines if the packet matches at each node until it reaches a leaf node (decision node).

B. Forwarding

Forwarding packets to their destination hosts is the primary task of a router. To forward a packet, a router has to lookup a route table to determine its outgoing port. A route table lookup algorithm is used to search for a most specific route prefix. Because of Classless InterDomain Routing (CIDR) forwarding becomes a non-trivial problem, and quite a few routing algorithms have been proposed in the literature [29]. Among them the Radix Tree Routing (RTR) route lookup algorithm is from the public domain BSD Unix [21] distribution that performs lookup based on a radix tree data structure. It is widely used, and included in both CommBench [32] and NetBench [18]. We thus incorporate it as our benchmark for forwarding.

In the RTR algorithm, a route lookup is performed based on the radix tree and starts from the root node. Each tree node (radix node) indicates which bit of the 32-bit destination IP address should be examined. The value of that bit (1 or 0) determines which of the children nodes will be the next one to visit. When a leaf node is reached, the destination IP is ANDed with the route prefix mask to verify the final match. Backtracking is possible if there is not a match in the leaf node.

C. Queuing

Queuing may happen at both the input side and output side of a router. Different queuing policies are proposed to control congestion, guarantee fair sharing of bandwidth, and provide QoS [30], [10], [14]. There are two categories of queuing policies: scheduling and dropping. Scheduling policy decides how a number of packet sources, usually queues, can share a single output channel. Dropping policy drops packets when the queue size is beyond certain threshold. We use the following two algorithms in our *RouterBench*.

1) DRR (Deficit Round Robin): DRR is a Deficit Round Robin fair schedulig algorithm [30] that is commonly used for bandwidth scheduling on network links. DRR assigns a quantum to each queue and serves the queues in a round robin fashion. A queue cannot send a packet if the packet size is larger than the quantum. After a packet is sent, its size is subtracted from the quantum. The reminder of the quantum is added to the quantum for the next round so that the large packet gets a chance to be sent out later. DRR algorithm is implemented in one form or another in currently available routers (e.g. Cisco 12000 series).

2) RED (Random Early Detection): RED is used to drop packets when there is network congestion. A link is considered congested when there are too many packets in the queue serving that link. RED monitors the average queue size of each output queue. Congestion in the link increases the average size of the queues. If the average size of a queue exceeds the maximum threshold, RED drops a packet with a probability that is proportional to the connection's share of the throughput, and notifies the sender about this congestion. RED is first proposed by Floyd and Jacobson in [10], and later implemented on various platforms. UCLADew [35] is one of the implementations of RED, and we port it to build our RED benchmark application.

DRR and RED are usually performed on LC instead of FE because FEs are responsible for processing packet headers, while LCs are responsible for queuing management. However, future high performance routers are likely to implement the QoS activities in the FE. Also, when an FE is physically located in an LC, like in Cisco 12000 routers[7], it has to perform QoS activities.

D. Miscellaneous

There are some other tasks that are executed in the critical path of processing a packet; however, it is not appropriate to put them into the above categories. These functions are included in both BBN MGR and Click router configurations. We run profiling tool on a Click software router and identify that these functions are consuming a non-negligible amount of time. This miscellaneous category consists of CheckIPHeader, DecTTL and Fragmentation.

1) CheckIPHeader:

- CheckIPHeader verifies the validity of an IP packet.
- It checks if the packet's length is reasonable, and that the IP version, header length, packet length, and checksum fields are valid.
- It also checks if the IP source address is a legal unicast address – the address should not be 0.0.0.0 or 255.255.255.255, or local broadcast addresses as the administrator defines.

2) DecTTL:

- DecTTL updates the time-to-live (TTL) field of a packet. TTL is used to track the lifetime of a packet in the Internet; a packet should not travel in the Internet for ever in the presence of network routing errors.
- If the TTL value is ≤ 1 (i.e. time-to-live has expired), DecIPTTL generates and sends back an Internet Control Message Protocol (ICMP) error notification packet to the sender. Otherwise it decrements the TTL, and recalculates the checksum.

