Network Processor based Router and the Cache Design: Implementation and Evaluation

Yeim-Kuan Chang and Kai.-Ming. Hsu

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan R.O.C.
ykchang@mail.ncku.edu.tw

Abstract—**High performance routers are mostly implemented with network processors because of their software programmability, hardware computation power, and high bandwidth interface design. In this paper, a 5-dimensional packet classification algorithm based on the hierarchal binary prefix search is first implemented in IXP1200 network processor. Our classification implementation is faster and smaller than other existing schemes and makes it possible to put entire rule table in SRAM. Moreover, we proposed a cache mechanism for IXP1200 because we observed that the traffic patterns of backbone routers have a strong temporal locality. Our proposed cache scheme not only caches the results from packet classification but also caches the results from IP lookups. Only one SRAM read is needed to perform IP lookups and packet classification for a cache hit. With this cache mechanism, the throughput of our system is very close to the theoretical maximum bandwidth with a reasonable hit ratio. Specifically, with a cache of 8192 rule entries, the proposed cache mechanism has 50% improvement in throughput over the system with no cache.**

Keywords—**IXP1200 network processor, binary prefix search, IP lookups, packet classification, and cache..**

1 Introduction

With the evolution of network technologies, the requirements for the routers have become higher. Traditionally, there are two major types of router implementations: software and hardware. In software-based routers, the whole routing process is programmed and run on general-purpose processors. They can support new services by writing codes and updating the software. In hardware-based routers, Application-Specific Integrated Circuit (ASIC) chips are designed to support higher processing power. However, designing and manufacturing ASIC chips are expensive and time comsuming and lack the flexibility of adding new services.

Nowadays, network processors are emerging as an alternative solution to ASIC for providing scalable capability for user-plane packet processing while retaining programmability. Network processors typically consist of an embedded control processor and several data processing engines. The control processor is responsible for executing the control plane functionality (e.g., routing table maintenance), whereas data processing engines perform the data-plane operations (e.g., IP Lookups). An example of such a network processor is the Intel IXP1200 (Internet Exchange Processor), which consists of one StrongArm core and six co-processors, known as microengines. Each microengine can execute up to four threads and its instruction set is specially designed for packet processing.

In addition to the general data-plane operations, many routers support packet classification. Today, there are many layer-4 switching technologies such as Resource Reservation Protocol, differentiated services, and quality of service. All require the routers to classify the packets into different flows and then perform appropriate actions. The packet classification is supported according to pre-defined rules. Typically, these rules based on header fields in layers 2, 3, and 4. Rule match may be the exact matching or prefix/range matching on multiple fields. IP lookup is a special case of one dimensional classification. A multi-dimensional classification includes more than one field and the packet to be processed should be matched with all of these fields.

In this paper, we use the RadiSys ENP-2505 evaluation board [1] that consists of one IXP1200 network processor chip and four Ethernet ports (4*100Mbps) as our development environment. Our router design is based on RFC1812. In addition, we used a multi-dimensional classification algorithm based on binary prefix search and implemented a standard five-dimensional classifier (including IP source address and destination address fields, transport protocol fields, and source and destination port fields) in IXP1200. The initial results show that the performance of classification based on binary prefix search is faster and more scalable than other schemes. However, our system still needs more input processing power for achieving the line speed. This means we need allocate more microengines for input processing. For reducing the resources requirement in IXP1200 and leaving ample headroom for additional/faster ports, we propose a layer-4 cache design for IXP1200 because we observed that backbone router have strong temporal locality. Our proposed cache scheme caches the results not only from packet classification but also from IP lookups. Only one SRAM-read is needed to perform IP address Lookups and packet classification procedure for a cache hit. With this cache mechanism, our system performance can be improved significantly.

The rest of the paper is organized as follows. Our router implementation augmented with a 5D classification algorithm based on binary prefix searches is illustrated in section 4. The performance results are also included. To further optimize the performance of our router, we design a layer-4 cache for IXP1200 and show the performance improvement in section 5. The final conclusion of this paper is given in the last section.

2 Implementation of IXP1200 Router

The functional specification of our router implementation is based on RFC 1812 [1]. The main functionality of the router includes the following: packets with invalid address, invalid IP version numbers, or TTL=0 and broadcast packets are dropped. Packet header checksum is calculated and the packet is dropped if the checksum is invalid. After decrementing the valid TTL and recalculating checksum, the packet is routed to the output port by performing the IP lookups.

In addition, our router also implements packet classification. For convenience, we name this router as IXP Router. Figure 2.1 shows the software architecture of IXP Router, which follows the IXP1200 ACE programming model.

Most ACEs components in the figure are supported by IXA SDK except the Classifier ACE. All microblocks in microengines are implemented as Micro C functions. We use two microengines for processing packets in data-plane, one for receiving packets and the other for transmitting packets. The tasks of Receive/Transmit microengine threads are listed in table 2.1.

