# HIGH PERFORMANCE CACHE ARCHITECTURES FOR IP ROUTING: REPLACEMENT, COMPACTION AND SAMPLING SCHEMES

By

RUIRUI GUO

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of RUIRUI GUO find it satisfactory and recommend that it be accepted.

____________________________________

Chair

____________________________________

____________________________________

## ACKNOWLEDGMENT

First of all, I would like to thank my advisor, Dr. José G. Delgado-Frias for his precious suggestions, grateful kindness and incredible patience. His continued support and guidance were crucial for the successful completion of my PhD study.

I gratefully acknowledge my committee member, Dr. Jabulani Nyathi and Dr. Krishnamoorthy Sivakumar, for all their help, advice and time. I thank all students in High Performance Computer Systems Research Group (HiPerCops) for their hard work. I truly appreciate the School of Electrical Engineering and Computer Science, for giving me a chance to study here and supporting me through my graduate study.

Last but not least, I would like to give special thanks to my family, for their endless love, encouragement and constant support.

HIGH PERFORMANCE CACHE ARCHITECTURES FOR IP ROUTING:

REPLACEMENT, COMPACTION AND SAMPLING SCHEMES

## Abstract

by Ruirui Guo, Ph.D.
Washington State University
August 2007

Chair: José G. Delgado-Frias

IP routing is an important operation in the forwarding of packets through the Internet. It decides how and where to deliver incoming packets to the appropriate output interface of a router. The process is performed by looking up IP addresses in a routing table stored in memory. The speed of this operation has a great influence on the overall performance of network processors. With the growth of Internet, the routing table lookups are required to be faster to match the increasing link bandwidth.

This dissertation presents novel cache-based schemes to obtain high routing table lookup performance. This study involves the following aspects. In regard to the cache architectures, a victim cache is implemented to store the entries discarded by the main cache. A randomly selected index (RSI) method is designed to redirect indexes away from those entries that have a large possibility to cause conflict misses. As for the cache replacement policy, two new policies that tend to remove an inactive entry by considering its previous access references are introduced and evaluated. In order to reduce memory size, novel route entry compaction schemes are designed based on the special features of

Ternary Content Addressable Memory. In addition, two improved sampling techniques are introduced to alleviate the port error which is a side-effect of caching. A set associative caching scheme specially implemented for the compaction schemes is also described.

These schemes are evaluated through extensive simulation based on IPv4 and IPv6 routing information. The results show our schemes can significantly enhance cache hit rate up to more than 20%. The higher hit rate makes average memory access time shorter, this in turn speed up the route lookups. Moreover, a small port error ratio is beneficial to reduce the possibility of incorrect routing.

TABLE OF CONTENTS

LIST OF TABLES

LISTOF FIGURES

# Chapter 1

# Introduction

## 1.1 Computer network

A computer network is formed by a number of computer systems, which are interconnected and can communicate and exchange information. Computer networks emerged in the late 1960s as a result of computing and communication technologies' development. Computer networks have grown at an increasing rate with the rapid growth of the Internet [1]. One use of computer networks is the sharing resources to make all programs, equipment, and especially data available to anyone that is connected to the network overcoming geographic barriers.

Communication networks usually have one of two approaches to achieve communication: circuit switched and packet switched network. Circuit switching is the method used by the telephone network, which operate by forming a dedicated connection between two points. It usually has three phases: establish circuit, communicate and close circuit. The important advantage of circuit switching is its guaranteed capacity. On the other hand packet switching is a method where data are contained in packets and transferred across a network. Packet header contains control information, such as source and destination addresses. At each connection node, the entire packet is received, processed, stored briefly and then forwarded to a specified destination. Consequently, different packets might follow different paths. Packets might be recorder, delayed or even dropped sometimes. The significant advantages of packet switching are the simply implementation and efficient bandwidth usage. That is, multiple data communications can be proceed concurrently. Packet switching is used in the Internet.  In this dissertation, all our research concentrates on packet switched networks.

In order to communicate, different computers must agree on both standards and protocols.  A protocol is a set of rules that governs how two parties are to interact with each other. As for the Internet, there is a suite of communication protocols. TCP/IP (Transmission Control Protocol/Internet Protocol) is the most commonly used protocol, which contains the details and standard for transmitting data over the lower layer of the Internet [2]. In addition, there are some protocols involving upper-layer applications, such as SMTP [3] for electronic mail, FTP [4] for file transfer etc.  More specifically for TCP/IP:

- IP contains addressing information and some control information, which is responsible for transfer packet of data from node to node. IP provides connectionless, best-effort delivery service. It forwards each packet based on a destination address.
- TCP provides reliable transmission of data. It is responsible for verifying the correct delivery of data from sender to receiver. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received. Moreover, TCP offers efficient flow control, full-duplex operation service.

## 1.2 IP addresses

An IP address (Internet protocol address) is a unique address to identify electronic devices on a computer network when utilizing the Internet Protocol. It is important for hosts to communicated with each other. In the packet switched Internet, the data is forwarded based on the destination address in the packet.  The followings are two kinds of IP address currently used.

### 1.2.1 IPv4 (Internet Protocol Version 4)

Our current Internet architecture is based on the version 4 of the Internet protocol [5]. IPv4 uses 32-bit IP network addresses, and it assigns at most $2^{32}$ or almost 4 billion unique addresses for those hosts connected on the Internet.  Each address encodes its network number and host number.

For several decades, the IP address space were divided into five categories listed in Figure 1.1. This allocation is called classful addressing [6]. Obviously, the several bits can distinguish the different classes.

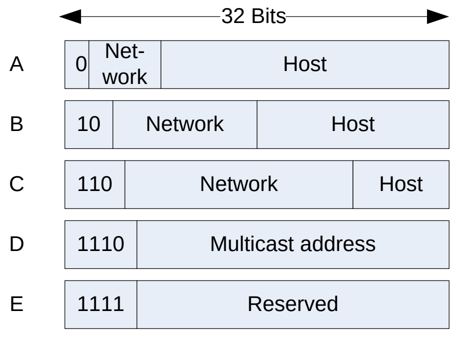


Figure 1.1: IPv4 address formats.

The IPv4 address is usually written in dotted decimal notation, which includes four decimal integers separated by decimal points to present 32-bit numbers. The lowest IP address is 0.0.0.0, and the highest is 255.255.255.255. For example,

11000000 01100101 00110010 00010100

This address is written as

192.101.50.20

However, this traditional classful addressing has its limitation. It does not allow the address space to be used to its maximum potential because of assigning blocks of addresses with strict boundaries. The lack of access to IP address' full penitential is becoming a serious problem as the Internet grows in number of unique devices in the network. It is predicted that the available addresses will be run out someday. In that situation, no more network device or user can be added to the Internet. This is the problem of IPv4 address exhaustion.

The CIDR (Classless InterDomain Routing) [7] is one solution to temporarily alleviate the shortage of address. The basic idea of CIDR is allocate IP addresses in variable-sized blocks by using a mask to delineate network and host part. For example, 192.168.0.0/21. Hence, instead of classful addressing, which restricts the bits of network address to 8, 16 or 24 bits, CIDR allows the range from 1 to 29 bits.

Another method to solve address shortage is Network address translation (NAT) [8]. This process enable a large number hosts on a private network share a small number of public IP address. When a packet transmits between the local networks to the Internet, the address on the packet is translated to a public address. Thus, Internet can contain far more hosts than its normal capacity, while it makes the end-to-end application difficult to be implemented.

### 1.2.2 IPv6 (Internet Protocol Version 6)

Although CIDR and NAT schemes are helpful to alleviate the IPv4 address exhaustion, they are considered as short-term solutions. They do not solve the problem completely. A new protocol with a larger address space has been proposed and started to use, IPv6.

IPv6, Internet Protocol version 6, is the next generation protocol designed to replace current IPv4 [9]. It is also a data-oriented network layer protocol to transmit data across a packet-switched network, similar to its predecessor. The major change from IPv4 to IPv6 is the length of network addresses. IPv6 enlarged address format form 32 to 128-bit, which has the potential to have up to $2^{128}$ addresses. At present time, only a small portion of these addresses have been assigned to devices, and many addresses are available for future use. There is no need to have NAT any more due to the large address capacity of IPv6.

There are three types of IPv6 addresses: Unicast, Anycast, and Multicast address [10]. Unicast address is used to identify one single interface, while other two are used for a set of interfaces. There are two notations for 128-bit IPv6 addresses. One is writing as eight 16-bit integers separated by colons. Each integer is represented by four hexadecimal digits. For example,

FEDC:BA98:7654:3210:FEDC:BA98:7654:3210

Another notation is used to represent address prefixes. It is similar to the CIDR notation for IPv4 addresses. For example,

12AB:0000:0000:CD30:0000:0000:0000:0000/60

And this address is also compacted as 12AB:0:0:CD30:: / 60. The first part is a valid IPv6 address, and the decimal value tells us its prefix length.

IPv6 has additional features. The header format of IPv6 packet is simpler than that of IPv4. It is only composed of six fields, followed by two 128-bit IPv6 source and destination addresses. The total length of header is 40 bytes, while IPv4 has 10 fixed header fields, two addresses and some options. The length of its header is variable. IPv6 is relatively easy to manage because of its hierarchical addresses. IPv6 also provides fundamental support for security and quality-of-service (QoS).

## 1.3 Internet routing and address lookups

IP routing plays an important role in the forwarding of packets through the Internet. It decides how and where to deliver incoming packets to the appropriate output interface of a router using the packet’s destination address. Usually, this process involves two basic activities: determining optimal routing paths and transporting packets through the Internet. The first part is more relative to our research topics, that is, determining which output ports that the received packets should be forwarded to. This process is completed inside a router [11].

### 1.3.1 Router architecture

Router is a network device that physically connects similar or different networks. It makes decision about where and how to deliver an incoming packet to the appropriate output interface by using its destination address. In this dissertation, the network router architecture has the form shown in Figure 1.2 where there is a set of input ports that receive packets from other nodes in the network, a switching fabric that forward these packets to the proper out port, and a set of output ports that deliver the packet to the following hop.

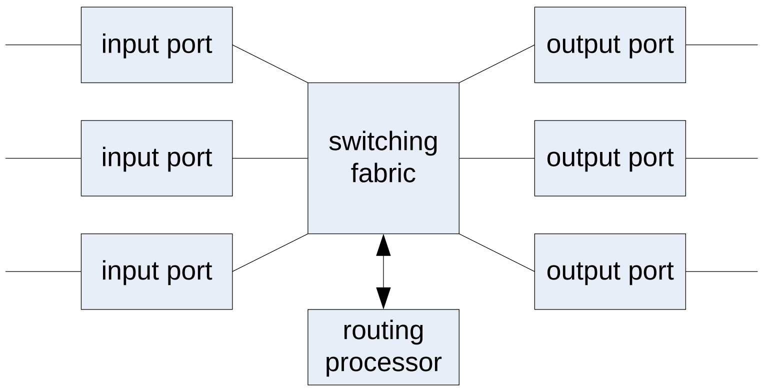


Figure 1.2: General router architecture.

### 1.3.2 Basic routing process

The process is performed by looking up entries in a routing table, which contains information relating to other networks and hosts in the Internet. Routing table is initialized and updated to determine path by variable routing protocols, such as RIP[12], OSPF[13] etc. Each entry in the routing table comprises an address prefix, a forwarding address, and the interface to which the packets should be delivered when their address prefix matches [14, 15]. The router makes decision based on route entries in the routing table, which is maintained in memory storage.

The routing process works as follows:

- The router receives a packet from one of router's interface
- It extract the destination address in the packet, and then check it in routing table to see if there is a match to forward this packet.
    - If there is a match that informs us the output port to the subnet directly attached, then send this packet to the destination host by lookup ARP for a MAC address.
    - If the router determines the destination network is not local, then deliver the packet to next hop router.
    - If there is no directly match in the routing table, deliver the packet to a default port. If there is no such port, it sends an error back to source.

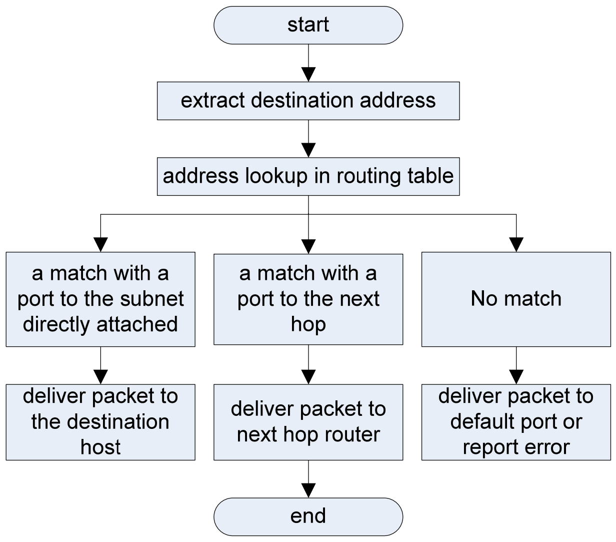
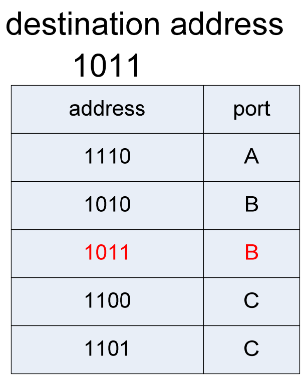

start
extract destination address
address lookup in routing table
a match with a port to the subnet directly attached
a match with a port to the next hop
No match
deliver packet to the destination host
deliver packet to next hop router
deliver packet to default port or report error
end

Figure 1.3: IP routing process.

### 1.3.3 Route lookup schemes

Typically, there are two lookup mechanisms: exact match searching and longest prefix matching searching (LPM).

Exact match is very direct. Each entry in the routing table is of fixed length. By comparing the whole destination address of the arriving packet to the entries in routing table, we can get the port simply. This method can be realized by direct lookup, or hashing techniques. For example, 4-bit address 1011 is found an exact match in the following routing table and it will be delivered to port B directly in Figure1.4.



destination address
1011

| address | port |
|---|---|
| 1110 | A |
| 1010 | B |
| 1011 | B |
| 1100 | C |
| 1101 | C |

Figure 1.4: An example of exact matching.

LPM is a method to make a delivering decision by the address and its prefix. When a packet arrives, IP router finds entries matching with the incoming packet’s destination address and selects the entry with the longest prefix. Then forward the packet to the output port that the selected entry provides. Since the destination address of an arriving packet does not carry the prefix length information, routers need to search among the space of all prefix lengths as well as the space of all prefixes of a given length. Hence the longest prefix matching is more complicated to implement than the exact matching. LPM can be realized by PATRICIA trie, or TCAM [15].

## 1.4 Challenges

With the continuous growth of the Internet, higher demands are placed on the IP routing in terms of speed; in particular, the growth of link bandwidth requires increasingly fast IP routing table lookups. A router needs to handle roughly 1,000 packets per second for every $10^6$ bits per second of line rate [17]. Therefore, 10M routing lookups per second are needed for a current route with the line rate of 10 Gb/s (OC-192). Moreover, in the near future, the line speed will grow towards 40 Gb/s (OC-768) with the continued technological advances in optical and electronic devices [16]. Such a high line speed requires fast lookups to match.

In addition, routing tables are becoming larger with the development of Internet; the implementation of IPv6 needs more memory storage because of its long address format. Thus a large memory is usually required to store such tables in a local router. This in turn may restrict the lookup speed since the complete routing table is stored in main memory with slow access time. Memory access time becomes a long-term bottleneck. It slows down the address lookup operation.

With regards to the above challenges both in speed and space for IP routing, the purpose of our research is to provide high-performance routing schemes and technologies without cost overhead.

## 1.5 Outline

The remainder of this dissertation is organized as follows. Chapter 2 presents three improved cache schemes and their implementation of pipeline. We estimate their performance both from cache hit rate and throughput speedup. Chapter 3 discusses the impact of cache replacement policies; provides two optimized policies which make replacement decision based on route's history activities. Chapter 4 proposes four compaction schemes for routing table entries by taking advantage of the special characters of Ternary CAM. We analyze their cache performance and memory consumption.   Chapter 5 deals with port error occurrence caused by caching. This chapter describes two novel sampling techniques to alleviate this problem. Chapter 6 describes new set associative caching with regard to the compaction schemes. We develop this scheme based on the practical routing table address space and historical cache hit distribution. Chapter 7 provides some concluding remarks which include a summary of the contribution of this research work.

# Chapter 2

# Cache Architectures for IPv6 Routing and Address Lookups

## 2.1 Introduction

With the large Internet growth, the speed of IP route lookups is becoming an important issue that has a great influence on the overall performance of network processors. One of the significant factors is the slow access time of main memory, where the routing tables are usually saved. Since small storage has the advantage of high memory access speed and low power consumption, caching technique can be used in network processor to solve this problem. The method is storing the most frequently used destination addresses and their forwarding information in cache to reduce the average memory access time.  Using a cache scheme one has to take into account the impact of cache size, replacement policies, and associativity on cache hit rate.