3) Fragmentation:

- Fragmentation fragments packets to small ones when they are transfered to a network with smaller Maximal Transfer Unit (MTU). Link layer MTU of physical network, connected to the router, determines the size of fragments. For example, a typical MTU is 576 bytes for the X.25 network [19].
- It also recalculates checksum based on new size, and updates the header.

V. Simulation Results and Analysis

It is desirable to conduct performance evaluation based on real routing table from a backbone router with real packet trace from the same site. The only routing table and trace pair that is publicly available is from FUNET [24]. The trace includes near-real IP addresses (the last 8

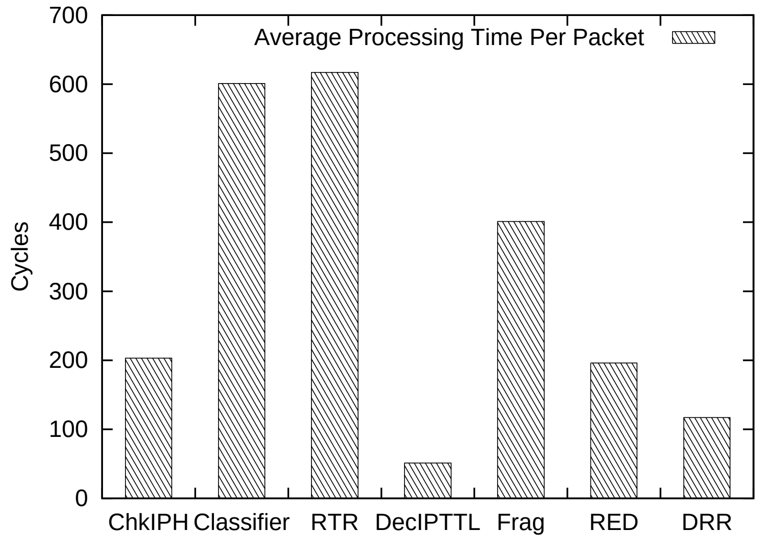

Fig. 7. Processing time distribution in packet processing path.

bits of an IP are masked to 0), and is acceptable for evaluating the RTR lookup because most route prefixes are less than 24 bits. The FUNET routing table has 41K entries, and the trace file contains 100K destination IP addresses. The other functions, such as CheckIPheader, Classifier, DecTTL and Fragmentation, need other IP header fields besides the destination IP. However, FUNET traces do not contain such information. Therefore we use traces from NLANR [20] because they have a full packet header up to TCP layer. On the other hand, NLANR traces have sanitized IP addresses that make them unsuitable for routing table lookup.

A. RouterBench Applications Performance

Fig. 7 depicts the per-packet execution time of each *RouterBench* application in a uniprocessor environment, where the processor has one level cache. The cache has 32-byte cache block, 32KB cache size and is two-way set-associative. Cache latency and memory access latency are assumed to be 2 cycles and 30 cycles, respectively. The experimental results show that classification and routing table lookup take up to 56% of the total processing time (2186 cycles). CheckIPHeader, DecIPTTL, Fragmentation, RED and DRR share the rest 44%. Classification and route table lookup have comparable execution time complexity. RTR route table lookup takes most of the time, 617 cycles per packet, and DecIPTTL takes the least, 51 cycles. The total processing time for a packet (excluding queuing time) is 2186 cycles. Assuming memory access cycles do not change with increase in CPU frequency, the above result corresponds to a maximal throughput of 91.5K packets/sec with a 200MHz CPU or 457K packets/sec with a 1GHz CPU. In reality, as we move to faster CPUs, the number of cycles for each memory access will increase proportionately. As a result, the packet processing time will increase and throughput will decrease. In any event, the throughput obtained in a uniprocessor is far below the multi-gigabit link requirement (multi-million packets/sec), and justifies the need for a multiprocessor architecture in routers. Hence, in the following subsections, we show how multiprocessor architectures can improve the performance of packet header processing applications.

B. Multiprocessor Router Architecture Performance

We individually run each benchmark application on the SMP and CC-NUMA architectures of our execution-driven simulator to get various performance measurements. Table I summarizes default parameters of our architectures. We adopt packet-level parallel processing in the sense that when a packet arrives at a line card, it is sent to a forwarding engine for processing in a round-robin manner. However, they all share the same shared memory.