Table 2: microengine thread assignments

microengine	Thread	Port/Task Assignment	Comments
Receive microengine	Thread 0, 1, 2, and 3	4 Receive Thread	Thread i to receive port i for i = 0 .. 3
Transmit microengine	Thread 0	Transmit Scheduling	microengine 4 transmits on four ports (0~3). (One scheduler thread and three transmit threads)
	Thread 1,2, and 3	Three dynamically assigned Transmit Threads	

The Classifier ACE builds the special data structure based on rule table. Then the Classifier microblock classifies packets into different flows according to the data structure built by Classifier ACE. The major task of L3 Forwarder MicroACE is to perform IP lookups and then forwards packets to appropriate output ports. Based on IXP SDK, we adopt multibit tries of 4-bit stride [16] as the IP lookups algorithm. The L3 Forwarder microblock focuses on searching the next hop route information. For certain types of packets (e.g. packets with IP options in the header, fragmented packets, ARP, etc.), this microblock sends them to the L3 Forward ACE for performing appropriate action. These packets are called "exception" packets.

Packet Flow in IXP Router:

Receiving and transferring packets are two basic tasks of the router. When a packet arrives, it is divided into several 64-byte chunks called mpackets and put into SDRAM. In order to reassemble the packet, each mpacket can be identified as the start of the packet (SOP), the end of packet (EOP), both, or neither. The packet in SDRAM will be serviced by another application (i.e. IP Lookups procedure) and be assigned outgoing port number. Finally, the mapckets are put into the outgoing MAC buffer sequentially. The outgoing MAC devices transmit the complete packet when detecting it is EOP.

Packet Classification Algorithm

In this subsection, we shall illustrate the multi-dimensional classification algorithm that is implemented in IXP1200. First of all, we briefly describe the *binary prefix search* [3] that is foundation of the classification algorithm. Then we describe the details of the multi-dimensional packet classification. Finally, the performance of classification algorithm is evaluated on the platform of IXP1200.

Binary Prefix Search

To apply the binary search in a set of prefixes, two problems must to be taken into account. The first one is that the binary search works only for sorted lists. We must have a mechanism that can compare and sort the prefixes. Therefore, the comparison rule defined in [3] is given as follows.

The inequality 0<<1 is used to compare two prefixes in the ternary representation of prefixes.*

Example 1: Given three 8-bit prefixes in ternary format, A=0000-0***, B=0000-01**, and C=10**-****. Based on the definition, we have A < B < C.

The second problem is that *prefix enclosure* defined in [3] may result in multiple matches. From the same example above, an 8-bit stream "0000-0101" will match not only prefix A but also prefix B. Prefix B is enclosed by prefix A. The characteristic of "prefix enclosure" may make the exact match search algorithms such as binary search fail. To solve this problem, the sorted array used by binary search must be constructed by the *enclosure split process*, which includes the following steps:

Step 1. Build the binary trie according to the prefixes set.

Step 2. Make the binary trie complete.

Step 3. Store all leaf nodes of binary trie in an array sequentially by an inorder traversal.

Step 4. Perform merge for compressing the prefix array.

We give a simple example to illustrate above steps in detail. Assuming there are four prefixes (A=1*, B=000*, C=11*, and D=111*). First, we build the binary trie shown in Fig. 2.2(a) and then convert this binary trie into a complete binary trie shown in Fig. 2.2(b). In the complete binary trie, all prefixes are stored in the leaf nodes. The prefix enclosure problem is now removed because all the prefixes from the complete binary trie are disjoint with each other. All the prefixes in leaf nodes are stored in a prefix array by performing an inorder traversal (Fig. 2.2(c)). Note that enclosure split process may generate some auxiliary prefixes. To decrease the number of auxiliary prefixes, "prefix merge" scheme [3] was proposed for handling this problem. It tells us that the consecutive prefixes can be merged, if they are generated from the same prefix. Take Fig. 2.2(c) as our example. Because the second and the third entry are both generated from prefix A, they can be merged together. Fig. 2.2(d) shows the merged results. After "prefix merge" scheme, we can apply the binary search to look for the longest prefix search. Detailed search procedure for merged prefix array can be found in [3]. Notice that, when the search reaches the final two prefixes, both prefix need to be matched against the input IP. The longest prefix of the two is matched first. If the longest prefix matches the input IP, then it is done. Otherwise, the other prefix is matched against the input IP. It is possible that both prefixes do not match the input IP.

Multi-Dimensional Packet Classification

Our goal is to construct a standard five-dimensional classifier including IP source address and destination address fields, transport protocol fields, and source and destination port fields. The IP source address and destination address fields are in prefix format, which can apply binary prefix search directly. The source port and destination port fields may form in range format. The ranges are converted into prefix format using the technique described in [18].

Now we illustrate how the binary prefix search can be applied to the multi-dimensional packet classification. Assuming a d-dimensional rule is in a form of *ri=(F1i,…Fdi)*, where Fik, called the Kth filter, is a variable length prefix bit string. We use r = (F1, … Fd) when no confusion is incurred. The proposed d-dimensional classification algorithm is described as follows.

1. Build the binary based on F1 of the rule table.
2. Push all the sub-rules (F2, … Fd) of enclosing nodes to their descendant nodes of the F1 binary trie. We call this operation the *rule pushing* step.
3. Perform the enclosure split process based on F1 to construct the sequential list of sorted nodes. Now each node contains multiple entries of sub-rules (F2, … Fd).
4. Continue steps 1, 2, and 3 for each dimension except that the rule push step (step 2) is not required for last dimension.