In this Chapter, we present several novel cache schemes and implement their pipelining architectures to enhance the performance of route lookups. These schemes are all improvements based on common cache architecture. One scheme uses a victim cache (VC) to save the entries discarded by the main cache. Another uses randomly selected index (RSI) to redirect indexes away from those entries that have a large potential of having conflict misses. The last scheme uses a combination of these two above schemes to obtain more enhancements. Moreover, using pipelining, the system throughput is also greatly increased. Our studies here focus on the next generation IP address, IPv6, on which the memory access time is crucial because of its long address format (128 bits). All these schemes are realized and evaluated through extensive IPv6 traces. The simulations later validate that these schemes help to reduce conflict misses effectively.

This chapter is organized as follows. In Section 2.2, an introduction about route lookups and related works are provided. Section 2.3 describes that victim cache architecture, randomly selected index method; the combinational scheme of both to improve routing performance. The implementation of pipelining is presented in Section 2.4 to enhance the system throughput. In Section 2.5, we present the simulation results using IPv6 routing information. Some conclusion remarks are provided in Section 2.6.

## 2.2 Route lookup fundamentals

In this section, we begin with a brief description of route lookup approaches by other researches, and then provide the conventional caching technique for routing operations, which is particular important for our improved caching schemes.

### 2.2.1 Route lookup approaches

There have been a number of techniques proposed to increase the performance of lookups in Internet processor, roughly, which can be divided two major approaches.

One approach is based on hardware solutions [17, 18, and 19]; this is the emphasis of our proposed schemes. DIR-24-8-Basic scheme was proposed by Gupta [17].

It is implemented for IPv4 routing. It includes two tables, TBL24 and TBLlong, to store all possible route prefixes that are less than 24 bits and greater than 24 bits, respectively. The lookup method is simple and efficient for IPv4, because more than 99% of prefix in routing table have 24 bits or less. Huang proposed another lookup mechanism later but based on SRAM, which simplified hardware design [18]. However, both schemes depend on the prefix distribution and are difficult to scale to IPv6, because of its long address of 128 bits. This requires a lot of memory to store all possible prefixes for IPv6.

Ternary content addressable memory (TCAM) is able to match in parallel stored data with incoming data [22, 23, 15]. The major advantage of this memory is the o(1) search time; i.e. only one memory access. However, it requires an extra priority encoder to deal with the problem that several matches occur at the same time. This increases the complexity in hardware. And for IPv6 with long length of 128 bits, there will be a challenge to store long length entries in TCAM because of its high price and power consumption.

Another kind of approaches is software method in which efficient lookup algorithms are used to find the longest prefix matching of a destination address, which is stored in a table data structure [20, 21]. These algorithms are designed to produce short searching path or reduce the memory storage requirements.

### 2.2.2 Cache architecture for IP routing

Cache memories are widely used in computer systems. In a program (with many memory accesses) it is very likely that the same data in memory is accessed multiple times in a short time period; this is called temporal locality. Thus, a small and fast memory can be used to reduce lengthy memory access times. Studies have shown that the network packet streams indeed have temporal routing locality. That is, a routing entry accessed before it is possibly referenced again in a short period of time. This feature allows caches to be used in IP address lookups. Consequently, more active forwarding entries are saved in cache; this in turn has the potential of making lookup faster.

The process of lookups is associated with a destination address. When a router receives a packet from one of its interfaces, the destination address in this packet is extracted, and compared with current entries in the cache using a Longest Prefix Matching (LPM) mechanism. The basic organization of the cache is depicted in Figure 2.1. If there is a matching entry in cache, this packet will be delivered to the interface specified by this entry. If a miss occurs, an address search is made in the large routing table stored in memory. Subsequently, the matching entry is written back into cache. Here, we use Least Recently Used (LRU) replacement policy, in which the route entry that has not been accessed for the longest time will be evicted from the cache.

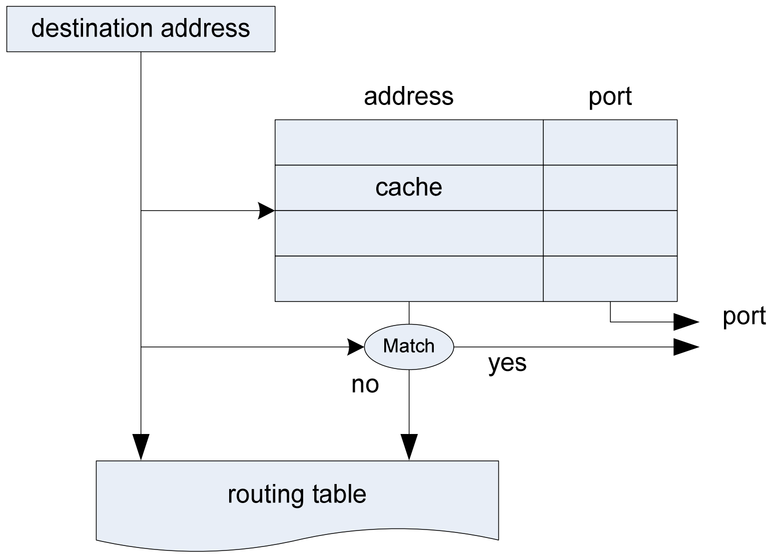


Figure 2.1: A cache organization for IP address lookup.

The average memory access time (AMAT) of the cache organization is determined as follows [24].

$$AMAT = (H_{cache} \times AT_{cache}) + ((1 - H_{cache}) \times AT_{memory}) \quad \text{(Equation 2.1)}$$

where $H_{cache}$ is the cache hit rate, $AT_{cache}$ is the cache access time, and $AT_{memory}$ is the routing table access time.

Current technology and memory implementations indicate that cache (using SRAM) has an access time that is 8 to 16 times faster than main memory (using DRAM) [24]. It is clear that a cache technique is needed to obtain small average access time. A small size cache usually yields low access time ($AT_{cache}$) while temporal locality helps to increase hit rate ($H_{cache}$). This chapter deals with different cache architectures to increase hit rate without hardware implementation overhead.

## 2.3 Proposed cache architectures

Similar to a generic CPU cache memory, the data searched in routing operations are associated with incoming destination addresses. Although cache misses due to conflict can be reduced by using higher associative cache [24], such cache also tends to increase hardware complexity and affects performance [25]. Therefore, direct-mapped (DM) cache is usually used in order to keep design simple. In this section we present three schemes that are based on DM cache and need no special memory devices, but can achieve effective cache performance.

### 2.3.1 Victim cache architecture

Victim cache (VC) was first proposed by Jouppi [26] to reduce misses without affecting the hit time or the miss penalty. It is a small, fully associative cache that stores those entries discarded by the active (or primary) cache. If those entries are needed later, they will be retrieved from VC. Thus, there will be no need to access the routing table.

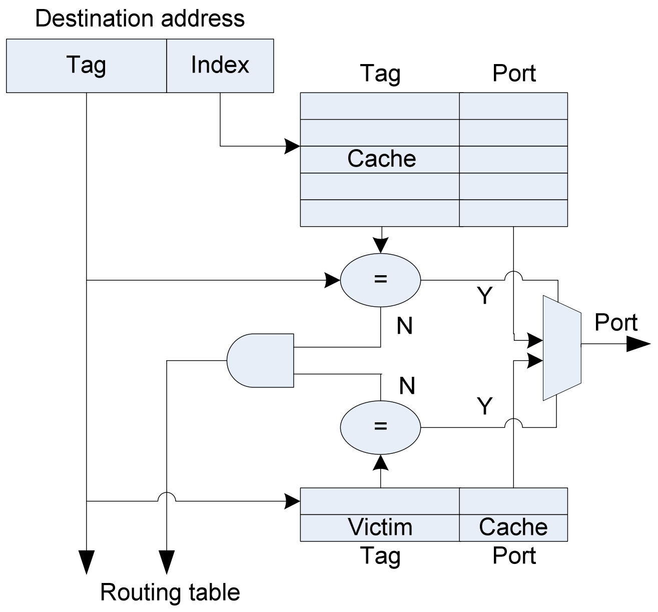


Figure 2.2: Victim cache architecture.

Victim cache architecture for our application is shown in Figure 2.2. Destination address is first sent to both caches. The cache is indexed by some bits of the address, while the whole address is compared in the VC. There are three possible outcomes, when both caches perform a lookup. These are:

*Cache Hit*. There is a hit on main cache; thus, the forwarding port can be retrieved from this cache.

*Victim Cache Hit.* There is hit on victim cache and not hit on cache. The forwarding port can be retrieved from this victim cache. A swap between this matching entry in the victim cache and its corresponding entry in the main cache is performed.

*Cache System Miss.* Both main cache and VC have a miss. Using a replacement policy, the new entry from the routing table is placed into the main cache. The discarded entry is placed into VC. We use least-recently used algorithm (LRU) as replacement policy for VC, i.e. replaced entry is the one that has been unused for the longest time.

### 2.3.2 Randomly selected index architecture

If we use direct-mapped cache, conflict problems affect hit rate negatively. In our simulations, we observed that there are some entries that are prone to conflict problems. Hence, we propose the randomly selected index scheme to reduce conflict misses for DM cache.

Based on statistics of the behavior of the cache conflict misses over a period of time, we can adjust the mapping of the cache. Conflict misses occurrence is due to mapping of entries to the same location in cache. We label the index of the entry that prone to conflict misses as a predictor and use it to predict misses in the next period. In next period, if one index from a destination address is accessed to the set that is the same as predictor, then we use randomly selected mechanism to choose another set indexed uniformly. This method avoid many conflict misses that might occur in the set have most misses before, and disturbed them to some other sets with less accessed or less misses. This technique has a simple implementation and can enhance the performance of cache by reducing cache misses due to conflict.

In Figure 2.3, the Index Selections (IS) of the scheme is shown. For a direct-mapped cache with the size of 512 entries, each index needs to be 9 bits. In the case that the original index of a destination address matches with the predictor, another index bits are chosen randomly from a range in the address with the priority from top 1 (when there is no match with the entry address) down to 4. Once these bits are selected they are kept fixed for this period of the simulation. In this figure, input 1 is the original index of 9 bits, and input 2, 3 and 4 are new indexes of 9 bits that randomly selected from the following bit positions of the IPv6 address 89-118, 30-88, and 0-29, respectively.

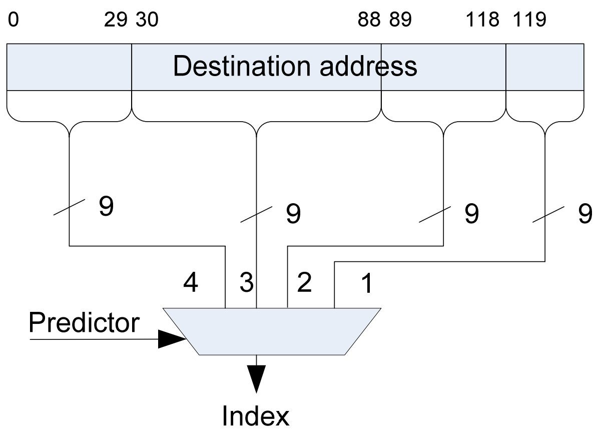


Figure 2.3: Index selection scheme.

### 2.3.3 Combination of the above two methods

Obviously, there are still some conflict misses even with the implementation of the above two methods. Due to the independence of these methods, we can use a combination of them to achieve further improvements. Using both victim cache and randomly selected index, cache hit rate can be increased.

## 2.4 Pipelining implementation in IP routing

In order to keep a high throughput for the above mechanisms, we propose pipeline structures. Pipelining is an implementation technique whereby multiple instructions are overlapped in execution [24]. The proposed pipeline is divided in three stages. Since these stages are independent, the pipeline has no data hazards. Figure 2.4 shows the pipeline with index selection, which has the following stages.

*Stage 1.* *Index Selections (IS).* Select different index to reduce conflict miss, if needed.

*Stage 2.* *Cache Access (CA).* Access cache (this includes VC access)

*Stage 3.* *Get Port (PT).* If a hit occurs in cache, the forwarding information is retrieved directly; otherwise, the address is sent to routing table to find the port.

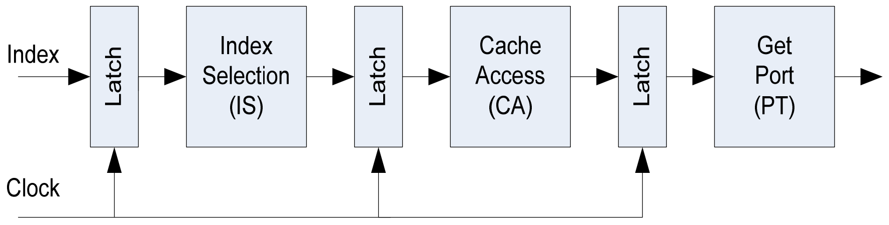


Figure 2.4: Pipeline for index selection cache architecture.

The length of the machine cycle is determined by the time required for the slowest pipe stage. Hence, the balance of the length of each stage should be considered in pipeline. If the stages are perfectly balanced, then the speedup from pipelining equals the number of pipe stages [27], however, the stages usually do not be perfectly balanced. For each lookup in cache architecture, if a hit occurs in cache, the forwarding information is retrieved at once. If there is a miss in cache, we need more time to get results. Usually, the time to get data from memory is called penalty [24]. The number of cycles in the penalty depends on the memory type.

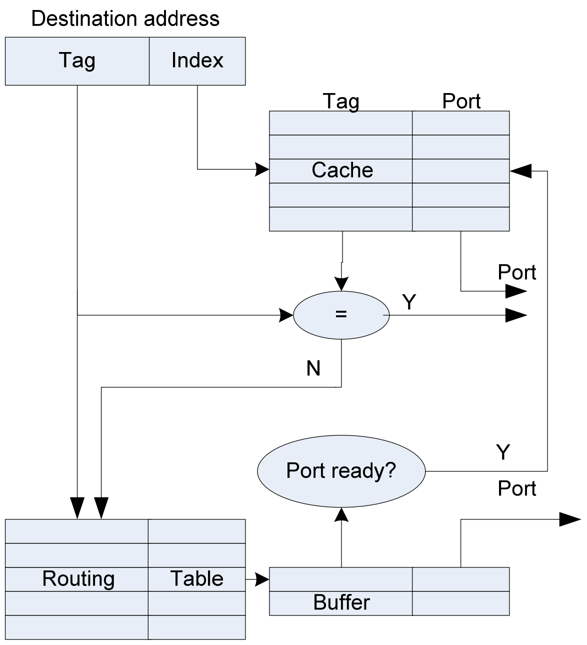


Figure 2.5: Routing table and cache Architecture.

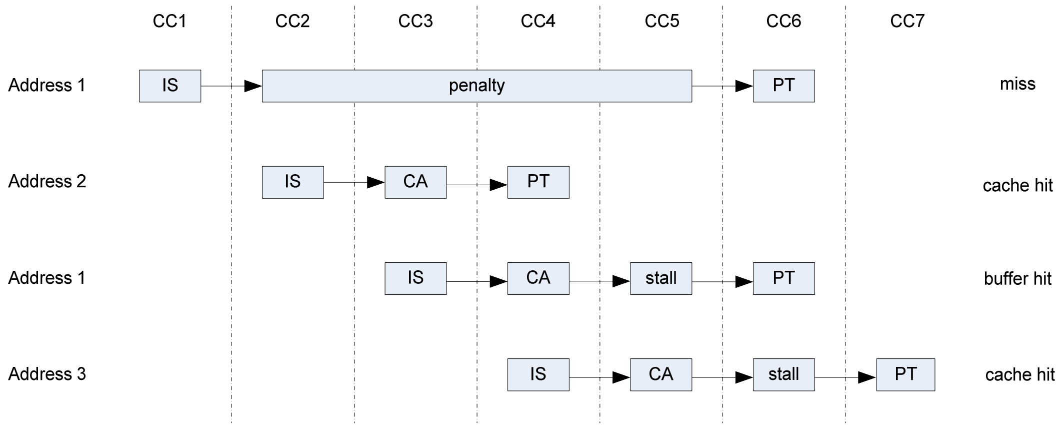


Figure 2.6: An example of route lookup operations in pipelining.

Figure 2.5 shows an implementation of the proposed scheme that shows the routing table. A buffer is used to store the entries that are being searched at the routing table when a miss occurs. There are three possible results when compare one destination address with the entries in cache, routing table or buffer. If a match in cache, it is a hit. If no match in cache and buffer, it is a miss. If no match in cache and a match in buffer, this is considered a pseudo-miss. The miss penalty is less than those of a real miss since the entry is being searched in the routing table. Figure 2.6 shows an example of route lookup operations in pipelining.