TABLE I

ARCHITECTURAL PARAMETERS

	SMP	CC-NUMA
Cache Size	32KB	32KB
Cache Block Size	32B	32B
Cache Associativity	2-way	2-way
Cache Replacement	LRU	LRU
Cache Latency	2 cycles	2 cycles
Local Memory Access Latency	-	30 cycles
Bus Transaction Time	50 cycles	-
Switch Transaction Time	-	50 cycles

1) SMP Performance: Figure 8 plots the execution time of various *RouterBench* applications on an SMP architecture. Except for RTR, the execution time of all other applications in *Router-*

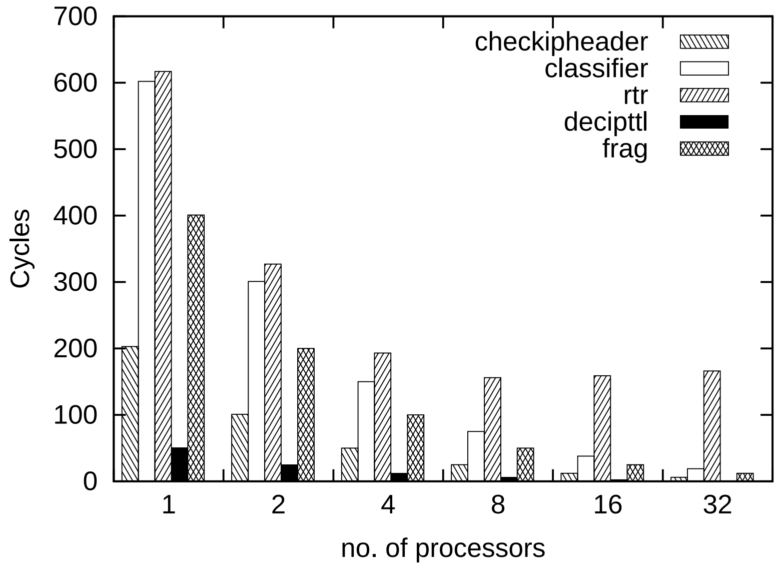

Fig. 8. RouterBench speedup on SMP architecture.

Bench decreases almost by half as the number of processors doubles. RTR lookup gets speed up until eight processors, after which its execution time increases.

In order to understand the cause of RTR performance trend on SMP, we further break down the RTR lookup execution time in Fig. 9. The computation time decreases up to 50% when the number of processors doubles, so does the memory stall time. Memory stall time is the time taken at the centralized memory of Fig. 3. The bus contention time is considered separately, which increases with the number of processors. This is because the radix tree is located in the centralized memory and every processor has to request and load the radix tree data structure via the bus when a cache miss occurs. Bus contention becomes the bottleneck in performance, which leads to the increase of total execution time after a certain number of processors.

Fig. 10 illustrates cache behavior of the RTR lookup algorithm on SMP. There are few write misses since we are not simulating the route update procedure of the radix tree structure. The read misses come from the fact that radix tree structure is allocated in the shared memory, and every processor has to traverse the tree structure to perform lookup. When a processor traverses a node in the tree for the first time, a cache miss occurs. The next visit to the same tree node will usually be a cache hit. The cache miss ratio is relatively low (between 3% and 4.5%) because the

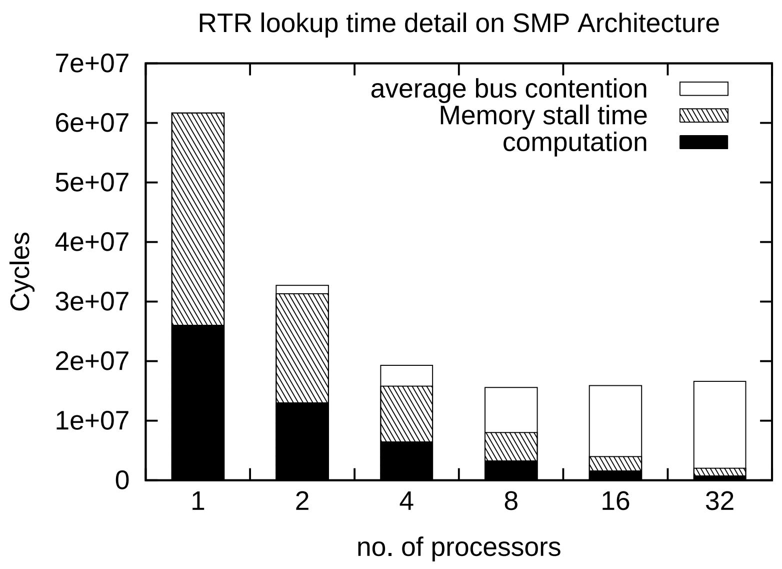

Fig. 9. RTR lookup time detail on SMP architecture.