Consider the 2D rule table consisting of A=(*,01*), B=(000*,000*), C=(001*,*), and D=(0*,111*). We first build the binary trie according to the field 1 of classifier shown in Fig. 2.3(a). Secondly, the rules of enclosing prefixes are duplicated and pushed to their descendant prefixes as shown in Fig. 2.3(b). And then the enclosure split process is performed on the field 1 binary trie with rule duplications.

This example does not introduce any auxiliary nodes. Now the sequential list of the rules based on filed 1 is completed.

Each element of the sequential list constructed so far contains the field 2 rule information. We need to construct the binary tries for all the nodes based on filed 2. Fig. 2.3(c) shows the binary tries constructed. Since this the last dimension, no rule pushing is required. We then perform enclosure split process to add the necessary auxiliary prefixes of field 2. Fig. 2.3(d) shows the 2-D view of the structure, which is implemented, in a 2-level hierarchy shown in Fig. 2.3(e).

Assume the first two fields of the incoming packet are (0000, 0000) and the rule set is the one described above. The first-level list in Fig. 2.3(e) is first checked with field 1 that is 0000. Binary search finds that the longest prefix match is B. Then following the pointer from B, binary search is performed using field 2 that is 0000. Field 2 of B is matched. Therefore, the matched rule is B.

So far, we have described how to apply binary prefix search to multidimensional classifier and give an example of 2D classifier. In our works, we implement a 5-dimensional classifier for IXP1200 by constructing five sorted prefix array. The hierarchical sorted prefixes are in the order of protocol, destination address, source address, destination port, and source port. In the first 4 levels of the hierarchy lists, we only need longest prefix match, without considering the priorities of the rules. Only at the least level, we need to determine which rule's priority is the highest.

Performance Evaluation

In order to evaluate the performance and scalability of our scheme, we use synthetic rule tables of various sizes. These synthetic tables are generated by using ClassBench, which is a publicly available tool for benchmarking packet classification algorithms [20]. Table 2.2 shows the memory requirements of the proposed 5-dimensional scheme. In addition, we also implement 2-D version of Grid of Trie [18] to compare with 5-D version of the proposed scheme. All experiments are evaluated by using IXP NetBench, a traffic generator/analyzer developed by our lab.

Table 2.2 Memory required by proposed scheme with synthetic rule tables.

Table Name	size	Memory Requirement	
		Proposed 5D classifier	GOT 2D classifier
50_rule	50	2.96KB	20.99KB
100_rule	100	4.79KB	31.57KB
500_rule	500	22.22KB	92.46KB
1000_rule	1000	43.29KB	189.88KB

Grid-of-Tries

Memory requirement is the critical issue for implementing a classification algorithm on the platform of IXP1200 because of the limited SRAM size (only 4MB are available in our evaluation board). We surveyed several packet classification algorithms in [4]. By considering the requirement in both memory and performance, we think the Grid-of-Tries may be suitable in IXP1200 because it avoids backtracking problem of the hierarchical tries [18] and reduces the storage requirement of the set purring tries [18]. Therefore, we implemented 2-D version of Grid-of-Tries and evaluated its performance. Note that we set a simple 2D version of rule table for testing the Grid-of-Tries in this part and use IXP NetBench for generating traffic streams with different size of packets. Fig. 2.4 shows the experimental results. It can be seen that IXP Router cannot perform very well with small packet sizes. We doubt that the bottlenecks of our system occur in receiving process. However, in hardware mode, we can only get the router's throughputs by IXP NetBench. But in simulation mode, the Developer Workbench can provide detailed processing information such as memory and thread histories, memory reference latencies and cycle-by-cycle interactions among the threads and memory units. This information is very useful for analyzing our system. Therefore, we decide to execute IXP Router in simulation mode and divide the tasks of Receive microengine into five categories for determining which one needs maximum processing time and is the bottleneck. These five categories are Receive Packet, Classifier (2D Grid-of-Tries), IP Lookups (multibit tries of 4-bit stride), Miscellaneous, and Queuing .

The queuing category has two types of actions: queuing packet for slow-path processing (StrongARM) or queuing packet for fast-path processing (microengines). The miscellaneous category consists of Layer 2 Header Validation, Layer 3 Header Validation and TTL validation. They are described as follows.

1. Layer 2 Header Validation: Ensure the destination MAC address of the datagram matches the destination MAC address of the interface where the datagram is received. (RFC1812, Section 5.3.4). If a mismatch occurs, check the Ethernet type, if it equals 806 (ARP request) and L2 destination MAC address is broadcast, this packet is queued for slow-patch exception handling (StrongARM exception handling). Otherwise, drop this packet.
2. Layer 3 Header Validation: Checking the packet's length is reasonable and that the IP version, header length, packet length, and checksum fields are valid. Checking IP source/destination address is legal unicast address.
3. TTL Validation: If the TTL value is smaller than one, this packet is queued for slow-patch exception handling. Otherwise, it decrements the TTL and recalculates the checksum.