### 2.5 Performance analysis and evaluation

Cache hit rate is one of major measurements of performance for a cache memory architecture. It also provides a good indication of the potential gain in performance. Because main memory access time is much longer than cache access time, many searches in memory will slow down the lookup process. However, with a higher hit rate, fewer routing table memory accesses are needed. In this section we evaluate the performance of the proposed schemes and their pipelined implementation.

#### 2.5.1 Routing information

Evaluations of our proposed caching schemes are based on six IPv6 trace files which are downloaded from the Measurement and Analysis on the WIDE Internet (MAWI) Working Group http://tracer.csl.sony.co.jp/mawi/ [28]. Each trace file has 2 million destination addresses. The length of each IPv6 address is 128 bits, which is four times longer than the current network address IPv4. Analyzing these destination addresses in each trace file, we found that there are only several thousands unique addresses. This in turn indicates there exists temporal locality in these trace files and it supply the possibility of using a cache architecture. The ratio of number of packets to number of unique address is shown in Table 2.1. The higher ratio indicates higher locality.

Table 2.1: IPv6 address traces (2M packets/trace).

| Trace ID | Number of unique IP address | Ratio |
|---|---|---|
| 030715 | 1022 | 1957 |
| 031015 | 1539 | 1299 |
| 031213 | 3164 | 632 |
| 031214 | 3949 | 506 |
| 031215 | 3681 | 543 |

### 2.5.2 DM and set associative cache performance

Table 2.2 shows hit rate of DM cache with different cache size. Figure 2.7 shows this information in a graphical form. From this graph it can be observed that as cache size is increased the hit rate improves. The improvement rate diminishes as memory size grows. The curves are almost flat after the point of 1024 entries.

Table 2.2: Hit rate of direct-mapped cache with different sizes.

| Trace ID | Cache size (number of entries) | | | | | |
|---|---|---|---|---|---|---|
| | 64 | 128 | 256 | 512 | 1024 | 2048 |
| 030715 | 95.08% | 95.94% | 96.69% | 96.98% | 97.16% | 97.19% |
| 031015 | 88.21% | 90.11% | 93.74% | 94.45% | 94.64% | 94.84% |
| 031213 | 76.92% | 80.96% | 84.73% | 86.68% | 88.69% | 89.40% |
| 031214 | 52.65% | 61.55% | 69.48% | 73.26% | 76.57% | 78.05% |
| 031215 | 60.33% | 67.18% | 73.72% | 77.07% | 79.59% | 80.82% |

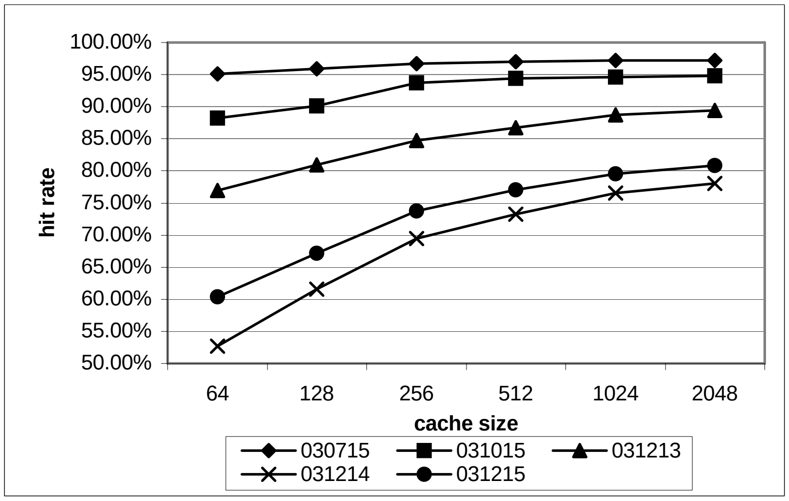


Figure 2.7: Plot of hit rate of DM cache with different size.

By increasing cache associativity, a better hit rate is obtained. Table 2.3 shows the 512-entry cache hit rate for each our traces under different cache associativities including direct-mapped, 2-, 4-, 8-way and fully associative. The data in Table 2.3 has been plotted and shown in Figure 2.8. It should be pointed out that, in general, higher associativity increases the complexity of hardware and hit time [24].

Table 2.3: Hit rate of cache with different associativity.

| Trace ID | DM | 2-way | 4-way | 8-way | Associative |
|---|---|---|---|---|---|
| 030715 | 96.98% | 98.28% | 99.27% | 99.64% | 99.91% |
| 031015 | 94.45% | 96.94% | 98.20% | 99.28% | 99.81% |
| 031213 | 86.68% | 91.35% | 94.54% | 96.71% | 97.68% |
| 031214 | 73.26% | 83.57% | 90.35% | 93.98% | 96.52% |
| 031215 | 77.07% | 85.31% | 91.43% | 94.74% | 96.81% |

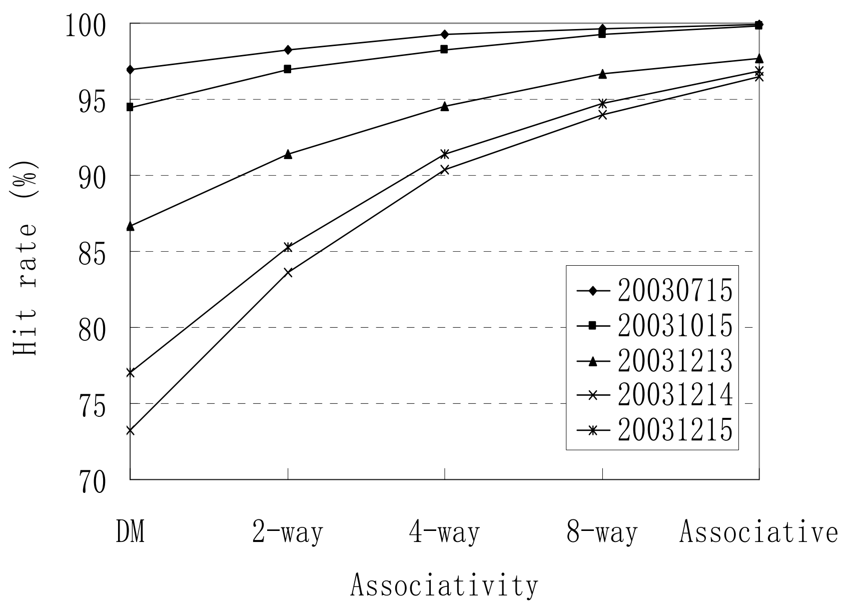


Figure 2.8: Plot of cache hit rate with different associativity.

### 2.5.3 Victim cache (VC) architecture performance

For the victim cache, we have chosen the number of entries to be 4, 8 and 16 and compare the performance with the cache without VC. Table 2.4 shows the hit rate of victim cache associated with a 512-entry primary cache.

Table 2.4: Hit rate of VC architecture.

| Trace ID | Hit rate increments of victim cache (VC) | | |
|---|---|---|---|
| | 4-entry | 8-entry | 16-entry |
| 030715 | 98.66% | 99.06% | 99.28% |
| 031015 | 97.38% | 98.07% | 98.47% |
| 031213 | 90.10% | 91.81% | 93.25% |
| 031214 | 80.84% | 84.31% | 87.25% |
| 031215 | 83.92% | 87.01% | 89.41% |

Comparing VC and DM (Table 2.3) we observe the following:

- The range of improvement goes from 1.68% to 13.99% when comparing with a simple DM cache.
- DM cache cooperating with a 16-entry victim cache can achieve a hit rate better than 2-way or 4-way set associative cache of the same size.
- For all of these traces, the hit rate of cache 256 with 16-entry victim cache is better than direct-mapped cache and close to 2-way associative cache with the size of 512 entries. This seems to reduce 48.43% the memory (storage) requirements. Similarly, 512-entry cache with 16-entry victim cache has a better hit rate than direct-mapped cache with the size of 1024 entries, or even better than 2048 entries; and close to 2-way associative cache with the size of 1024 entries(see Figure 2.9).

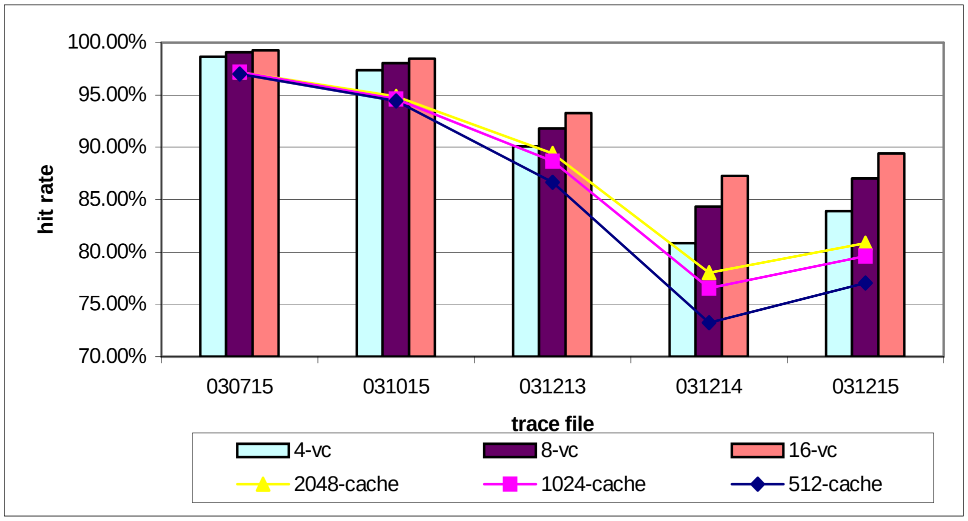


Figure 2.9: Compare between VC and cache.

### 2.5.4 Randomly selected index (RSI) performance

We have chosen two traces (031214 and 031215) to evaluate our prediction scheme. The prediction is based on an earlier trace 031213. It should be pointed out that these traces were obtained in three consecutive times. Every time the prediction is changed the cache has to be flushed. Table 2.5 shows there are hit rate increments of these two traces with RSI ranging from 9.98% to 11.29% over direct-mapped cache with cache size. The RSI performance is between 2-way and 4-way set associative with the same cache size. Although there is requires a stage to process index selection in this scheme, it also decreases miss rate without any damage on processing time if implement it in pipelining.

Table 2.5: The performance of RSI.

| Trace ID | Schemes | Cache size | | |
|---|---|---|---|---|
| | | 256 | 512 | 1024 |
| 031214 | DM | 69.48% | 73.26% | 76.57% |
| | RSI | 79.91% | 84.55% | 86.55% |
| 031215 | DM | 73.72% | 77.07% | 79.59% |
| | RSI | 84.05% | 87.52% | 89.69% |

### 2.5.5 Combination of VC and RSI performance

Table 2.6 shows an example of 512-entry main cache with different implementations. Obviously, the VC and RSI have the similar performance for the following two trace files. Moreover, if we combine the VC and RSI schemes together, the increments in hit rate are on the order of 2~3%.

Table 2.6: Performance of three schemes.

| Trace ID | DM | w/ 8-entry VC | RSI | Combination |
|---|---|---|---|---|
| 031214 | 73.26% | 84.31% | 84.55% | 87.24% |
| 031215 | 77.07% | 87.01% | 87.52% | 89.92% |

### 2.5.6 Pipelining technique

As we mentioned before, the pipelining technique is used to overlap the execution stages of one operation. It does not decrease the time for each lookup, but it increases the throughput. Hence, we choose the number of clock cycles needed to process 2M destination address traces as one measure of the performance of the pipeline besides a measure of hit rate.

Table 2.7: Hit rate of cache with pipeline and penalties (trace 031215).

| Penalty | Cache size | | | | |
|---|---|---|---|---|---|
| | 256 | 512 | 1024 | 2048 | Maximum |
| 0 | 91.09% | 96.81% | 99.44% | 99.77% | 99.82% |
| 1 | 88.72% | 95.84% | 99.36% | 99.74% | 99.79% |
| 2 | 88.22% | 95.67% | 99.32% | 99.72% | 99.77% |
| 3 | 87.79% | 95.54% | 99.29% | 99.71% | 99.76% |
| 4 | 87.48% | 95.45% | 99.27% | 99.69% | 99.75% |

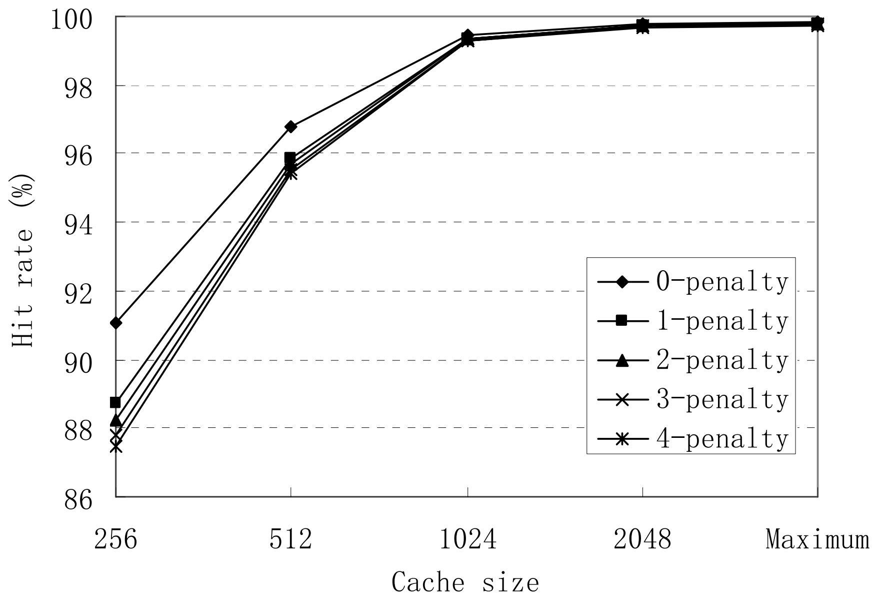


Figure 2.10: Plot of hit rate of cache with pipeline and penalties.

Table 2.7 shows the change of hit rate in pipeline. We choose trace 031215 as a sample. We use a buffer to store the entries in waiting states in pipelining. It is assumed that the cache could continue receiving requests while the routing table is accessed (in a miss case). This in turn has an effect on cache hit rate due to locality (such as two consecutive requests to the same address). Figure 2.10 shows that as the cache size increases the penalty has less impact on cache hit rate.

Table 2.8: Clock cycles of fully associative cache with pipeline and penalties

| Penalty | Cache size | | | | |
|---|---|---|---|---|---|
| | 256 | 512 | 1024 | 2048 | Maximum |
| 0 | 2000001 | 2000001 | 2000001 | 2000001 | 2000001 |
| 1 | 2113174 | 2041026 | 2009254 | 2003670 | 2002982 |
| 2 | 2140794 | 2055980 | 2010129 | 2004050 | 2003299 |
| 3 | 2143183 | 2056746 | 2010270 | 2004114 | 2003350 |
| 4 | 2147131 | 2058001 | 2010524 | 2004213 | 2003426 |

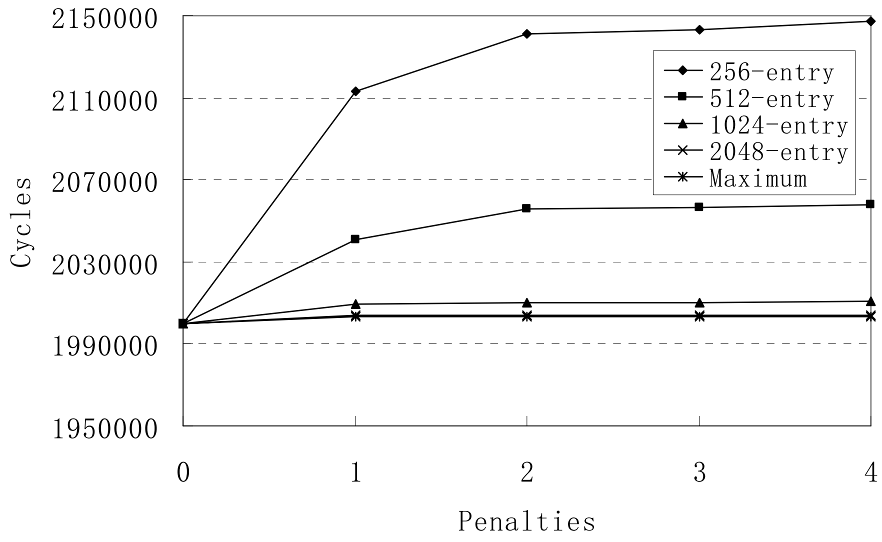


Figure 2.11: Plot of clock cycles of fully associative cache in pipeline.