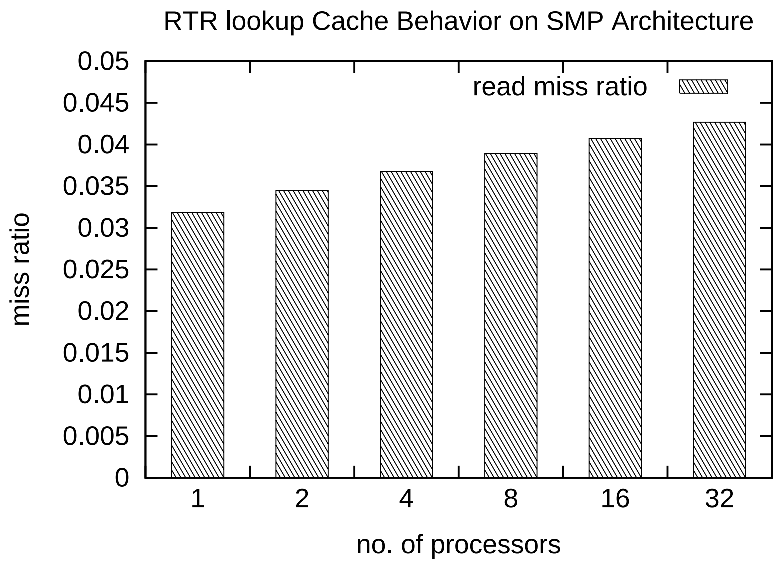

Fig. 10. Cache behavior of RTR lookup on SMP architecture.

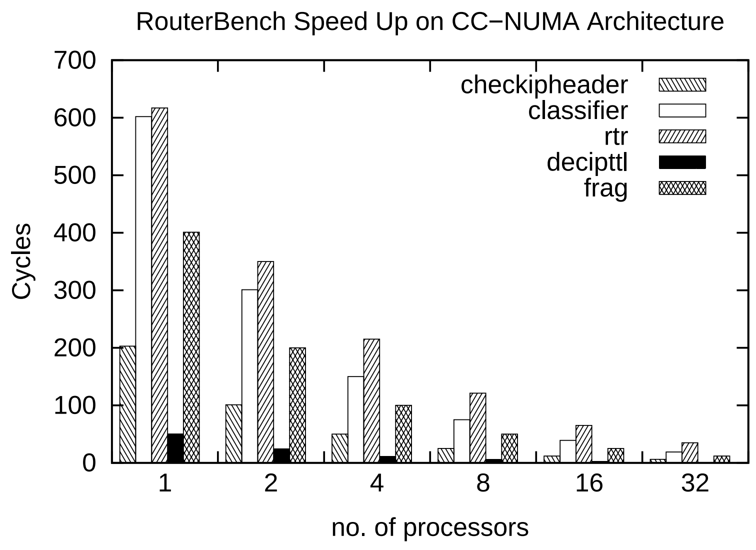

Fig. 11. RouterBench execution time on CC-NUMA architecture.

temporal locality in packet traces makes lookup procedure follow the same path along the radix tree. The miss ratio increases as the number of processors increases because arriving packets are distributed to all processors in a round robin fashion, and temporal locality in the trace as seen by each processor will be reduced. The reason of round robin distribution is that the LCs do not classify the incoming packets in our current design, thus no information is available to distribute a flow of packets to the same FE. If more computation power is invested into LCs, simple classification can be done and processors can be allocated to preserve locality between packet trains.