Now, we can gather the required processing time of each category by analyzing the thread execution history using Developer Workbench. The simulated results are shown in Fig. 2.5. Only the task of receiving packet will be influenced significantly with bigger packet size. Not surprisingly, the Classifier is the heavy task of Receive microengine. Each search almost requires 3500 clock cycles that take up 70% processing time for a 64-byte packet. Therefore, the packet classification algorithm, Grid-of-Tires, can not perform well in IXP1200. Although Grid-of-Tries bounds memory usage to O(NW) but the search time is O(dW), where N is the number of filters and W is the maximum number of bits specified in the source or destination fields. For the case of searching on IPv4 source and destination address prefixes, it may need 65 memory accesses in the worst case. In this environment, we set prefix length "30" in the rule table, which is very closely to the worst case. In [20], David E. Taylor and Jonathan S. Turner performed a battery of analyses on 12 real rule tables provided by Internet Service Providers (ISPs). Unfortunately, we find that the proportion of maximum address prefix length (32) takes up higher percentage in real rule table. It should be concluded, from what has been said above, that trie based packet classification algorithms are not suitable in IXP1200. Therefore, we didn't spend more time on implementing the EGT (5D-D version of Grid-of-Tries), which may requires 84 to 137 memory accesses per lookup reported as in [2].

Scalability Test

To conduct the scalability test, we set microengine 0 as Receive microengine and microengine 4 as Transmit microengine for processing packets. One input traffic stream is generated by IXP NetBench with different size of packets for analyzing the throughput based on different rule tables. Note that the packet pattern of traffic stream must be predefined based on the synthetic rule tables and the forwarding table. This test is to investigate the scalability of our proposed scheme. Fig. 2.6 shows the experimental results. We can find that the throughputs of our system are very closely with different size of rule tables. Observably, our proposed scheme has high scalability. One reason is that our proposed scheme is logarithmic search method. When increasing the number of rules in rule table, the processing time won't be linearly increased but logarithmically increased. The other reason is that most rules are independent with each other; the enclosing rules are not very popular in the common case. Specifically, our proposed scheme for 500 rules achieves throughput of 38.9 Mbps at minimal size of packets, which is much better than the result reported in [18].

Multi-Threads Test

As described in section 3, the design of our system takes advantage of multi-treads architecture of IXP1200. Each thread of receive microengine services specific port of our system. Above test involves only one traffic stream transmitted by one port of IXP NetBench. In this kind of situation, only one thread of Receive microengine will be waked up to service this traffic stream. To wake up the other threads of Receive microengine and evaluate the maximum throughput of our system, we configure several traffic streams (1~4) by IXP NetBench and transmit them to different ports of router. Fig. 2.7 shows the experimental results. Naturally, the throughout of two threads is approximately two times of the one of a single thread. However, due to the lack of the computing power, the throughout of three or four threads in a microengine is not three or four times of the performance by only one thread.

Multi-microengine Test

For improving the performance of packets of smaller sizes, increasing the processing power is the straightforward method. Above test only uses one Receive microengine for dealing with the input packets. In this part, we configure two microengines (micorengine 0 and microengine 1) as Receive microengines. In this way, one port can be serviced by two receive treads concurrently. Note that we must increase several mutex operations in our code for coordinating threads with competitive problem. Figure 2.8 shows the experimental results. We can find that the throughout of two microengines approximately achieves to maximum transmit speed of IXP NetBench. Increasing the computing power can improve the performance of our system. However, it also increases the used resources of IXP1200. For reducing the used resources of IXP1200 and leaving ample headroom for additional/faster ports, we propose a layer-4 cache mechanism for IXP1200 in next section.

3 Cache Design for IXP1200

Cache has been proved to be a very effective technique to improve memory access speed. Nowadays, on-chip cache memory is usually designed for general processor (also called CPU cache). However, in IXP1200, the microengines do not have cache memory but they share three different memory interfaces: SRAM, SDRAM and Scratchpad. Our idea is to design a cache mechanism in IXP1200 for improving packet classification and IP lookups. To select the appropriate memory interface for caching the layer-3 and layer-4 information, we measured the latencies for reading/writing different size of data based on different types of memories. The SRAM read/write speed is faster than SDRAM and very close to scratchboard memory and the size in IXP1200 is big enough to hold layer routing tables and classification rule tables. Therefore , we selected SRAM as cache memory.

Fig. 3.1 illustrates the functional blocks of our caching approach for IXP1200, which is similar to CPU cache. Each cache block consists of one or more cache entries, which are the basic unit for storing cache information including Flow-ID and searching results (from classification and IP lookups). Our cache approach consists of three major steps that are described as follow.

1. When a packet arrives, the packet header information (Flow-ID) is used to generate cache index via hash which will be decreased later. With this cache index, we can get the number of cache block and the meta information of cache block.
2. As long as the incoming Flow-ID can match one cache entry in the cache block (cache hit), no further searches are needed; that is, this cache entry gives us the layer-3 and layer-4 forwarding information: Classifier Action ID (result from packet classification procedure) and Next Hop Information Pointer result from IP lookups procedure).

If the incoming Flow-ID mismatches all cache entries in this cache block (cache miss), this packet will be forwarded by running packet classification and IP lookups procedure. Besides, the mismatched cache block also needs to be updated.