Table 2.8 and Figure 2.11 include the clock cycles information to finish the lookup of trace 031215. Increasing the cache size leads to an increase on hit rate and decreases on conflict misses and the stalls used to solve structure hazards. Hence, the number of total cycles decreases and the average cycle per address also goes down. As penalty increases the number of cycles increase. Compared with those of scheme without pipeline, the speedup ranges from 1.89 to 1.99 depending on the cache size and the miss penalties.

The following tables have information about the performance of direct-mapped cache, fully associative cache, victim cache architecture, and randomly selected index architecture with pipeline. Taking trace 031215 with 512-entry cache as an example, the VC and RSI still have about 10% increment over DM cache. All cache architecture's hit rate is decreasing because of the increasing penalty; see Table 2.9. The number of clock cycles needed is shown in Tables 2.10. The throughput speedup is ranging form 1.75 to 2.99 depending on pipeline stage and time penalty as compared with those without pipeline.

Table 2.9: Hit rates of pipelining architectures.

| Penalty | Cache organizations | | | |
|---|---|---|---|---|
| | Fully | VC | RSI | DM |
| 0 | 96.81% | 89.41% | 87.18% | 77.07% |
| 1 | 95.84% | 87.11% | 84.51% | 73.60% |
| 2 | 95.67% | 86.50% | 83.68% | 72.36% |
| 3 | 95.54% | 85.96% | 82.93% | 71.04% |
| 4 | 95.45% | 85.51% | 82.22% | 69.73% |

Table 2.10: Clock cycles of pipelining architectures.

| Penalty | Architectures | | | |
|---|---|---|---|---|
| | Fully | VC | RSI | DM |
| 0 | 2000001 | 2000001 | 2000002 | 2000001 |
| 1 | 2041026 | 2143138 | 2168880 | 2260315 |
| 2 | 2055980 | 2166798 | 2192301 | 2273450 |
| 3 | 2056746 | 2168328 | 2192876 | 2262312 |
| 4 | 2058001 | 2170727 | 2193457 | 2243828 |

In pipeline, another factor we consider is the ratio of the number of pseudo misses to the number of real misses. This is shown in Table 2.11. If the ratio is high, like the penalty 3 and 4 of DM cache, clock cycles do not increase as fast as for lower penalties. The reason is that there are relative more pseudo-misses.

Table 2.11: Ratio of pseudo miss to real Miss.

| Penalty | Architectures | | | |
|---|---|---|---|---|
| | Fully | VC | RSI | DM |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0.305 | 0.2191 | 0.2161 | 0.2011 |
| 2 | 0.3566 | 0.2788 | 0.2882 | 0.3138 |
| 3 | 0.4 | 0.3334 | 0.3581 | 0.4549 |
| 4 | 0.4278 | 0.3799 | 0.4283 | 0.6471 |

The buffer size of all simulation does not change a lot with these different architectures. Buffer size ranges from 1 to 57 depending on the penalty size.

## 2.6 Summary of this chapter

In order to speed up the route lookup process, a cache architecture is generally used in network processors. A cache is a small memory with a short memory access time.

Based on the conventional caching technique, we proposed three improved schemes that need no special memory devices. They are investigated to reduce conflict misses of general direct-mapped cache architecture. Their implementations in pipeline are also analyzed. The simulations on IPv6 routing show our proposed schemes achieve better performance[29,30]. The summary of our features is as follows.

- **Victim caching:** The VC hit rate increments range from 1.68% to 13.99% depending on the cache size,when comparing with a simple direct-mapped cache. DM cache cooperating with a 16-entry victim cache can achieve a hit rate better than 2-way or 4-way set associative cache of the same size.
- **Randomly selected Index:** the cache hit rate increments of two traces (20031214 and 20031215) with RSI range from 9.98% to 11.29% over DM cache. The RSI performance is between 2-way and 4-way set associative with the same cache size.
- **Combination of the two methods:** There is about 1%~3% over that of using one schemes, if we put the VC and RSI together.
- **Pipelining:** The route lookup throughput is increased significantly using pipelining. Compared with fully associative caching without pipelining, the speedup of throughput ranges from 1.89 to 1.99 depending on the miss penalty and cache size. Given a 512-entry main cache, the throughput speedup is ranging form 1.75 to 2.99 depending on pipeline stage and time penalty as compared with different schemes without pipelineling.

# Chapter 3

# Cache Replacement Policies for IP Address Lookups

## 3.1 Introduction

In this chapter, the replacement policies of cache management are studied and two new replacement policies are presented aimed at IP routing and address lookups. In the previous chapter, we already know the cache architectures are generally used in network processors to satisfy the high demands on IP routing in terms of speed [31, 32, 33]. This is achieved by simple storing the most frequently accessed routing table entries within a small cache memory that has much shorter access latency than the main memory where the routing table is usually stored. By caching, if a destination address does not match any entry in cache, the whole routing table is searched, and then the cache is updated. In order to decide which entry in the cache should be replaced with a new one from routing table, a replacement policy needs to be considered.

In current cache organizations, two replacement policies are commonly used: Least Recently Used (LRU) and Least Frequency Used (LFU). Both of them take into account one parameter in replacement estimation. In this chapter, we propose two new policies, namely Least Access and Recently Used (LAR) and Relatively Least Average Interval (RLAI) policy, which help to achieve higher cache hit rates and in turn to make routing table lookups fast. These two policies are utilized in a simple fully associative cache while running both IPv4 and IPv6 traces respectively, and enhance both IP routing performance without cost overhead.

This chapter is organized as follows. In Section 3.2, two conventional replacement policies are described. Section 3.3 introduces our two novel replacement policies. The simulation results using IPv4 and IPv6 traces are provided in Section 3.4 Some conclusions are presented in Section 3.5.

## 3.2 Existing replacement policies

In cache architectures, a well-known factor that is affecting the performance of caches in general is the replacement policy. As for IP routing, when a fully associative cache saved a part of route entries is full and another entry found a match in the routing table needs to be included; a decision must be made to decide which entry in the cache to be replaced with a new one. Least Recently Used [43] and Least Frequently Used [44] are two of the most commonly used replacement policies in cache systems. These policies are briefly described below.

*Least Recently Used (LRU) policy:* this policy keeps a list of entries that are currently in the cache. When an entry is accessed, this entry is moved to the front of the list. When a miss occurs and the cache is full, a replacement is needed; the entry at the bottom of the list is removed. The new entry found in memory is placed into the cache and the list is updated. Simply said, the LRU policy evicts the entry that has not been accessed for the longest time.

*Least Frequently Used (LFU) policy:* this policy evicts the entry that has been the least frequently used. The motivation behind this policy is that some entries are accessed more frequently than others in a given time. This policy sets an access counter as an estimate of the frequency. The entry with the lowest access count is removed from the cache.

## 3.3 Proposed replacement policies

Replacement policies are important when storage conflicts occur. Some 'faulty' replacements can lead to the removal of still useful cache entries. An effective replacement policy should help to enhance a cache hit rate with inexpensive implementation. In this section, we propose two novel replacement policies, which involve more than one parameter to make an eviction decision. These policies can yield a higher cache hit rate than the traditional LRU and LFU policies. In order to make the following description concise, we use IPv6 routing information to illustrate these two policies.

### 3.3.1 Least Access and Recently used (LAR) policy

The goal of this policy is to evict the relatively inactive entry. An inactive entry is the entry that has not been accessed for a relatively long time and, potentially, it will not be accessed in the near future. The potential future access is determined by a parameter introduced in the new replacement policies which is based on the history of the particular entry.

By analyzing IPv6 traces, we found an interesting case that an entry just evicted from cache by either LRU or LFU policies tends to be re-accessed in a short time later. Table 3.1 shows an example of the number of entries that are re-accessed within M number of lookups after being evicted by the LRU policy. This table shows different number of cache entries.

Table 3.1: The number of re-accessed evicted entries for trace (A)031214.

| Num.of cache entries | Num. of evicted entries | Num.of lookups after eviction(M) | | | | |
|---|---|---|---|---|---|---|
| | | 0- 50 | 51-100 | 101-200 | 201-300 | 301-400 |
| 64 | 587968 | 69174 | 52114 | 64108 | 40614 | 30475 |
| 128 | 369922 | 18147 | 14972 | 24714 | 19681 | 17223 |
| 256 | 205667 | 4782 | 4297 | 7673 | 6612 | 6142 |

As for a given 64-entry cache, more than 69,000 entries among all the 587,968 entries are evicted but are re-accessed within the next 50 lookups. Over 50,000 evicted entries are re-accessed between 51 and 100 lookups. Evicting entries that are accessed within a small number of lookups decreases the cache efficiency. LRU or LFU have no means to address this issue since they are implemented solely based on one aspect of the past activity of entries in the cache. One is based on the unaccessed time, and the other is on the access count. If we take into account both aspects, and choose the inactive entry in cache, this will increase the hit rate. This in turn decreases the average memory access time.

In Figure 3.1, list A is a group of cache entries sorted by unaccessed time; this is the same as that of the LRU policy. The recently accessed entry B is put in the top of list. The LAR replacement policy removes the entry with the lowest access count among the bottom N recently unaccessed entries in the list. In the example, entry C is the candidate to be evicted by the LAR policy due to its lowest access count. However, if N is close to the number of the cache entries, and the candidate happens to be accessed recently, and then we evict the least recently used entry instead of this candidate. That is, the evicted entry by LAR is the result of considering both access time and access count.

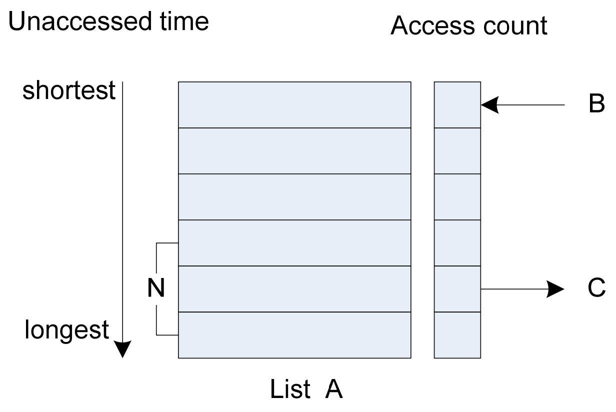


Figure 3.1: LAR replacement policy.

The value of N could be optimized by analyzing the cache performance under different N. Table 3.2 shows the hit rate increments of LAR over LRU policy for different N with a 128-entry cache organization. Figure 3.2 shows that for N equal to 32 (1/4), the LAR policy achieves the best performance on the sample IPv6 traces. For the traces we have studied the value of N that yields the highest hit rate is 1/4.

Table 3.2: Hit rate increment of LAR over LRU with different size N (128-entry cache).

| Trace | N | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1/8 | 1/4 | 3/8 | 1/2 | 5/8 | 3/4 | 7/8 | 1 |
| A | 1.22 | 1.82 | 1.79 | 1.29 | 0.98 | 0.54 | 0 | 0 |
| B | 1.05 | 1.61 | 1.53 | 0.98 | 0.76 | 0.46 | 0 | 0 |
| C | 0.92 | 1.27 | 1.08 | 0.87 | 0.81 | 0.58 | 0 | 0 |
| D | 0.53 | 0.68 | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 0.51 | 0.83 | 0.55 | 0 | 0 | 0 | 0 | 0 |
| F | 0.55 | 0.96 | 0.99 | 0.83 | 0.37 | 0.11 | 0.02 | 0.02 |

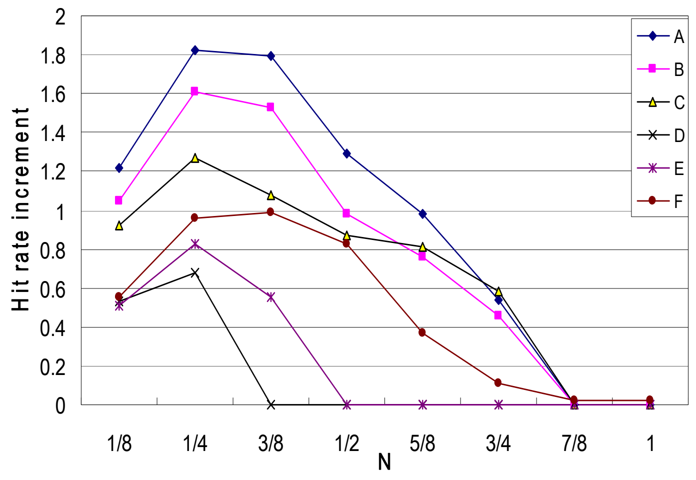


Figure 3.2: Plot of the hit rate increments of LAR over LRU with different N.

### 3.3.2 Relatively Least Average Interval (RLAI) policy

The prime motivation for this policy is similar to LAR, but a different parameter is considered when making replacement decisions. An array is used to save the average interval between two accesses of the same entry. This interval is determined by counting the number of lookups between such two accesses. Each time the entry is accessed a new average interval is computed. If an entry is not accessed for a time larger than its average interval, then it is considered as an inactive entry and it has the potential to be evicted. This algorithm chooses an entry with the longest average unaccessed time among these entries potential evicted. The implementation is depicted in Figure 3.3.

These replacement policies can be implemented using a simple linked list in hardware. This list needs few entries with the size of the optimum N of the total cache. These entries are arranged in the order of access count or average access interval.

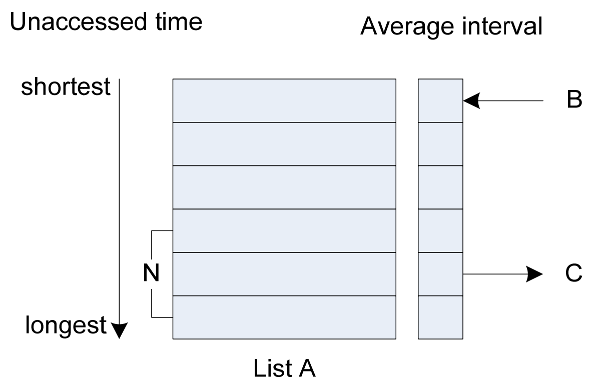


Figure 3.3: RLAI replacement policy.

## 3.4 Performance evaluation

In this section, we implement our proposed replacement policies on cache architecture, and compare their performance to the LRU policy; we first use IPv4 destination address traces to evaluate the cache performance, and then extended to apply these replacement policies on IPv6 traces to indicate that the policies proposed also contribute to this new protocol. The routing information is shown in Table 3.3. We have labeled these files from X to Z for IPv4 and from A to F for IPv6 to ease file references.

Table 3.3: IPv4 and IPv6 traces.

| IP | Trace files | Number of unique addresses | Ratio |
|---|---|---|---|
| IPv4 | (X) 060524 | 70627 | 28 |
| | (Y) 060609 | 76533 | 26 |
| | (Z) 060615 | 63485 | 32 |
| IPv6 | (A) 031214 | 3949 | 506 |
| | (B) 031215 | 3681 | 543 |
| | (C) 031216 | 3422 | 584 |
| | (D) 040128 | 3764 | 531 |
| | (E) 040129 | 3874 | 516 |
| | (F) 040130 | 3772 | 530 |

We also obtain IPv4 routing tables from the website of University of Oregon Route Views Archive Project http://archive.routeviews.org/ [35]. In order to keep these tables neat, we delete those redundant entries in tables. And obtain IPv6 routing tables both from the machine “n6tap.es.net” on the website of 6TAP router information http://www.6tap.net/ [36] and from the machine “route-server.he.net” on Hurricane Electric Internet Services http://lg.he.net/cgi-bin/index.cgi [37] via telnet. Actually, since currently there are few users on IPv6, the sample IPv6 routing table size is not large. We have extended or combined tables to satisfy the unique destination addresses in the trace files.

In our evaluation, we still choose hit rate as the measurement of performance for cache architecture with different replacement policies. We do not include the LFU policy to make comparisons, because its performance is usually worse than that of LRU. Table 3.4 is an example of trace (B) 031215 to show the difference between these two policies. Obviously, the hit rate of LFU is less than that of the LRU by up to 19 % depending on different cache sizes. The other test traces share the same trend as trace (B).

Table 3.4: Difference in hit rate between LFU and LRU of trace (B) 031215.

| Policy | Num.of cache entries | | |
|---|---|---|---|
| | 64 | 128 | 256 |
| LFU | 56.05 | 67.75 | 81.60 |
| LRU | 75.70 | 84.53 | 91.18 |
| Difference | 19.65 | 16.78 | 9.58 |

### 3.4.1 IPv4 performance evaluation

The following Table 3.5 shows the cache hit rate of the three replacement policies, including the two novel policies we introduces with different number of cache entries. Figure 3.4 indicates that the average hit rate changes with the increasing cache entries.