2) CC-NUMA Architecture Performance: A CC-NUMA architecture overcomes the shared bus bottleneck by providing a crossbar network, and by distributing the shared memory among different processors. Fig. 11 plots the execution time of the same *RouterBench* applications on CC-NUMA architecture. It shows that all *RouterBench* applications gain linear speedup as the number of processors doubles. Fig. 12 analyzes the RTR lookup execution time that consists of four parts: computation, local memory stall, remote memory stall and crossbar contention. As radix trees are created in shared memory, nodes of the tree may reside in one of the physical memory modules. If a processor accesses the node located in its own memory module, the access is a local memory access. Accesses to other nodes in the remote memory are accomplished

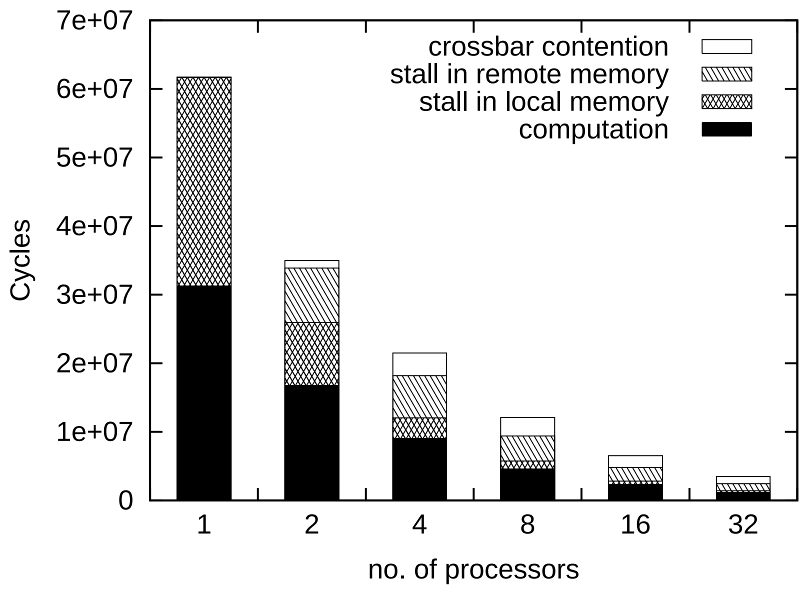

Fig. 12. RTR lookup time detail on CC-NUMA architecture.

through the crossbar switch, which has a high delay. However, the experimental data show that the crossbar mitigates the remote memory access cost by providing simultaneous accesses to several memory modules. As a result, the performance is much better than a bus-based SMP. It is also observed that the crossbar contention time increases until the number of processors equals four, after which the crossbar contention drops. This implies that, as we distribute the radix tree data structure to more processors, read accesses to the radix tree are also directed to all memory modules, instead of a particular one. Crossbar switch provides separate path when the accesses are to different memory modules, which leads to less contention. Hence, the results show that CC-NUMA has better scalability than SMP for RTR lookup algorithm.

VI. Impact of Routing Table Updates

In this section, we study the impact of routing table updates on performance of the CC-NUMA multiprocessor router architecture.

Routing tables are updated periodically by routing protocols such as BGP. The route in a routing table may be modified or deleted, or a new route entry may be added into the table. The frequency of route updates is known to be from hundreds to thousands of updates per second [12], [27]. As the future routing tables scale, we conjecture this number can reach up to 100,000 updates per second, which is used as a test case in performance study in [33]. In the following experiment, we study the impact of routing table updates at the frequency of 1K, 10K, 100K updates per second, which correspond to one update per 1M, 100K, 10K cycles for 1GHz processors.

There are three kinds of route updates: modification, deletion, and addition of a route. When a route entry is modified, its outgoing link number, priority or other route property are updated. From the perspective of radix tree, updating such information does not change the tree structure. Thus, the cost of such udpates is not significant. However, when adding a new route entry or deleting an existing one, the radix tree structure will have radix nodes allocated or deallocated, and pointers updated or deleted. Such updates to the radix tree have significant costs.