Hit Ratio Analysis

The simplest way to evaluate cache performance is to analyze the cache hit ratio. Since backbone routers are considered as the most critical points, we obtained a traffic trace from OC-48 backbone routers provided by the NLANR PMA project [13]. This traffic trace has 42,870,847 entries, including layer-3 and layer-4 information (source IP/port, destination IP/port and protocol), which are useful for our investigation.

Several components may affect cache performance such as cache size, hash functions, cache associativity, and replacement policies. We will consider whether these components have major impact on cache hit ratios.

1. **Cache Size:** The cache size is equal to the product of the number of cache entries and the cache entry size. The cache entry is the basic unit for storing cache information which is composed of two fields. Field *Flow-ID* is used to store flow information such as source IP address (src_ip), destination IP address (dst_ip), source port (src_port), etc. The other field, *Results*, used to record the results from packet classification and IP lookups. These two fields influence the size of cache entry. If each flow is identified by 2-dimensional fields (src_ip,dst_ip), it only needs 8 bytes for storing Flow-ID. However, in our system, each flow is identified by 5-dimensional fields; each cache entry needs 16 bytes for storing Flow-ID and information of results. In general, the more cache entries is in the system, the higher is the cache-hit ratio. However, we can not use the entire SRAM for caching because SRAM need also be used for storing other data structures such as rule table for packet classification and IP lookups. We propose to reserve 128KB SRAM as our cache memory, which is enough for storing 8192 cache entries.
2. **Hash function:** Remember that packet header information is used to generate cache index via hash. Without an effective hash function, a large number of collisions may occur and degrade the cache performance. Traditional hash functions, such as SHA-1, MD5, are popular because they produce well-balanced output in which a single bit change in the input can change every bit of the output with equal probability. However, in IXP1200, SHA-1 or MD5 hash functions are not supported in hardware. Therefore, designing a hash function with less computation to achieve approximate performance of MD5 or SHA-1 will be helpful for our cache scheme. We propose two simple hash algorithms, XOR based hash and ADD based hash, which are shown in Fig. 3.2.
3. **Cache associativity:** Increasing associativity is a widely used technique in CPU cache to reduce the impact of conflict misses and increase hit rate. However, our cache approach is to put cache information in SRAM, getting or updating cache information must issue memory references to SRAM. In IXP1200, it is necessary to use SRAM

transfer registers for communicating between microengines and SRAM unit. Nevertheless, in terms of microengine, each thread only can access 8 SRAM read/write transfer registers, which result in the maximum transfer size to be 32-byte (8*4-byte) in one memory access. For this reason, we consider that direct mapped or 2-way set associative cache is suitable for our cache approach because only one SRAM read or write is needed for getting information from cache or updating information to cache.

4. **Cache Replacement Algorithm:** From Fig. 3.1, the cache index is the pointer used to get the number of cache block. If we adopt n-way set associative cache scheme (Typically, n = 2, 4, 8 etc.), each cache block has n cache entries. Once a cache miss occurs, it is necessary to choose one cache entry and replace it with the new one. We apply three conventional replacement algorithms (first in first out (FIFO), Least-Recently-Used (LRU), and random) to 2-way set associative cache scheme for evaluating the impact on cache hit ratio in our system.

Trace Simulation

So far, we have described our cache structure and several factors that may influence cache hit ratio. In this subsection, we use the collected traffic trace as input to analyze cache hit ratio based on these factors. Due to the space limit, only results for XOR are illustrated in table 3.1.

First, consider the performance of hash functions. The ADD based hash is better than XOR based hash when cache entries are larger than 512. On the contrary, the XOR based hash is better than ADD based hash function with small cache entries. In summary, the performance of simple hash functions is almost equal to that of MD5 hash across all conditions.

Secondly, we focus on the impacts of cache-hit ratio with different cache associativity. Not surprisingly, the 2-way set associative cache is better than direct mapped cache by 2.5% to 5.6%. But the differences are not so significant; we cannot ensure the performance of 2-way set associative cache is better than direct map. This is because 2-way set associative cache has two cache entries, which needs more clock cycles for getting/updating information from/to cache memory. In addition, it also needs double time for comparing the information of cache entry.

Finally, we consider different replacement polices, the FIFO is better than random in most of situations. As to LRU, it is better than FIFO and random by 0.4% to 3.7%. However, implementation of LRU algorithm is more complex than FIFO, this is because LRU needs to record the cache entry which is least used and updates this information to cache. In other word, LRU needs to update the information (issue a SRAM write) to cache whether the cache hit or not, but FIFO only needs to update that for a cache miss.

Table 3.1: cache-hit ratio based on XOR hash function with different cache-size and replacement algorithms.

Cache Entry/Size	Direct map	2-way FIFO	2-way Random	2-way LRU
64/1KB	0.431586	0.464565	0.454718	0.481495
128/2KB	0.503763	0.536422	0.526164	0.55388
256/4KB	0.56481	0.593504	0.585397	0.615631
512/8KB	0.622608	0.642971	0.63848	0.672233
1024/16KB	0.682031	0.695614	0.694472	0.730671
2048/32KB	0.740807	0.751845	0.751835	0.786135
4096/64KB	0.796555	0.806116	0.805535	0.833534
8192/128KB	0.840697	0.847871	0.846772	0.866453

Experiment Results on IXP1200 Enp-2505 board

To evaluate the effectiveness of our proposed cache design, we implemented three cache schemes including direct mapped cache, two-way FIFO cache and two-way LRU cache. However, we face another problem: all packets generated by IXP NetBench will be cached, and the cache hit ratio will be almost 100%. For solving this problem and creating different testing cases, we implement a software counter that will invalidate the cache block periodically. With this counter, we can create different cache hit ratio in our system.