Table 3.5: Cache hit rate with different replacement policies (IPv4).

| Num.of cache entries | Policy | Traces | | | |
|---|---|---|---|---|---|
| | | X | Y | Z | Average |
| 512 | LRU | 73.94 | 69.12 | 74.76 | 72.61 |
| | LAR | 78.11 | 74.60 | 79.51 | 77.41 |
| | RLAI | 78.26 | 74.85 | 79.70 | 77.60 |
| 1K | LRU | 83.98 | 81.45 | 85.43 | 83.62 |
| | LAR | 85.58 | 82.52 | 87.10 | 85.07 |
| | RLAI | 85.54 | 82.32 | 87.10 | 84.99 |
| 2K | LRU | 89.53 | 88.04 | 91.29 | 89.62 |
| | LAR | 90.1 | 88.85 | 91.72 | 90.22 |
| | RLAI | 90.04 | 88.81 | 91.70 | 90.18 |
| 4K | LRU | 92.71 | 91.88 | 94.12 | 92.90 |
| | LAR | 92.89 | 92.41 | 94.35 | 93.22 |
| | RLAI | 92.84 | 92.40 | 94.32 | 93.19 |

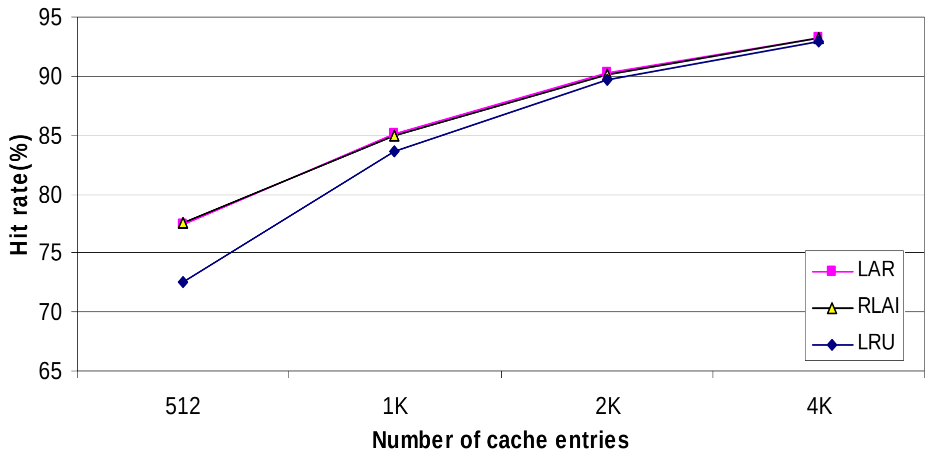


Figure 3.4: Average cache hit rate performance.

From the above information, it can be observed that both LAR and RLAI policy have a better performance than the common LRU. If LRU is applied, only one parameter is considered in replacing an entry. This entry, however, may be needed shortly after it has been removed. When using our proposed policies, this problem is alleviated. Our simulation results show that the hit rate is increased from 0.18% to 5.48% for LAR and from 0.13% to 5.73% for RLAI, depending on the trace files and the number of cache entries.

Table 3.6 shows another aspect of cache performance, i.e. increment ratio, which is the ratio of the hit rate increment by novel replacement policy to the hit rate increment by doubling the number of cache entries. Obviously, without any change in cache size, our proposed policies can achieve up to 46% increment performance by comparing with doubling the cache. Although the hit rate improves as the cache size doubles, the expense on caches also increases, especially for expensive fully associative cache. Thus our replacement policies are of advantage to enhance cache performance without the overhead cost.

Table 3.6: Increment ratio of cache performance.

| Num. of cache entries | Policy | Traces | | | |
|---|---|---|---|---|---|
| | | X | Y | Z | Average |
| 512 | LAR | 41.51 | 44.5 | 44.51 | 43.51 |
| | RLAI | 43.01 | 46.49 | 46.31 | 45.27 |
| 1K | LAR | 28.72 | 16.37 | 28.50 | 24.53 |
| | RLAI | 28.07 | 13.26 | 28.53 | 23.29 |
| 2K | LAR | 17.82 | 21.19 | 15.10 | 18.04 |
| | RLAI | 16.01 | 20.02 | 14.36 | 16.80 |

Obviously, the LAR and RLAI policy are actually good at achieving higher hit rate with regard to IPv4 routing information. We will show below both of them keep effective when extending to IPv6 routing lookups, which is important for Internet development in future.

### 3.4.2 IPv6 performance evaluation

By applying these two replacement policies LAR and RLAI, the cache hit rate with different number of cache entries when processing six IPv6 traces is explained in Table 3.7. Figure 3.5 depicts the different hit rates for a 128-entry cache.

Similar to IPv4 routing, our simulation results of IPv6 show that the hit rate is increased from 0.17% to 2.35% for LAR and from 0.83% to 2.58% for RLAI, depending on the trace files and the number of cache entries. Actually, LAR and RLAI yield over thirty thousand additional matches (or hits) as compared to LRU.

Table 3.7: Cache hit rate with different replacement policies (IPv6).

| Num.of cache entries | Policy | Traces | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | A | B | C | D | E | F | Average |
| 64 | LRU | 70.60 | 75.70 | 76.63 | 80.34 | 81.64 | 84.01 | 78.15 |
| | LAR | 72.27 | 76.95 | 77.89 | 81.36 | 82.34 | 84.18 | 79.17 |
| | RLAI | 73.01 | 77.64 | 78.59 | 81.90 | 82.93 | 84.84 | 79.82 |
| 128 | LRU | 81.50 | 84.53 | 85.26 | 88.49 | 88.96 | 89.85 | 86.43 |
| | LAR | 83.31 | 86.14 | 86.53 | 89.17 | 89.79 | 90.84 | 87.63 |
| | RLAI | 83.85 | 86.57 | 87.12 | 89.90 | 90.30 | 91.27 | 88.17 |
| 256 | LRU | 89.70 | 91.18 | 92.13 | 93.88 | 94.14 | 94.83 | 92.64 |
| | LAR | 92.05 | 93.07 | 93.56 | 95.04 | 95.16 | 95.61 | 94.08 |
| | RLAI | 92.28 | 93.48 | 94.02 | 95.27 | 95.29 | 95.77 | 94.35 |

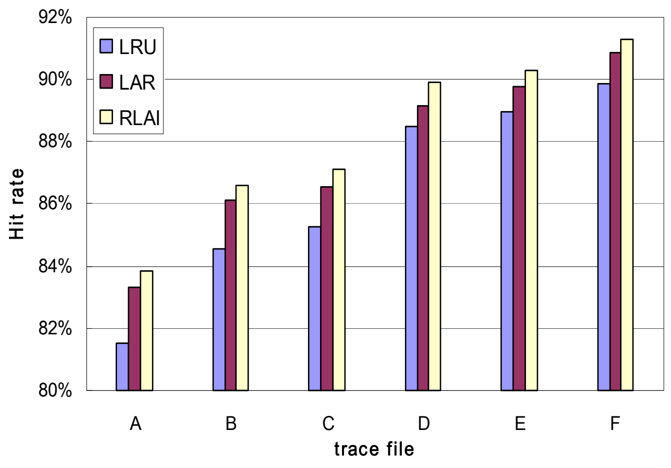


Figure 3.5: The performance of cache with the size of 128 entries.

Figure 3.6 depicts the average hit rate with the increasing cache size (number of entries). As we mentioned before, since the cache cost must be increasing with the increasing cache size, our replacement policies help to enhance cache performance. The RLAI is a little better than the LAR, and both of them have higher hit rate than the LRU policy for different cache sizes. The improvement rate diminishes as the cache size grows. The higher hit rate in turn is translated into higher speedup of the routing table lookup.

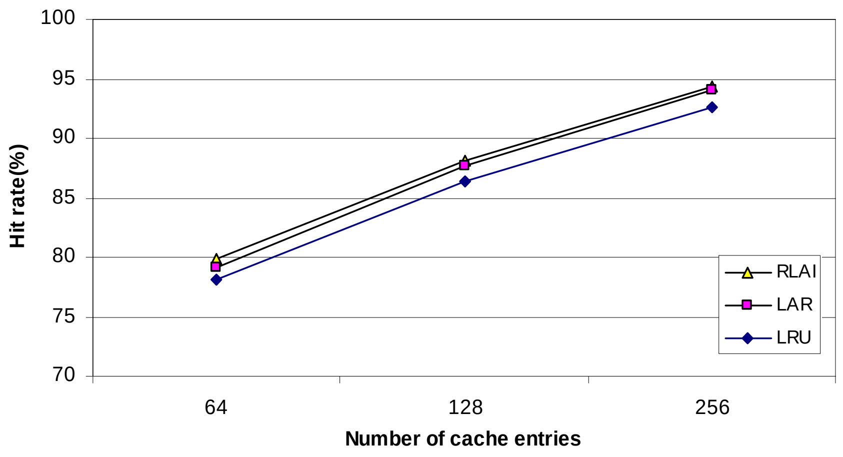
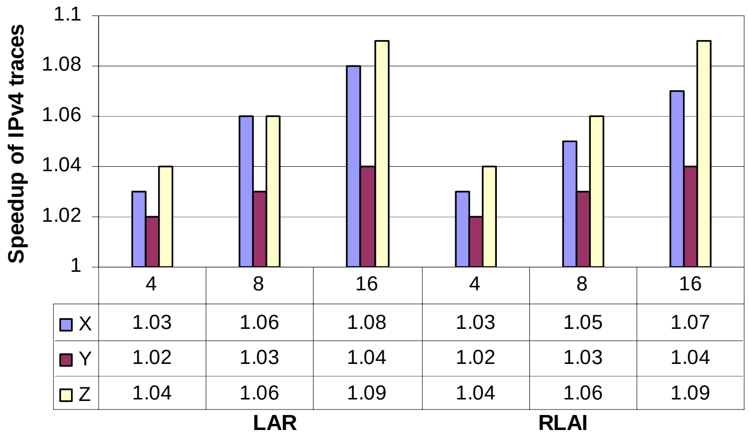


Figure 3.6: Average cache hit rate with increasing cache entries.

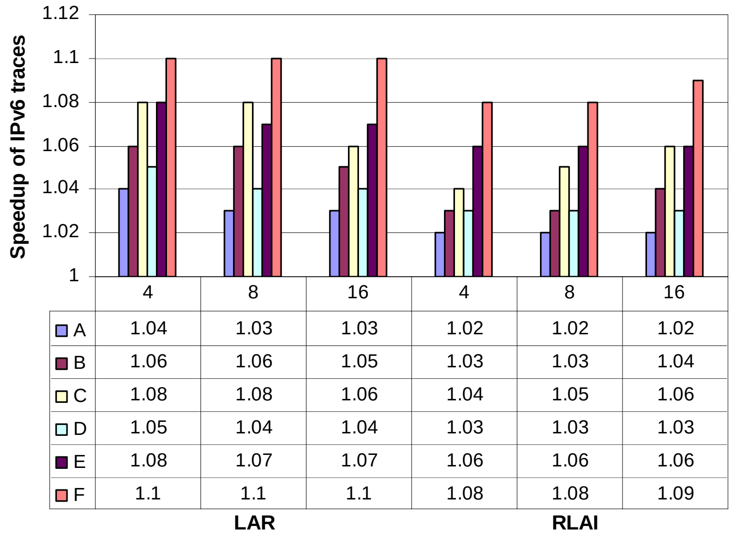
## 3.4.3 AMAT speedups

To compare the impact of the proposed policies on the average memory access time (Equation 2.1), we use speedup as the main metric [24]. The speedup here is the ratio of the AMAT by LRU and the proposed policies. For example, the speedup of RLAI is shown in Equation 3.2. Assume routing table access time is L times larger than cache access time.

$$Speedup = \frac{AMAT_{LRU}}{AMAT_{RLAI}} \qquad \text{(Equation 3.2)}$$



| | LAR | | | RLAI | | |
|---|---|---|---|---|---|---|
| | 4 | 8 | 16 | 4 | 8 | 16 |
| X | 1.03 | 1.06 | 1.08 | 1.03 | 1.05 | 1.07 |
| Y | 1.02 | 1.03 | 1.04 | 1.02 | 1.03 | 1.04 |
| Z | 1.04 | 1.06 | 1.09 | 1.04 | 1.06 | 1.09 |



| | LAR | | | RLAI | | |
|---|---|---|---|---|---|---|
| | 4 | 8 | 16 | 4 | 8 | 16 |
| A | 1.04 | 1.03 | 1.03 | 1.02 | 1.02 | 1.02 |
| B | 1.06 | 1.06 | 1.05 | 1.03 | 1.03 | 1.04 |
| C | 1.08 | 1.08 | 1.06 | 1.04 | 1.05 | 1.06 |
| D | 1.05 | 1.04 | 1.04 | 1.03 | 1.03 | 1.03 |
| E | 1.08 | 1.07 | 1.07 | 1.06 | 1.06 | 1.06 |
| F | 1.1 | 1.1 | 1.1 | 1.08 | 1.08 | 1.09 |

Figure 3.7: AMAT speedups of LAR and RLAI.

Figure 3.7 depicts the speedups gained by the LAR and RLAI policies over LRU for all traces. The AMAT speedup is improved up to 9% by the LAR and RLAI policy for IPv4 traces. With regard to IPv6, this speedup by the LAR policy is increased by up to 10%, while the RLAI policy improves the AMAT ranging from 2% to 9%. It can be observed that the AMAT speedups are increasing with the increasing time penalties.

## 3.5 Conclusions

In a cache scheme, a replacement policy is used to make a decision about which entry in cache should be removed to allow a new one to enter. This policy affects the cache performance. In order to increase the cache hit rate for IP address lookup, we have proposed two novel replacement policies and evaluated their impact. As compared to the commonly used Least Recently Used (LRU) policy, the simulations show our proposed policies achieve better performance on both IPv4 and IPv6 routing[34]. The features of our replacement policies are as follows.

- **Least Access and Recently used (LAR) improves hit rate from 0.18% to 5.48% for IPv4 and from 0.17% to 2.35% for IPv6 over LRU, respectively.** This policy makes a removing decision based on two major parameters: unaccessed time and access count. It evicts the entry with the smallest access count among N entries that have not been accessed for a long while.
- **Relatively Least Average Interval (RLAI) policy improves hit rate from 0.13% to 5.73% for IPv4 and from 0.83% to 2.58% for IPv6 over LRU, respectively**. It uses the average interval between two accesses of one entry as a

factor to make a decision. If an entry has not been referred longer than its previous access interval, it can be potentially evicted.

- **Our newly introduced policies tend to evict more inactive entries.** Both policies tend to remove an inactive entry by evaluating its previous access references, which include unaccessed time and access count or unaccessed time and average access interval.
- **Alleviating required access of recently evicted entries.** LRU performs evictions depending only on unaccessed time. An entry evicted from cache may be referred in a short period of time after its eviction; this is a larger problem for small caches. Our policies can alleviate such situation because they evaluate one entry based on multi-conditions, and they tend to keep more active entries in the cache to reduce the number of the replacements.
- **LAR and RLAI polices have speedups from 2% to 10% when considering average memory access time for both IP.** Due to the higher cache hit rate yielded by the LAR and RLAI policies, fewer accesses to routing table memory are needed for address lookups. Consequently, this helps to reduce the average memory access time, which is often used as measurement for cache-memory performance.

A replacement policy is a way to improve caching effectiveness. A good policy is able to increase the hit rate and should be inexpensive to implement. A high hit rate makes average memory access time fast and in turn speeds up the address lookup operations.

# Chapter 4

# Routing Table Entry Compaction Schemes in Ternary CAM

## 4.1 Introduction

In this chapter four novel routing table entry compaction schemes based on different rules to perform fast routing lookups in Ternary CAM are presented. As we well know, IP routing lookup operation is time consuming, because it requires the storage in memory of predetermined routing paths for all possible network destinations. With the continuous growth of the users in the Internet, the size of routing table is increasing steadily. Thus, a much larger memory is required to store table contents in local routers. On the other hand, the lookup speed may be adversely affected since the complete routing table is stored in main memory with slow access time. Thus, the reduction of the number of routing table entries is a potential solution to improve lookup speed, contain the routing table size explosion and save memory storages required.

The proposed compaction schemes are implemented and used in a cache organization (Chapter 2), which is simple storing the most frequently accessed routing table entries within a small cache memory that has much shorter access latency than the main memory where the routing table is usually stored. The objective of these schemes in this chapter is to achieve high caching effectiveness without wasting memory usage.

The contents of this chapter are organized as follows. In section 2, an overview of previous work and data structure on routing table compaction is described. Section 3 presents our proposed compaction schemes. Section 4 provides the performance evaluation of IPv4 and IPv6 routing respectively. Section 5 concludes with a summary of the contributions of this chapter.

## 4.2 Backgrounds of routing table entry compaction

In this section, we begin with a description of the Ternary CAM feature, which is particular important in the presentation of our compaction schemes, and then discuss some compaction approaches by other researchers.