It is desirable that we use a BGP update trace from the same FUNET site because we use FUNET routing table and packet trace pair in the simulation experiments. However, such an update trace is not accessible. Although there are BGP route update traces available publicly, they are not suitable for our simulation experiment because the BGP updates are highly site-dependent. Hence, we construct a synthetic route update trace using FUNET routing table. We take the raw FUNET routing table with 41K entries, scan the table linearly and extract one route entry out of every 10 entries [1]. Thus we obtain a new routing table *T*, which is used for simulating route update messages. We define each entry in *T* as three consecutive update messages: a route deletion, an addition and a modification. It is assumed that such an update message sequence repeats for all the entries in *T*. We use a separate processor to execute routing table updating procedure, all other processors perform route lookups as they do in the case of an IP router [7], [27].

Fig. 13 shows the impact of route updates on the routing table lookup performance of a CC-NUMA architecture. The routing table updates generally degrade the lookup time. This is expected since route updates will increase memory traffic and delay the memory access from other processors. Also the routes stored in each processor's local cache may have to be invalidated by the updating processor, thus the next lookup will generate a cache miss. The route update frequency clearly affects the amount of degradation in lookup time. As shown in Fig 13, when

[1]The number of entris can be arbitrary, we choose 10 here.

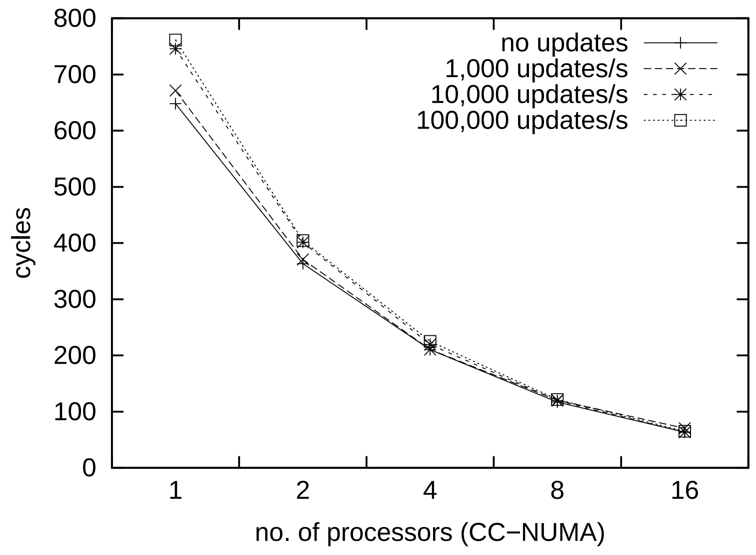

Fig. 13. Impact of route updates on average lookup time on CC-NUMA.

the number of processors equals to one, 1K updates per second cause an increase of 3.5% on average lookup time (from 648 to 671 cycles), 10K updates per second cause an increase of 15% (to 746 cycles), and 100K updates per second give rise to an increase of 18% (to 762 cycles).

As the number of processors increases, however, this lookup time degradation becomes insignificant. Especially, when the number of processors is beyond eight, the performance degradation is not noticeable. This is because although route updates affect the latency of one memory module, other modules operate without any conflict. Also, there are several caches which maintain copies of radix nodes, so the memory contention due to update and cache misses is reduced. When a route update message is received, the updating processor locates the appropriate radix node in the tree structure to allocate new node and append it, deallocate a node, or make modification. These updates only invalidate the particular cache block in the processor that has such a copy. As the number of processors increases, each processor tends to keep a different portion of the tree structure in its cache. As long as the invalidated block is not shared by many processors, the overall lookup performance does not degrade. Hence, a CC-NUMA architecture can sustain high lookup performance even in the presence of highly frequent route updates.

VII. Conclusion

In this paper, we proposed two shared memory multiprocessor architectures, SMP and CC-NUMA, to meet the increasing line speed of future IP routers. Such architectures not only save memory space for storing routing table, but also have the advantages of general purpose multiprocessor architectures, which include programmability, scalability and high performance.

We developed an execution driven simulation environment to evaluate the proposed SMP and CC-NUMA router architectures. A snoopy cache coherence protocol was incorporated into the SMP architecture to maintain coherence among caches. The CC-NUMA architecture is designed based on a directory protocol. To quantitatively analyze the performance of such routers, we proposed a benchmark application suite, *RouterBench*, which consists of key functions in the time-critical path of packet processing. The source code of this benchmark suite is publicly available at [17]. Experimental results showed that classification and route table lookup executions take up to 56% of the total processing time of a packet. CheckIPHeader, DecIPTTL, Fragmentation, RED and DRR share the rest 44%.