The experimental environment is the same as Figure 2.4; we adopt our proposed classification algorithm, using the rule_1000 as our classifier. IXP NetBench generates the input traffic with minimum size of packets in this experiment. We find that the performance of these three cache schemes described above is almost the same. Therefore, we only show the results of two-way FIFO cache scheme in Fig. 3.3. Observably, with the two-way FIFO cache mechanism, our system performance can be improved significantly. In the case of 87.5% hat ratio, our system can achieve the IXP NetBench transmit speed (71.34Mbps), which has 50% improvement in throughput over the system with no cache (35.53Mbps).

4 CONCLUSIONS

In this paper, we first explained the needs of network processors for today's complex applications, and introduced the hardware architecture and development environment in IXP1200. Then we explain the software architecture and detailed pack flow in our system. At first, we implement the Grid-of-Tries as our classification algorithm. However, its heavy memory accesses almost take the 70% processing time when dealing with minimum size of packet. To increase the performance, we designed and implemented a new 5-dimensional packet classifier. Our classification algorithm needs less memory space than Grid-of-Tries and makes it possible to put in fast SRAM. In addition, the logarithmic search method of our proposed algorithm is very suitable and scalable for packet classification because the typical rule tables are not very big in real world. In benchmarking our proposed algorithm, our scheme for 1000 rules achieves throughput of 35.5 Mbps at minimal size of packets, which is much better than the results reported in [12].

Furthermore, we proposed a cache mechanism for IXP1200 because we observed that backbone routers have strong temporal locality. Our proposed cache scheme not only caches the result from packet classification but also caches the result from IP lookup. Only one SRAM-read is needed to perform IP address Lookups and packet classification procedure when cache hits. With this cache mechanism, the throughput of our system is very close to the theoretical maximum bandwidth with reasonable hit ratio.

REFERENCES

[1] F. Baker, "Requirements for IP Version 4 Routers," *Request for Comments - 1812*, Network Working Group, June 1995.

[2] F. Baboescu, S. Singh, and G. Varghese, "Packet Classification for Core Routers: Is there an alternativeto CAMs?," In *IEEE Infocom*, 2003.

[3] Yeim-Kuan Chang, "Fast Binary and Multiway Prefix Searches for Software-Based Routers", submitted for publication.

[4] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE/ACM Trans. Networking*, vol.15, pp.24-32, 2001.

[5] Intel Corporation, "Product Brief: IXP12EB Intel® IXP1200 Network Processor Ethernet Evaluation Kit," 2000.

[6] Intel Corporation, "Development Tools User's Guide", March 2002.

[7] IEEE. Standard 802.3, October 2000.

[8] Intel Corporation, "IXA SDK 2.01 Developer's Guide: Intel IXA SDK ACE Programming Framework," December 2001.

[9] Intel Corporation, "SDK 2.01 Reference: IXA SDK 2.01 Developer's Guide: Intel IXA SDK ACE Programming Framework," December 2001.
[10]Intel Corporation, "Reference Manual: Intel® microengine C Compiler Language Support," August 2001.
[11]Intel Corporation, "Reference Guide: Intel® microengine C Networking Library for the IXP1200 Network Processor," December 2001.
[12]Ying-Dar Lin, Yi-Neng Lin, Shun-Chin Yang and Yu-Sheng Lin, "DiffServ Edge Routers over Network Processor: Implementation and Evaluation," *IEEE Network*, August 2003.
[13]F. Baker, "Requirements for IP Version 4 Routers," *Request for Comments - 1812*, Network Working Group, June 1995.
[14]Hewlett Packard, "Netperf: A Network Performance Benchmark," http://www.netperf.org/netperf/NetperfPage.html, 1995.
[15]RadiSys Corporation, "ENP2505 Hardware Reference", 2002.
[16]M. A. Ruiz-Sanchez, E.W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," IEEE/ACM Trans. Networking, vol. 15, pp. 8–23, 2001.
[17]Quinn O. Sell, Armin. R. Mikler and John L. Gustafson, "NetPIPE-A Network Protocol Independent Performance Evaluator," http://www.scl.ameslab.gov/netpipe/, 2004.
[18]V. Srinivasan, G. Varghese, S. Suri, and M. Waldvagel, "Fast and scalable layer four switching," In Proc. ACM SIGCOMM'98, pp. 191–202, 1998.
[19]A. Tirumala, M. Gates, F. Qin, J. Dugan and J. Ferguson. "Iperf - The TCP/UDP band-width measurement tool," http://dast.nlanr.net/Projects/Iperf.
[20]D. E. Taylor and J. S. Turner, "ClassBench: A Packet Classification Benchmark," Tech. Rep. WUCSE-2004-28, Department of Computer Science & Engineering,Washington University in Saint Louis, May 2004.