### 4.2.1 Ternary Content Addressable Memory (TCAM)

When we implement Longest Prefix Matching (LPM) lookup mechanism in a conventional memory, we need to store both a network address and its address mask in each route entry. Given a destination address, perform bit-wise AND operation between this destination address and its address mask in each entry, and then check if the result is equivalent to the network address in the same entry. Since the destination address of an arriving packet does not carry any prefix length information, the destination address is required to be compared with all the route entries in the routing table. It makes the IP address lookup operation a time-consuming task.

In order to expedite the packet processing, a content addressable memory (CAM) is used as a hardware-based solution. There are two types of CAM, namely binary CAM and ternary CAM.  Binary CAM cell only stores 0 or 1, and performs fixed-length comparisons and exact match search operation within one memory access. However, it is not suitable for LPM effectively. As for an IPv4 32-bit address lookup, it is required an exact match search in 32 separate CAMs. Obviously, this complexity is increasing when we implement IPv6 routing with 128-bit address format.

A ternary CAM (TCAM) makes this LPM process more direct by its special features. Different from the usual memories, each cell of TCAM stores one of three states: 0, 1 and *don't care* (*)[14, 38]. The TCAM makes the implementation of LPM much easier. All bits with the low order in an address below the prefix boundary are replaced with *don't care*. These bits are ignored when there is a comparison with the TCAM entry. TCAMs take the input address as a search key, and perform a parallel search of all entries in hardware in a single clock cycle [39]. The port associated with the longest prefix length that matches the search is chosen as output using a priority encoder [40]. An organization for the proposed associative cache is shown in Figure 4.1.

In Figure 4.1 an example of an address matching in this TCAM is included. Given a packet with a destination address 10110010, three matching entries 1011001*, 10110*** and 101***** are found in the routing table. This packet will be delivered to the port associated with the first entry by the LPM mechanism which is implemented by means of a priority decoder.

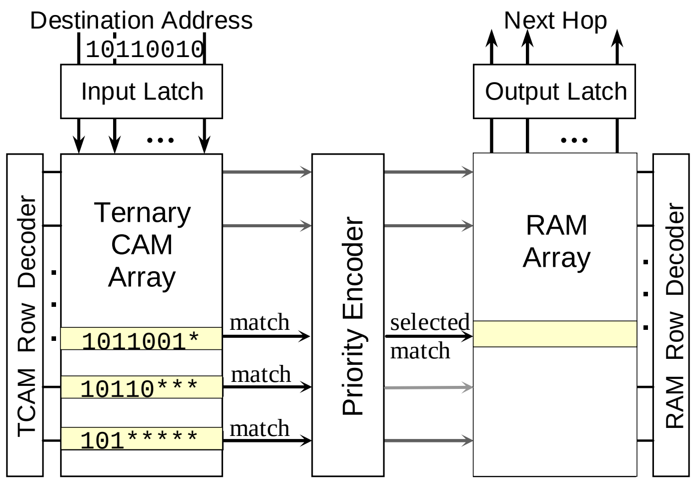


Figure 4.1: A TCAM address lookup organization.

### 4.2.2 Previous approaches on routing table compaction

Liu[39] proposed two techniques to reducing the size of routing tables stored in TCAM based on IPv4 routing information. The first technique is called pruning. It eliminates those redundant routing prefixes without affecting routing functionality. The second technique is mask extension, which exploits TCAM hardware's flexibility. It combines the routing prefixes that point to the same port number by using an arbitrary mask. That is, the bits of ones or zeros in mask needn't be continuous. The mask extension technique can be reduced to a logic minimization problem and Liu used Espresso-II algorithm to compute the minimal cover. The experimental result shows that applying pruning alone reduces the routing table size by roughly 25%, and mask extension without pruning can reduce table size between 27.3% and 30.4%. By applying both of them, the overall size can be reduced up to 48%.

Rooney [15] presented a similar approach to minimize the routing tables based on the TCAM bit pattern associative array. First, IP routing table is separated by prefix into sets, starting at the top by the longest network address; each set is separated into groups by output port number, and then each group is minimized with the Espresso program. That is, the minimization is performed only on the entries within the same set and having the same output port. After minimization, each group is returned to the original set. The test result shows the average reduction of routing table entries is 24.4%, and this reduction leads to improvements in cache hit rate in cache architecture.

## 4.3 Proposed compaction schemes

As we mentioned before, TCAMs have the main advantage of simplicity and can perform parallel searches to reduce the lookup time. Consequently, it can achieve high lookup throughput. However, a TCAM is usually more expensive as compared to a conventional memory and it dissipates more power. Therefore, a large TCAM may be too costly to be a feasible solution. On the other hand, a large conventional memory usually slows down the lookup process. This in turn creates a need for investigating compaction schemes cooperated with fast cache memory solutions. If the number of entries in the routing table can be compacted, the table can be contained into fewer TCAM chips. Thus, the process of address lookup could be improved effectively and economically with a cache organization. In this section, the features of IP routing tables are analyzed and followed by four compaction schemes based on different compression rules.

### 4.3.1 IP routing table features

Table 4.1 shows some features of several routing tables we used in our experimental simulation. Obviously, in a routing table, the number of port is much smaller as compared with the number of routing entries, because of the limited number of interfaces in a router. That is, there exists more than one entry with different prefix or content associatived with the same output port number. This in turn indicates the possibility of compaction to combine more relative entries into one without mis-routing function.

Table 4.1: IP routing table feature.

| IP | Routing table | No. of ports | No. of entries |
|---|---|---|---|
| IPv4 | RT1 | 43 | 90,000 |
|  | RT2 | 50 | 90,000 |
| IPv6 | RT1 | 90 | 3,685 |
|  | RT2 | 105 | 3,674 |

### 4.3.2 Compaction of same-port entries (Scheme C1)

This scheme is similar to those in [15,39].We scale and implement it to satisfy the long address format requirement in IPv6 routing. The basic steps are as follows.

- The entry addresses with the same destination port are considered for compaction.
- Two addresses that differ by only one bit including *don't care* (*) are candidates for compaction.
- If these conditions are satisfied, the entries with these two addresses can be combined into one by using *don't care* (*) to replace the bit.
- The two entries that have been combined are removed from further consideration
- The new entry is included in the potential entries for further compaction.

Assume that there are four destination addresses that share the same port number as shown in Figure 4.2. The addresses in entry *a* and *b* are the same except for the last bit.

These two entries can be combined into a new entry 00100110*. The same compaction can be performed on the entries *c* and *d*. The updated routing table has two entries after the first compaction. In this example, these two entries can be compacted since they have only one bit that is different. The compaction process continues until there are no potential entries to be compacted. In our example, those four entries are compacted into one with 2-bit ‘*’ using this scheme.

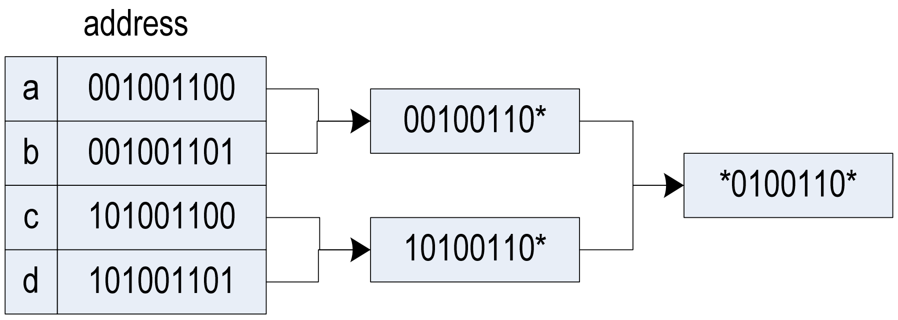


Figure 4.2: An example of compaction scheme C1.

### 4.3.3 Compaction of non-existing entries (Scheme C2)

We have developed another compaction scheme, which combines many more entries than scheme C1. This scheme uses the compacted entries produced by C1. The improvement is due to the way *don’t care* (*) are handled. Below are the additional steps that are needed to implement scheme C2.

- Two addresses with the same destination port that have only one different bit excluding *don’t care* (*) are candidates for additional compaction.
- If there is no other address with a different destination port that can be compacted with one of the two addresses in the previous step, then the compaction can be performed.

- If the previous step is not possible the current compaction is abandoned.

Figure 4.3 shows an example where entry d is the compaction result of entry *a* and *b* (Figure 4.2) by replacing the last bit with a *don't care* (*). It shares the same port number A with entry *c*. When comparing the addresses in entry *d* and *c*, we take into account the *don't care* '*' in the address of entry *d* that could have a match with the 0 in the same bit position in entry *c*. If the addresses in these two entries have exactly one different bit with the exception of the *don't care* (*), they could potentially be compacted into a new entry. Then other entries with different port assignments are checked to make sure that there is no entry that falls within the new entry (in the example is labeled as entry *e*) address range. If an entry exists (in the example entry *f*) this compaction is not performed. Otherwise, entry *c* and *d* are replaced by *e*.

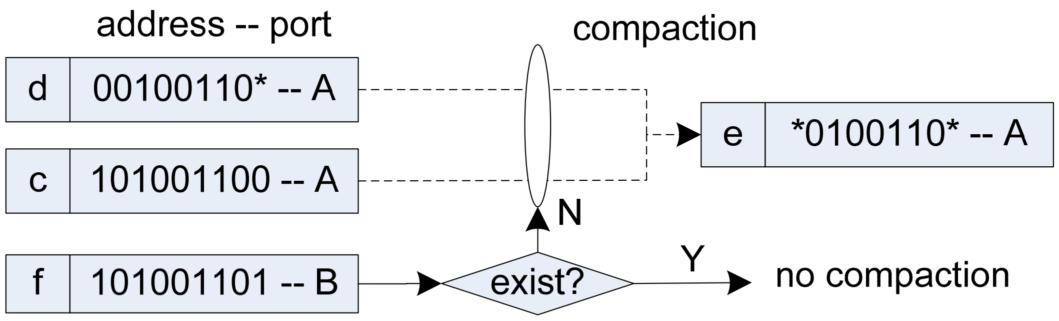


Figure 4.3: An example of compaction scheme C2.

### 4.3.4 Compaction using a threshold with continuous *don't care* (Scheme C3)

The goal of this scheme is try to further compact the routing table, even though this may cause a routing conflict. A routing conflict is defined as having two entries with different port assignments to have a match with an incoming address. We add restrictions to this compaction to reduce the side - effect of the routing conflicts. The following steps are incorporated to implement scheme C3.

- Two addresses with the same destination port that have only one different bit excluding *don't care* bits (*) are candidates for a compaction.
- If there is no routing conflict in the routing table, the compaction is performed.
- If there exists only one entry address with a different destination port, which is equivalent to the current compressed address, and the number of the continuous *don't care* bits in this compacted entry is less than a given threshold, then the compaction can be performed. In addition, the entry with the different port assignment is marked with higher searching priority, that is, it is searched before any search for the compacted entry.
- If the previous steps are not possible, the current compaction is abandoned.

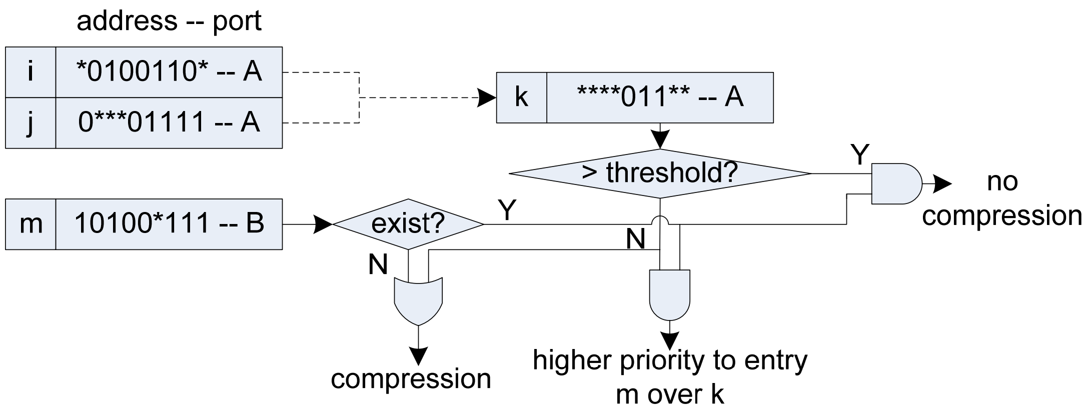


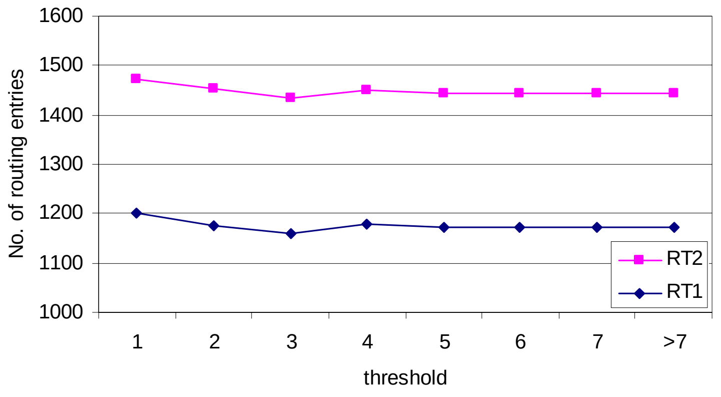
Figure 4.4: An example of compression C3.

Figure 4.4 provides a graphical representation of this scheme, where entry *i* and *j* can potentially be compacted into entry *k*. Since we take into account *don't care* that can be either 0 or 1 in comparison, it is possible that the compacted address in entry *k* has continuous *don’t care*. A long sequence of *don’t care* increases the probability of a routing conflict.

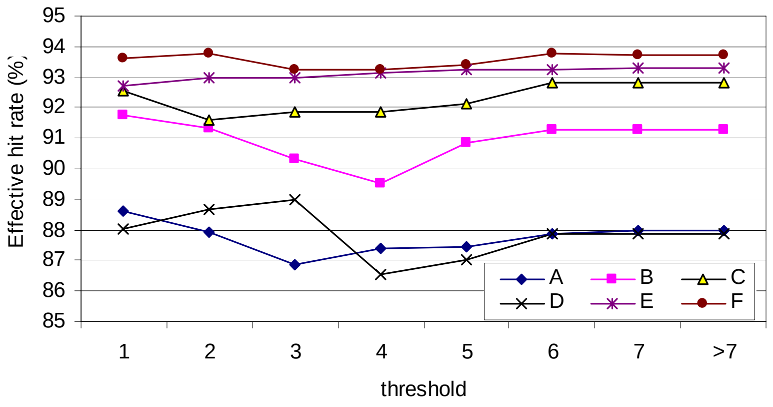
The purpose of setting a threshold of the number of continuous *don’t care* is to diminish such a problem. If an address in entry *m* with a different port assignment is equivalent to the address in entry *k*; and the continuous *don’t care* number is larger than the given threshold, this compaction is not performed. Otherwise, we replace entry *i* and *j* by *k*, and make entry m to have higher priority over entry *k* in the routing table to avoid incorrect routing in memory.

The value of the threshold is determined by the contents of the routing tables and the destination addresses in statistic studying. Obviously, this value affects the compaction performance, which in turn impacts the hit rate in cache organization. A large threshold value is usually helpful in compacting more routing entries, as well as increases the possibility of routing conflicts, thus causes more port errors (explained later in Chapter 5). Therefore, we should take into account both the number of routing entries and the effective hit rate, which is the difference between actual cache hit rate and port error ratio, and choose a threshold value to achieve a better performance.

Figure 4.5(a) shows the effect of choosing different thresholds on two IPv6 routing tables: RT1 and RT2 as examples; Figure 4.5(b) depicted the effective hit rate of all six IPv6 traces, namely A to F. We should point out that extensive simulations show the value of 5 generates a smaller port error ratio with fewer routing entries, and also need far less searches in routing table when implementing our improved sampling techniques to alleviate the port error problems (in Chapter 5 ). By the same procedure of analyzing, we can obtain suitable threshold value for IPv4 routing information.



(a) Number of compacted routing entries by different threshold values.



(b) Effective hit rate by different threshold values.

Figure 4.5: Scheme C3's threshold impact on entries and effective hit rate.

### 4.3.5 Compression using a threshold with the number of difference bits (Scheme C4)

Compaction scheme C4 is similar to scheme C3; the only difference is the threshold criteria. Instead of the continuous *don't care* as in scheme C3, the number of different bits between the addresses before and after a compaction are considered to compute a number to be compared to the threshold.

The C4 scheme steps are similarly to scheme C3. We explain scheme C4 by means of an example. Assume there are addresses such as *0100110* in entry x 0****1111 in entry y, these addresses can potentially be compacted to an address *****11** in entry z. Assume that an address 10100*111 in entry w with a different port assignment exists; entry z would cover this entry and has a port assignment conflict.