The SMP and CC-NUMA architectures improve *RouterBench* performance as the number of processors increases. However, the routing table lookup performance in SMP improves until the shared bus becomes saturated. On the other hand, the CC-NUMA architecture provides an excellent scalability for the design of high performance routers. We showed different components of the execution time and their variation with the number of processors for the RTR lookup algorithm when executed on both SMP and CC-NUMA architectures.

Finally, we studied the impact of routing table updates on the performance of the CC-NUMA multiprocessor router. The route updates degrade route lookup performance due to the cache block invalidation generated by the updating processor. However, this degradation becomes less significant as the number of processors increases. This implies that the CC-NUMA architecture can sustain high lookup performance even at a high frequency of route updates.

Acknowledgement

We would like to thank Li Zhao, Vaishnavi Krishnamurthy and Eugene R. Ulfig for their valuable comments.

REFERENCES

[1] P. Almquist, *Type of Service in the Internet Protocol Suite,* Internet RFC 1349, July 1992.

[2] F. Baker, *Requirements for IP Version 4 Routers,* Internet RFC 1812, June 1995.

[3] L. Bhuyan and Y. Chang, *Cache Memory Protocols,* Chapter in Encyclopedia of Electrical and Electronics Engineering, Feb. 1999.

[4] L. Bhuyan and H. Wang, *Execution-Driven Simulation of IP Router Architecture,* IEEE International Symposium on Network Computing and Applications (NCA'01), Boston, October 2001.

[5] H.C.B. Chan, H.M. Alnuweiri, and V.C.M Leung, *A Framework for Optimizing the Cost and Performance of Next-Generation IP Routers,* IEEE Journal On Selected Areas in Communications, Vol. 17, No. 6, pp1013-1029, June 1999

[6] B. Chen and R. Morris, *Flexible Control of Parallelism in a Multiprocessor PC Router,* USENIX'01

[7] *Cisco 12000 Series Internet Router Data Sheet,* Cisco Systems, 2001.

[8] *Cisco 12000 Series One-Port 10-Gigabit Ethernet Line Card Data Sheet,* Cisco Systems, 2002.

[9] P. Crowley, M. Fiuczynski, J. Baer, and B. Bershad, *Characterizing Processor Architectures for Programmable Network Interfaces,* Proceedings of the 2000 International Conference on Supercomputing, Santa Fe, NM, May, 2000.

[10] S. Floyd and V. Jacobson, *Random Early Detection Gateways for Congestion Avoidance,* IEEE/ACM Trans. On Networking, Vol. 1, pp.365-376, August 1993.

[11] P. Gupta and N. McKeown, *Algorithms for Packet Classifi cation,* IEEE Network Special Issue, March/April 2001, vol. 15, no. 2, pp 24-32.

[12] P. Gupta, S. Lin, and N. McKeown, *Routing Lookups in Hardware at Memory Access Speeds,* Infocom'98

[13] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach,* Morgan-Kauffman, 1995

[14] G. Huston, *Internet Performance Survival Guide - QoS Strategies for Multiservice Networks,* Wiley Computer Publishing, 2000

[15] S. Karlin and L. Peterson, *VERA: A Extensible Router Architecture,* Proceedings of the 4th International Conference on Open Architectures and Network Programming (OPENARCH'01), Anchorage, Alaska, April 27-28, 2001.

[16] E. Kohler, E. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, *The Click Modular Router,* ACM Transactions on Computer Systems 18(3), August 2000, pages263-297.

[17] Y. Luo and L. Bhuyan, *RouterBench - A Benchmark for High Performance IP routers,* http://www.cs.ucr.edu/∼cial/routerbench/

[18] G. Memik, B. Mangione-Smith, and W. Hu, *NetBench: A Benchmarking Suite for Network Processors,* Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD) - 2001, San Jose / CA, Nov. 2001.