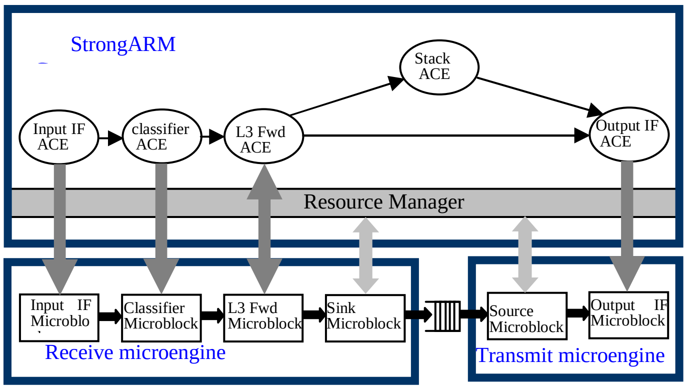


Fig.2.1 Software Architecture of IXP Router

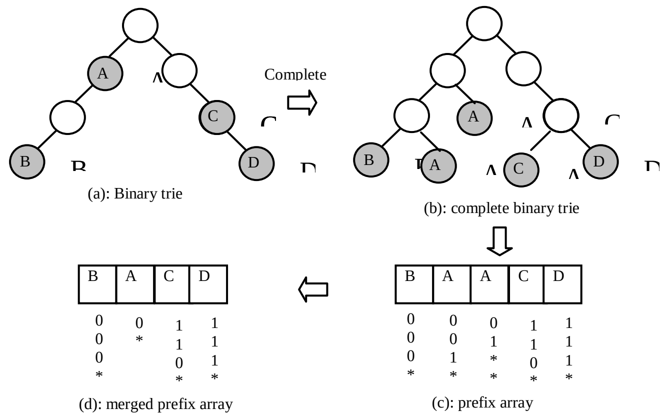


Fig.2.2 example of binary prefix search.

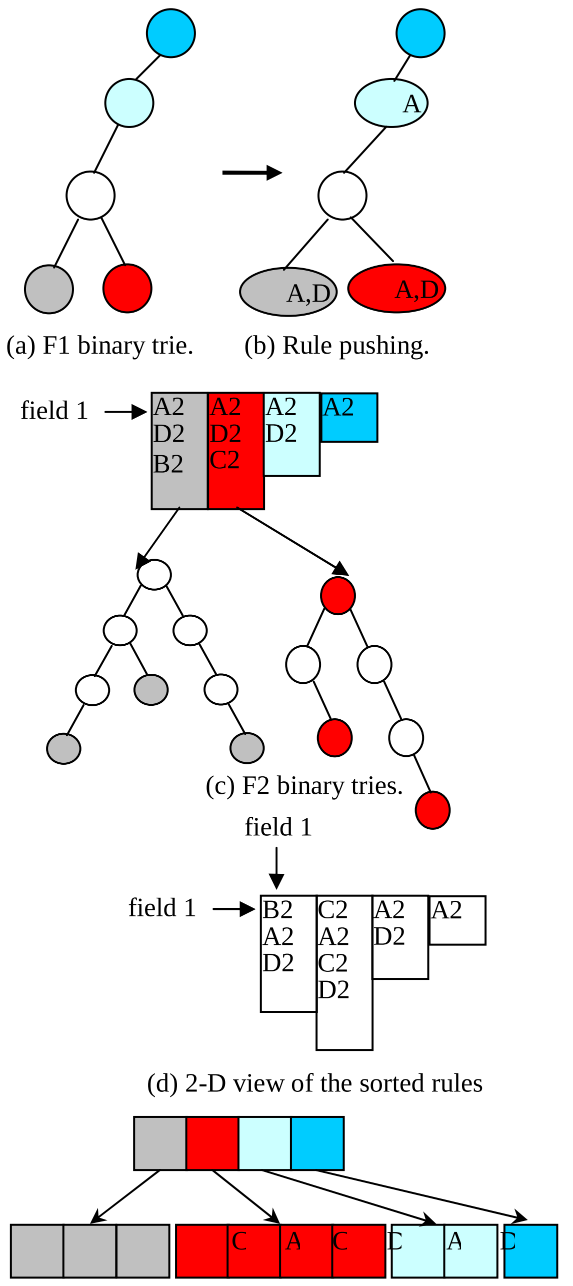


Figure 2.3: example of the multi-dimensional classification.

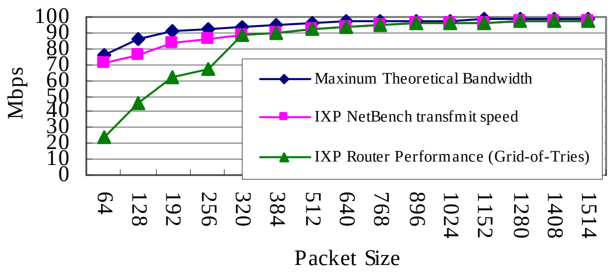


Figure 2.4: IXP Router Performance based on Grid-of-Tries

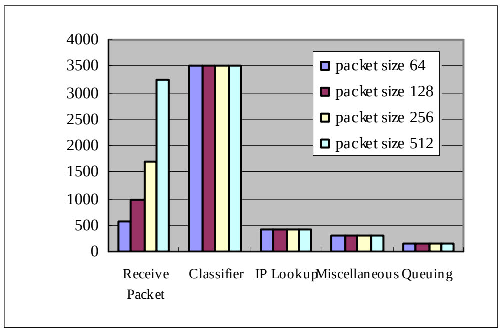


Figure 2.5: processing requirement (clock cycles) of these five categories with different size of packets; Simulation Configuration: Core Speed: 232 MHz; IX Bus Speed: 66 MHz