The number of different bits between the address in the potential new entry z and entries x or y are computed, these are 5 and 3, respectively. If these numbers are larger than a given threshold, then this compaction is abandoned. Otherwise, entries x and y are replaced by z and entry w is marked with a higher search priority than entry z. As scheme C3, an effective threshold should be determined by achieving a good balance between the number of routing entries and the effective hit rate. In this study the threshold for scheme C4 is set to 3.

## 4.4 Performance Evaluation

In this section, we implement our proposed compaction schemes on TCAM cache architecture, and analyze their effects on IPv4 and IPv6 routing tables, respectively, and we use their corresponding IP packet traces (in Chapter 3) to evaluate the address lookup performance.

### 4.4.1 IPv4 routing performance

We estimate the compaction performance from three aspects as follows: compaction ratio, cache hit rate and average memory access time speedups.

### 4.4.1.1 Compaction ratio

In our evaluation, we first analyze the compaction performance of our schemes. We use compaction ratio (CR) as a measurement. CR is defined as the ratio of the number of the routing entries in the compacted and original tables. Obviously, the smaller the CR is, the better the compaction performance is.

Table 4.2: The four compaction schemes (CS) and their impact on compaction ratio (CR).

| CS | RT1 | | RT2 | |
|---|---|---|---|---|
| | No. of Entries | CR (%) | No. of Entries | CR (%) |
| O | 90,000 | 100.00 | 90,000 | 100.00 |
| C1 | 59,019 | 65.58 | 60,185 | 66.87 |
| C2 | 26,505 | 29.45 | 30,279 | 33.64 |
| C3 | 26,842 | 29.82 | 30,916 | 34.35 |
| C4 | 26,738 | 29.71 | 30,346 | 33.72 |

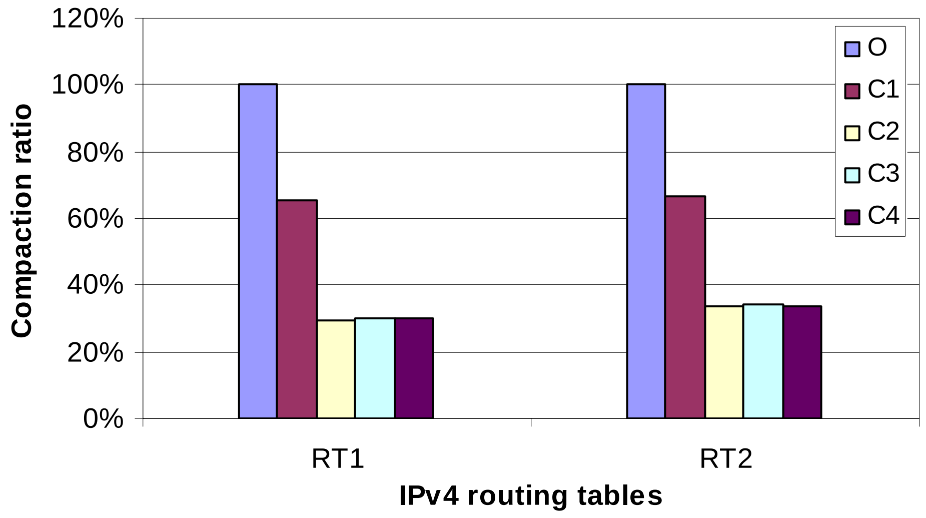


Figure 4.6: Compaction performance of IPv4 routing tables.

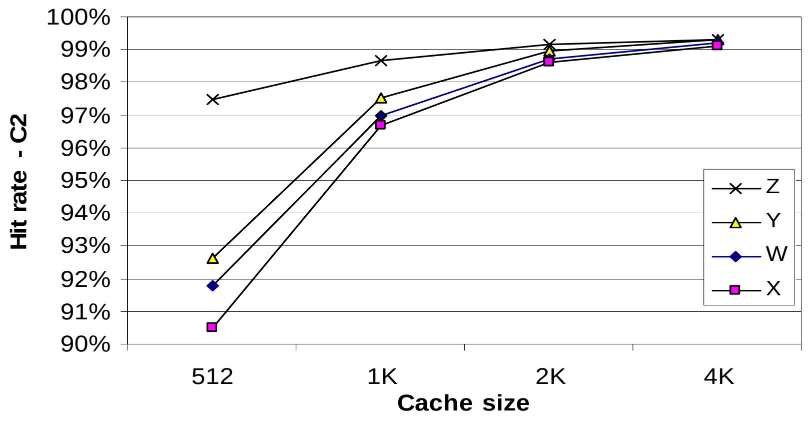
Table 4.2 shows the impact of these schemes on two different IPv4 routing tables RT1 and RT2. Figure 4.6 depicts that the routing table entries are reduced rapidly with these compaction schemes. Compared with the original tables (labeled by "O") without any compaction, Scheme C1's CR is around 65%, which is a good match with Liu's and Rooney's. Scheme C2, C3 and C4 achieve much lower CR than C1, although there is only minor difference among their CRs. Their compaction ratio ranges from 29.45% to 34.35%, which are impressive compaction results.

### 4.4.1.2. Cache hit rate

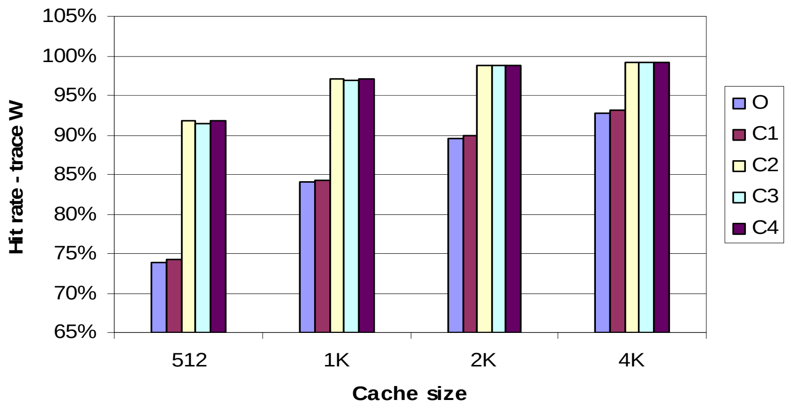
The different compacted routing tables result in the variety of hit rate in the cache architecture. The overall hit rate of cache with different compaction schemes is shown on Table 4.3. During the simulation, we use four IPv4 traces namely W to Z as incoming destination addresses, and perform routing lookups in a cache organization based on original and compacted routing tables.

Table 4.3: Hit rate (%) of cache with different compaction schemes (CS).

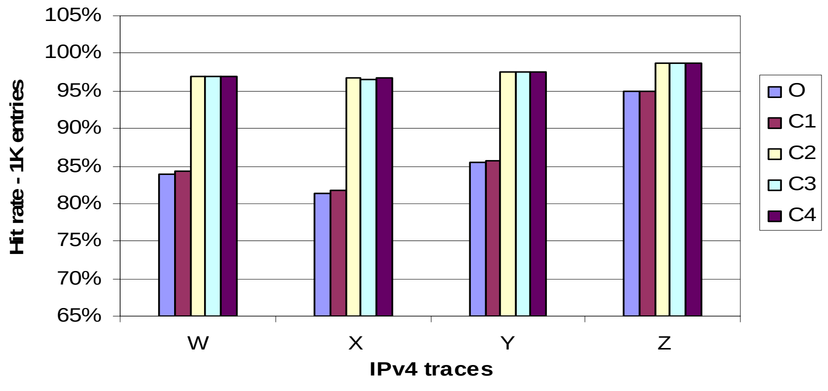
| IPv4 trace | CS | 512 | 1K | 2K | 4K |
|---|---|---|---|---|---|
| W | O | 73.94% | 83.98% | 89.54% | 92.71% |
| | C1 | 74.16% | 84.22% | 89.82% | 93.08% |
| | C2 | 91.79% | 96.99% | 98.73% | 99.20% |
| | C3 | 91.40% | 96.90% | 98.74% | 99.21% |
| | C4 | 91.73% | 97.00% | 98.74% | 99.21% |
| X | O | 69.12% | 81.44% | 88.04% | 91.89% |
| | C1 | 69.36% | 81.73% | 88.40% | 92.34% |
| | C2 | 90.48% | 96.68% | 98.61% | 99.13% |
| | C3 | 90.31% | 96.55% | 98.60% | 99.14% |
| | C4 | 90.40% | 96.69% | 98.59% | 99.13% |
| Y | O | 74.76% | 85.43% | 91.30% | 94.12% |
| | C1 | 75.02% | 85.70% | 91.57% | 94.44% |
| | C2 | 92.62% | 97.53% | 98.94% | 99.31% |
| | C3 | 92.49% | 97.48% | 98.94% | 99.30% |
| | C4 | 92.64% | 97.57% | 98.93% | 99.30% |
| Z | O | 94.04% | 94.89% | 95.59% | 95.82% |
| | C1 | 94.07% | 94.97% | 95.68% | 96.00% |
| | C2 | 97.47% | 98.67% | 99.14% | 99.31% |
| | C3 | 97.43% | 98.65% | 99.12% | 99.31% |
| | C4 | 97.48% | 98.68% | 99.14% | 99.32% |



(a) Hit rate with scheme C2.



(b) Hit rate of trace W



(c) Hit rate on 1K cache

Figure 4.7: Cache hit rate performance.

A minor improvement of the hit rate is obtained when scheme C1 is used, on the other hand, using the proposed scheme C2, C3, and C4, the hit rate improvement ranges from 3.43% to 21.36%. All of the improvements depend on the trace files and cache size.

Figure 4.7 gives us other clear plots on the improvements from different aspects. We observed the hit rates increase with the increasing cache size in Figure 4.7(a); by scheme C2, the increment grows from 90.48% for a 512 cache to 99.13% for a 4K cache on trace X; the amount of the increment is minished with the increasing cache size; and the amount is also depending on the different trace files. We observed the curve of trace Z is a little flat, while others are bending. Figure 4.7(b) shows that the significant effect of different compaction schemes on hit rate; for trace W, the hit rate improves rapidly by up to 17.85%, especially by scheme C2, C3 and C4. Figure 4.7(c) depicts the effect of compaction schemes on different IPv4 traces with a given 1K cache. All the traces share the similar increasing trend of hit rate, although the amounts of increment are different.

### 4.4.1.3 AMAT speedups

To compare the impact of the proposed compaction schemes on the average memory access time ( in Chapter 2), we still use the speedup as a measurement. The speed up here is the ratio of the AMAT by original tables and the tables after compacted, which is shown in equation 4.1. Assume routing table access time is L times larger than cache access time.

$$Speedup = \frac{AMAT_{original}}{AMAT_{Compacted}} \qquad \text{(Equation 4.1)}$$

For example, Table 4.4 depicts AMAT speedup gained by scheme C3 over the original tables with a 1K cache for all traces. We observed scheme C3 expedites the average memory access time ranging from 1.11 to 2.49, depending on the value of L and the traces used.

Table 4.4: Scheme C3 speedup with a 1K cache.

| L | Trace file | | | |
|---|---|---|---|---|
| | W | X | Y | Z |
| 4 | 1.35 | 1.41 | 1.34 | 1.11 |
| 8 | 1.74 | 1.85 | 1.72 | 1.24 |
| 16 | 2.32 | 2.49 | 2.31 | 1.47 |

### 4.4.2 IPv6 performance evaluation

We extend all of compaction schemes and implement them on the next generation protocol IPv6, which has 128-bit address length format. Because of the long address length, IPv6 routing table required more memory to store routing entries than IPv4 under the same condition. That in turn indicates the reduction of table entries is significative to save memory and speed up the lookup time.

#### 4.4.2.1 Compaction ratio

We use two IPv6 routing tables to complete the compaction process. Table 4.5 shows the impact of the four schemes on these tables. By compared with the original tables, the compaction ratio of scheme C1 is almost as low as 55%. The other three schemes have much lower CR, which ranges from 30.77% to 36.34%. The test results are nearly consistent to the compaction performance of IPv4 routing tables above. Figure 4.8 depicts the change of routing table entries by these compaction schemes.

Table 4.5: The four compaction schemes (CS) and their impact on compaction ratio (CR).

| CS | RT1 | | RT2 | |
|---|---|---|---|---|
| | **No. of Entries** | **CR (%)** | **No. of Entries** | **CR (%)** |
| O | 3685 | 100.00 | 3974 | 100.00 |
| C 1 | 2052 | 55.69 | 2325 | 58.51 |
| C 2 | 1155 | 31.34 | 1444 | 36.34 |
| C 3 | 1172 | 31.80 | 1444 | 36.34 |
| C 4 | 1134 | 30.77 | 1400 | 35.23 |

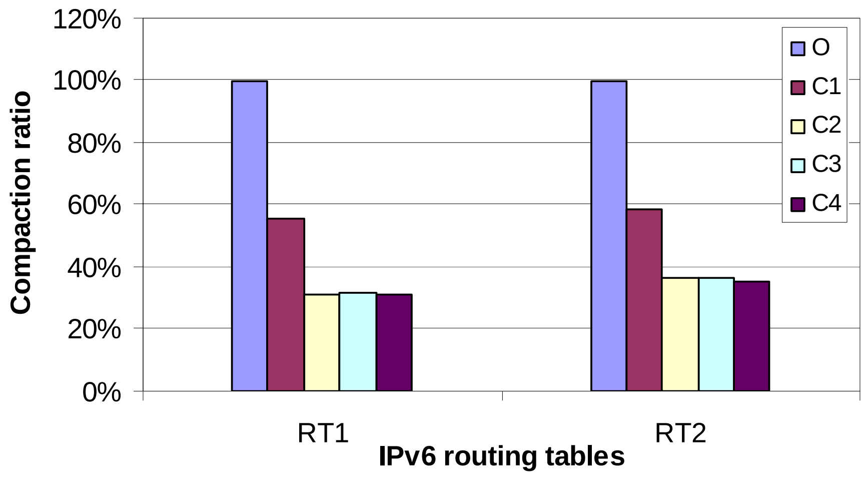


Figure 4.8: Compaction performance of IPv6 routing tables.

### 4.4.2.2 Cache hit rate

There are IPv6 six traces from A to F involved to perform routing lookup operations. The hit rate of different cache size with different compaction schemes is shown on Table 4.6, and Figure 4.9 depicts the compaction performance of 128-entry cache as an example.

Obviously, the cache hit rate is improved rapidly by reducing the number of routing entries. The hit rate increment ranges from 0.41% to 1.40% by applying scheme C1, while it is dramatically increased by scheme C2 from 3.51% to 8.14%. Scheme C3‘s hit rate is increased up to 9.13 %, and the increment of C4 is improved by up to 12.10%. All of the improvements depend on the trace files and the number of cache entries.

Table 4.6: Hit rate with different compaction schemes (CS).

| **IPv6 trace** | **CS** | **64** | **128** | **256** |
|---|---|---|---|---|
| A | O | 70.60% | 81.50% | 89.70% |
| | C1 | 71.51% | 82.72% | 91.10% |
| | C2 | 77.91% | 89.49% | 96.88% |
| | C3 | 79.73% | 90.59% | 97.77% |
| | C4 | 80.63% | 91.92% | 98.46% |
| B | O | 75.70% | 84.53% | 91.18% |
| | C1 | 76.42% | 85.49% | 92.46% |
| | C2 | 83.84% | 92.14% | 97.76% |
| | C3 | 84.69% | 93.28% | 98.62% |
| | C4 | 87.80% | 94.68% | 99.12% |
| C | O | 76.63% | 85.26% | 92.13% |
| | C1 | 77.25% | 86.20% | 93.41% |
| | C2 | 83.40% | 92.56% | 98.18% |
| | C3 | 84.68% | 93.84% | 98.84% |
| | C4 | 87.24% | 95.86% | 99.38% |
| D | O | 80.34% | 88.49% | 93.88% |
| | C1 | 81.13% | 89.37% | 94.67% |
| | C2 | 85.19% | 93.19% | 97.90% |
| | C3 | 85.39% | 93.57% | 98.34% |
| | C4 | 86.93% | 94.77% | 98.72% |
| E | O | 81.64% | 88.96% | 94.14% |
| | C1 | 82.05% | 89.58% | 94.76% |
| | C2 | 86.59% | 93.77% | 97.99% |
| | C3 | 87.18% | 94.28% | 98.43% |
| | C4 | 88.93% | 95.34% | 98.86% |
| F | O | 84.01% | 89.85% | 94.83% |
| | C1 | 84.52% | 90.49% | 95.50% |
| | C2 | 87.93% | 94.39% | 98.34% |
| | C3 | 88.76% | 95.23% | 98.77% |
| | C4 | 90.59% | 96.70% | 99.25% |

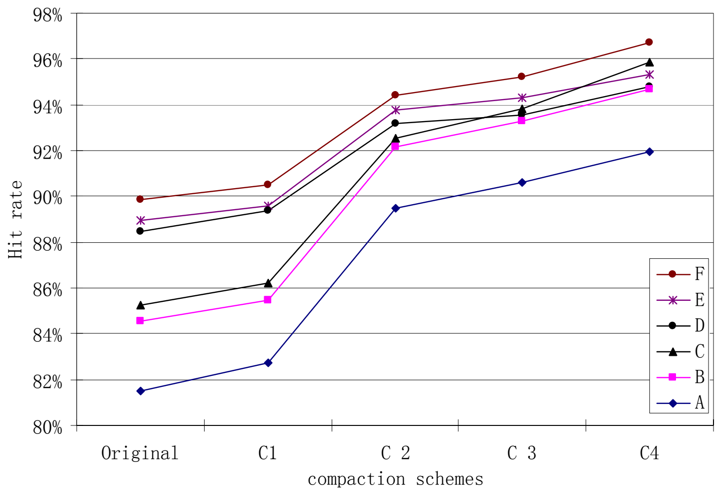


Figure 4.9: Compaction schemes’ hit rates using a 128-entry cache.