[19] J. Mogul and S. Deering, *Path MTU Discovery,* Internet RFC 1191, November 1990

[20] National Lab of Applied Network Research, *Sanitized Access Log,* http://pma.nlanr.net/Traces/, May 2002.

[21] NetBSD Foundation Inc., *NetBSD 1.3 Release,* January 4, 1998.

[22] A-T. Nguyen, M. Michael, A. Sharma, and J. Torrellas, *The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures,* Proc. of 1996 International Conference on Computer Design, October 1996.

[23] K. Nichols, S. Blake, F. Baker, and D. Black, *Defi nition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,* Internet RFC 2474, December 1998.

[24] S. Nilsson and G. Karlsson, *IP-Address Lookup Using LC-Tries,* IEEE Journal on Selected Areas in Communications, Vol. 17, No. 6, June 1999, pp. 1083-1092. (http://www.nada.kth.se/∼snilsson/public/code/router/Data/)

[25] V.S.Pai, P. Ranganathan, and S.V. Adve, *RSIM Reference Manual*, Technical Report 9705, Dept. of Electrical and Computer Engineering, Rice University, August 1997.

[26] K. Papagiannaki, S Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot, *Analysis of Measured Single-Hop Delay from an Operational Backbone Network,* IEEE INFOCOM'02

[27] C. Patrick et al., *A 50Gbps IP Router,* IEEE/ACM Trans. On Networking, Vol.6, No.3, pp.237-248, June 1998.

[28] Y. Rekhter and T. Li, *A Border Gateway Protocol 4 (BGP-4),* RFC - 1771,March 1995.

[29] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, *Survey and Taxonomy of IP Address Lookup Algorithms,* IEEE Network, March/April 2001, pp 8-23.

[30] M. Shreedhar and G. Varghese, *Efficient Fair Queuing using Deficit Round Robin,* Proc. of SIGCOMM 95, ACM, Cambridge, Mass.

[31] Keith Sklower, *A Tree-Based Packet Routing Table for Berkeley Unix,* USENIX Winter 1991: 93-104

[32] T. Wolf and M. Franklin, *CommBench - a Telecommunications Benchmark for Network Processors,* In Proc. of IEEE International Symposium on Performance Analysis of Systems and Software, pages 154-162, Austin, TX, Apr. 2000

[33] David E. Taylor, *User Guide: Fast IP Lookup (FIPL) in the FPX,* Gigabit Kits Workshop Jan. 2002, http://www.arl.wustl.edu/arl/projects/fpx/workshop_0102/fipL_user_guide_v2.pdf

[34] T. Wolf and J. Turner, *Design Issues for High Performance Active Routers,* IEEE Journal of Selected Area in Communications, Vol.19, No.3, pp 404-409, March 2001

[35] L. Zhang and S. Michel, *UCLADew,* http://irl.cs.ucla.edu/UCLAdew

Yan Luo received his BE and ME in computer science and engineering from the Huazhong University of Science and Technology in 1996 and 2000, respectively. He was a technical staff member in the R&D Center of Guangdong Nortel Co. from 1996 to 1997. He is currently a PhD candidate in the Department of Computer Science and Engineering at University of California, Riverside. His research interests include network processor, router architecture, parallel and distributed processing, and cluster/grid computing.

Laxmi Narayan Bhuyan is a professor of Computer Science and Engineering at the University of California, Riverside since January 2001. Prior to that he was a professor of Computer Science at Texas A&M University (1989-2000) and Program Director of the Computer System Architecture Program at the National Science Foundation (1998-2000). He has also worked as a consultant to Intel and HP Labs. Dr. Bhuyan's current research interests are in the areas of network processor architecture, Internet routers, and parallel and distributed processing. He has published more than 100 papers in related areas in reputed journals, and conference proceedings. His brief biography and recent publications can be found at his web page at http://www.cs.ucr.edu/∼bhuyan/. Dr. Bhuyan is a Fellow of the IEEE, a Fellow of the ACM, and a Fellow of the AAAS.

Xi Chen received the BE degree in computer science and engineering from Zhejiang University in 2000, the MS degree in computer science from University of California at Riverside in 2002 and is currently a PhD candidate in computer science from University of California at Riverside. His research interests include verification of embedded system designs, system-level design methodologies, and distributed computing.