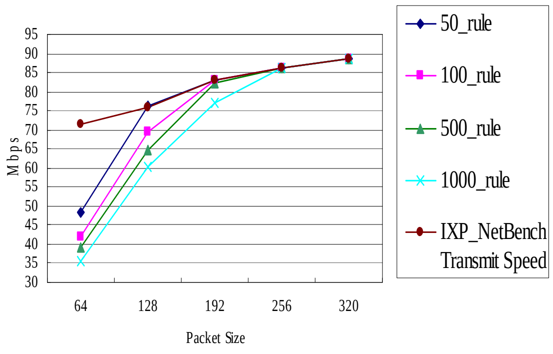


Figure 2.6: Throughput with varying number of rules

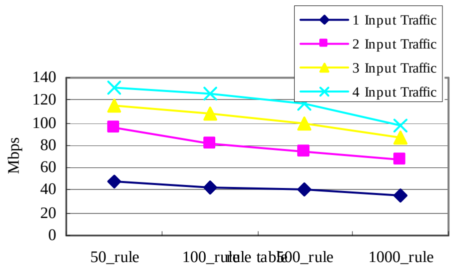
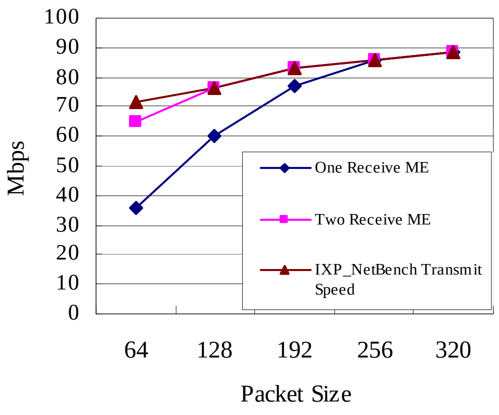
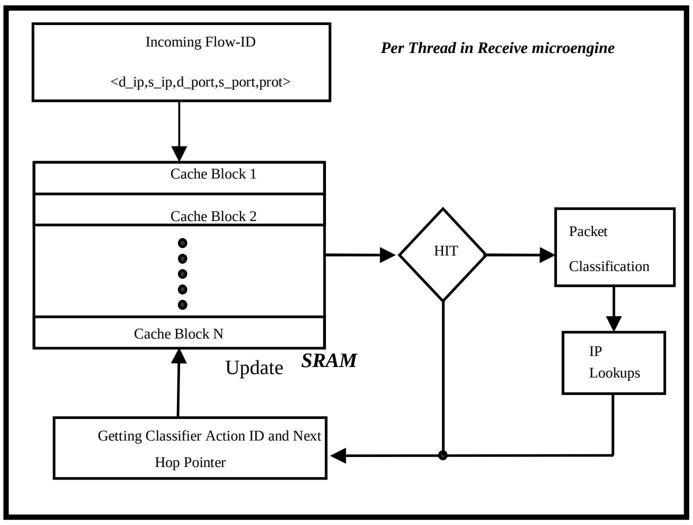


Figure 2.7: Total throughput (packet size=64byte, worst case)





Incoming Flow-ID
<d_ip,s_ip,d_port,s_port,prot>
Per Thread in Receive microengine
Cache Block 1
Cache Block 2
Cache Block N
HIT
Packet Classification
IP Lookups
Update
SRAM
Getting Classifier Action ID and Next Hop Pointer

Figure 3.1: Functional block of our Caching Approach for IXP1200.

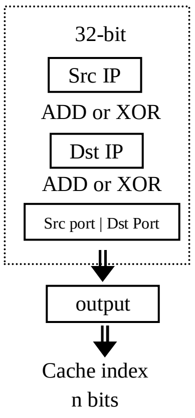


Figure 3.2: ADD or XOR based hash functionThe low order n bits of hash output are used to index the 2^n cache blocks.

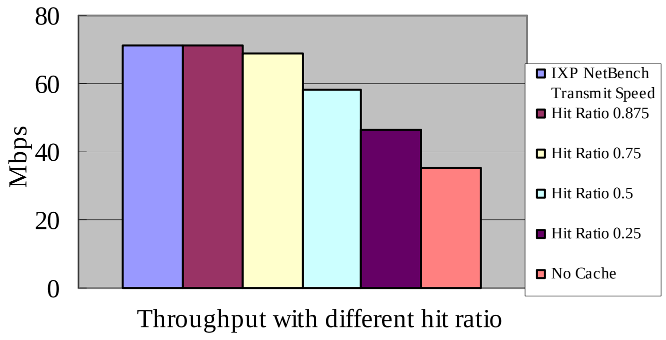


Figure 3.3: Throughput with different hit ratio where microengine 0 as Receive microengine; microengine 4 as Transmit microengine.