#### 4.4.2.3 AMAT Speedups

Table 4.7 shows the AMAT speedup gained by scheme C3 over the original tables with a 128-entry cache for all traces. With regard to IPv6, scheme C3 expedites the average memory access time ranging from 1.13 to 1.67.

Table 4.7: Scheme C3 speedup with a 128-entry cache.

| L | Trace file | | | | | |
|---|---|---|---|---|---|---|
| | A | B | C | D | E | F |
| 4 | 1.21 | 1.22 | 1.22 | 1.13 | 1.14 | 1.14 |
| 8 | 1.38 | 1.42 | 1.42 | 1.25 | 1.27 | 1.28 |
| 16 | 1.57 | 1.65 | 1.67 | 1.39 | 1.43 | 1.47 |

## 4.5 Conclusions and summary

Routing table lookup is an important operation in packet forwarding, especially with the increasing table size caused by Internet traffic. In order to speed up this process, we have proposed four novel compaction schemes for routing table entries and evaluated their impact on a TCAM cache architecture. The extensive simulations using IPv4 and IPv6 routing information have shown that our proposed schemes improve cache hit rate and save the memory usage [41, 42]. The features of our compaction schemes are presented as follows.

- **Four compaction schemes based on TCAM.** The scheme C1 is similar to the compaction technique via Espresso minimization algorithm, but it can be scaled to IPv6 easily. Scheme C2 is an improvement over C1 to obtain higher compaction performance. Scheme C3 and C4 try to provide further compaction with considering the routing conflicts.
- **High compaction performance.** The number of routing entries after compacted by scheme C1 is only almost 65% of the original ones for IPv4, and even low to 55% for IPv6. The other three compaction schemes reduce the number of the entries ranging from 63.66% to 70.55%.
- **High cache hit rate.** The cache hit rate is improved dramatically by performing searches in compacted routing tables, especially those tables compacted by scheme C2, C3 and C4. The hit rate improvements go up to 21.36% for IPv4 traces and to 12.10% for IPv6 traces.
- **High AMAT speedups.** The compaction scheme C3 has speeded up AMAT from 1.11 to 2.49 for IPv4 and 1.13 to 1.67 for IPv6. The average memory access time is decreased because of the higher cache hit rate obtained by reducing routing entries.

The routing table compaction has been shown to improve the caching effectiveness. A good compaction scheme is beneficial to increases cache hit rate, while reducing the memory usage. The high cache hit rate makes average memory access time shorter; this in turn speeds up the address lookup process.

# Chapter 5

# Port Errors and Sampling Techniques

## 5.1 Introduction

In the proposed cache organization, a port error occurs when the port selected by the cache doesn't match the port that would be selected by the routing table. There are two of the major causes for port errors are mentioned in [14].

1) *Longest prefix matching mechanism.* The longest prefix matching entry of a destination address in routing table is not being in cache. Because of the limited size of cache, only small numbers of routing entries are stored in cache. Hence, it is possible that a matching entry found in the cache has a different port assignment with the one found in routing table.

2) *Cache coherence problem.* Some changes made in routing table are not updated immediately in cache. Consequently, the entries in cache do not keep consistence with the entries in the routing table any more. This kind of port error can be reduced by updating the cache and the routing table at the same time, or flushing the cache periodically.

In our research, we take into account the port errors generated by the first source of port errors. Our compaction schemes (Chapter 4) combines many routing entries into one using the *don't care* element in Ternary CAM. This could in turn extend the time that the entry with a short prefix length or with many *don't care* stays in cache longer. Thus, it increases the probability that the matching entry in the cache could not be the longest prefix matching entry in the routing table. If both of these entries do not share the same port assignment, a port error is caused.

In order to lessen the side-effect of port errors, two novel sampling techniques namely, selective and adaptive sampling, are proposed in this chapter. These techniques are implemented with our compaction schemes to reduce port errors effectively without the damage of the high cache hit rate already obtained.

This chapter is organized as follows. In Section 5.2, a common interval sampling technique to reduce port errors is described. Section 5.3 presents two proposed samplings. Section 5.4 provides comparison, analysis and simulations on the port error control performance with IPv4 and IPv6 routing information. Section 5.5 gives a summary of this chapter.

## 5.2 Interval sampling

A sampling technique present in [14] is one way to alleviate this port error problem. It is independent of the reasons causing the port errors. The main feature of this sampling is its simplicity. It requires performing one search in routing table every *M* lookups, where *M* is the sampling rate. If the port number is found different between the cache and the routing table, the matching entry in the routing table will be written into the cache. This will help to reduce the port error ratio since future references to this entry will be found in cache.

Interval sampling obtains an improvement, as it will be shown in the simulation results in this chapter. However, it required a large number of searches in the routing table and port comparisons. The exact number of these searches depends on the sampling rate M. Below is the equation that helps to determine the number searches.

$$Search\_num \approx \frac{Number\_route\_reqs}{M} + Number\_cache\_miss \times \frac{M-1}{M}$$

Where the total number of route request and cache misses are considered.

This sampling is not appropriate for the control of port errors in our novel compaction schemes. Since the numbers of the port errors occurred in the compaction schemes are probably large, the port error ratios are probably still high using this interval sampling, which are shown in later simulations. Thus, it is necessary to find better solutions.

## 5.3 Proposed sampling techniques

In order to further reduce the number of port errors, especially for our compaction schemes, the contents of the routing entries and port error distributions are analyzed in this section and followed by our two sampling techniques to control port error and decrease the search amount in routing tables at the same time.

### 5.3.1 Port error distribution

Each routing entry has different probability in leading to port errors, which depends on both the incoming trace patterns and the features of this particular entry, such as the temporal locality of traces, the prefix length and the *don't care* elements of routing entries.

For instance, Table 5.1 shows the port error distribution for IPv6 traces in the compacted routing tables by scheme C3. The value of N means how many entries in this routing table causing routing conflicts for a particular entry in the same table. The value in the table depicts the sum of the port errors caused by the group of the entries with the same N.  Figure 5.1 illustrates the trend of this distribution, which actually reflects the typical port error distribution for scheme C3 and C4.  We observed those entries with the value of N less than 10 cause most of port errors. However, the interval sampling doesn't consider this issue and executes routing table search uniformly. Consequently, there are many unnecessary searches for those entries rarely causing port errors.

Table 5.1: Port error distribution.

| **N** | **IPv6 Trace** | | | | | |
|---|---|---|---|---|---|---|
| | **A** | **B** | **C** | **D** | **E** | **F** |
| 0 | 47 | 1 | 32 | 7 | 1 | 1 |
| 1 | 7685 | 29950 | 13597 | 29993 | 4843 | 15912 |
| 2 | 11402 | 4067 | 1574 | 3415 | 1348 | 4454 |
| 3 | 774 | 425 | 701 | 34071 | 3151 | 588 |
| 4 | 1268 | 1 | 0 | 23 | 4 | 132 |
| 5 | 92 | 0 | 0 | 296 | 29 | 461 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 2 | 0 | 0 | 1 | 5 | 0 |
| 8 | 39 | 0 | 0 | 4 | 0 | 3 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 0 | 1 | 0 | 35 | 55 | 0 |
| 15~33 | 0 | 0 | 0 | 0 | 0 | 0 |
| 34 | 0 | 0 | 0 | 30 | 0 | 0 |

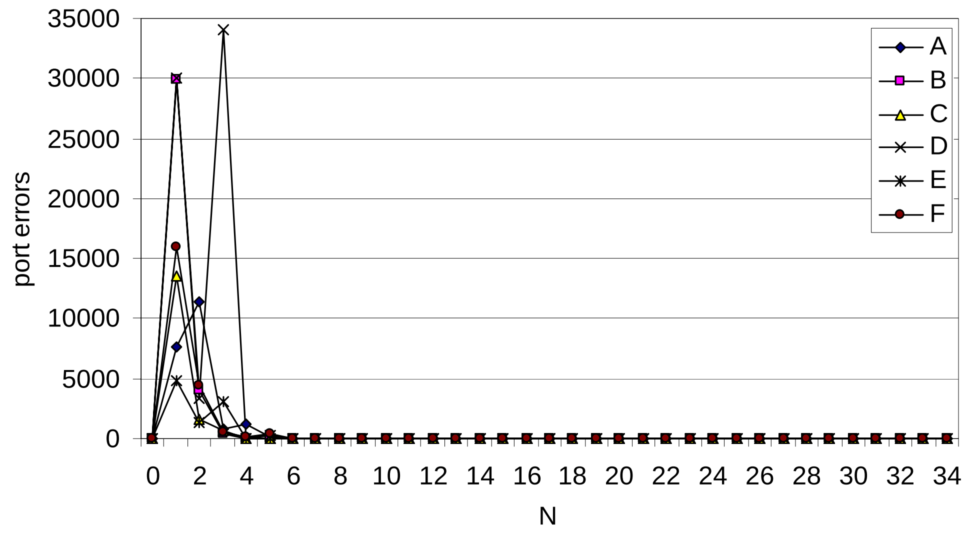


Figure 5.1: The port error distribution with compacted tables by Scheme C3.

Based on this distribution that is common for IPv6 and IPv4 traces, we developed two new sampling schemes, namely selective and adaptive sampling. These schemes help to not only reduce the number of searches in routing tables but also effectively control port error ratio in the C3 and C4 schemes. We should point out that both of these sampling techniques are suitable for C1 and C2 schemes, even though they already have a small port error ratio, and it is not necessary to implement these two advanced sampling. Their port error distributions can be considered as special cases that the values of N is always 0.

### 5.3.2 Selective sampling

This scheme focuses on only executing sampling on those entries with high probability of causing port errors. The purpose is to decrease the number of searches in routing table without increasing port error ratio. The selective sampling scheme works as follows (this is also shown in Figure 5.2 in a flow chart form):

- Label those entries with high probability of causing port errors according to the port error distribution;
- Do not search the routing table in the case of matching entry unlabeled;
- Otherwise, search routing table at the sampling rate and update cache if a port error occurs.

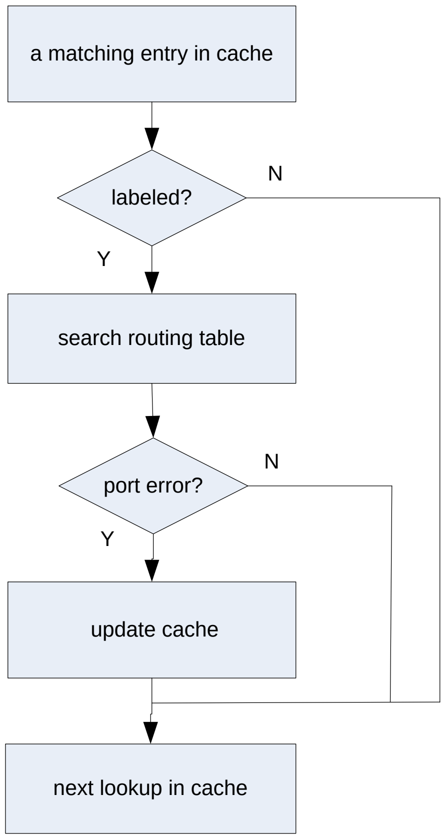


Figure 5.2: Flow chart of selective sampling.

### 5.3.3 Adaptive sampling

The main motivation for adaptive sampling is to adjust the sampling rate based on the previous activities of a particular entry. The same as the selective sampling, we take into account those routing entries with some probability of causing port errors. If it causes port errors frequently before, then set a low sampling rate; vice versa. The basic steps (also shown in Figure 5.3) are as follows.

- At the beginning, the sampling rate of each entry is set as a small value, such as 0.
- If no port error occurs when searching the entry in the routing table, and potentially there is small possibility that port error occurs in a short while due to packet stream’s temporal locality. Consequently, its sampling rate is increased by 1 and the interval between two searches of the same entry in routing table is extended to a longer period.
- If an entry found in cache is not searched in the routing table, its sampling rate is decreased by 1, which means the possibility of future port error is increased.

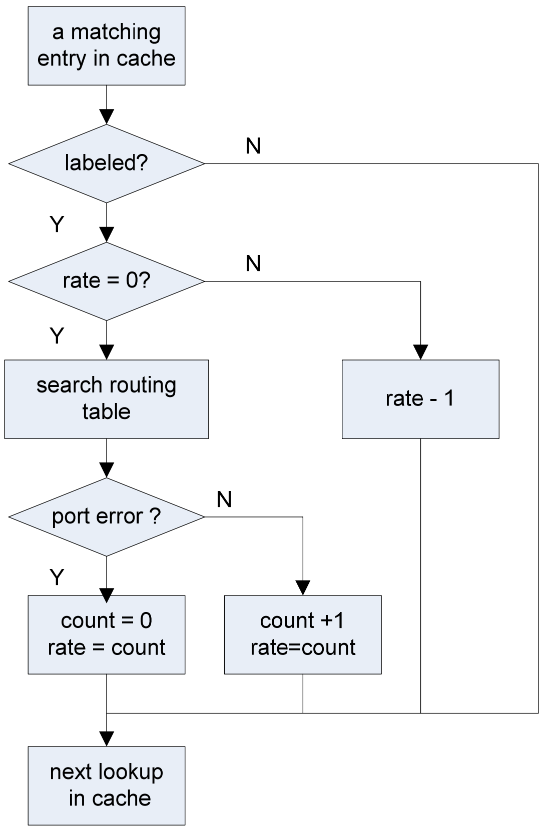


Figure 5.3: Flow chart of adaptive sampling.

Figure 5.3 shows the flow chart to explain this scheme. Those entries with some probability in causing port error are labeled firstly. Due to the temporal locality of traces and the corrections made by adaptive sampling in time; this technique not only reduces the number of searches in routing table, but also decreases the port error effectively.

## 5.4 Performance analysis and evaluation

In this section, we compare the port error reduction for both IPv4 and IPv6 routing. We analyze the samplings' effectivity on the amount of searches in routing table and the value of cache hit rate.

### 5.4.1 Port error ratio

We first choose port error ratio as one of the measurements to analyze the port error problems. Port error ratio is the ratio of the number of port errors to the number of cache hits.

Table 5.2: IPv4 port error ratio using interval sampling with 1K-entry cache.

| CS | Sampling | IPv4 Traces | | | | Average |
|---|---|---|---|---|---|---|
| | | W | X | Y | Z | |
| O | w/o | 0.000% | 0.002% | 0.002% | 0.001% | 0.001% |
| | w/ | 0.000% | 0.002% | 0.001% | 0.001% | 0.001% |
| C1 | w/o | 0.001% | 0.003% | 0.003% | 0.002% | 0.002% |
| | w/ | 0.001% | 0.002% | 0.002% | 0.001% | 0.002% |
| C2 | w/o | 0.783% | 1.215% | 1.882% | 0.914% | 1.199% |
| | w/ | 0.239% | 0.321% | 0.255% | 0.097% | 0.228% |
| C3 | w/o | 0.763% | 1.235% | 1.314% | 0.894% | 1.052% |
| | w/ | 0.248% | 0.337% | 0.266% | 0.098% | 0.237% |
| C4 | w/o | 0.894% | 1.177% | 1.730% | 0.905% | 1.177% |
| | w/ | 0.244% | 0.320% | 0.264% | 0.094% | 0.231% |

Table 5.3: IPv6 port error ratio using interval sampling with 128-entry cache.

| CS | Sampling | IPv6 Traces | | | | | | Average |
|---|---|---|---|---|---|---|---|---|
| | | A | B | C | D | E | F | |
| O | w/o | 0.052% | 0.006% | 0.045% | 0.032% | 0.033% | 0.000% | 0.028% |
| | w/ | 0.024% | 0.003% | 0.008% | 0.014% | 0.017% | 0.000% | 0.011% |
| C1 | w/o | 0.479% | 0.034% | 0.076% | 0.037% | 0.102% | 0.005% | 0.122% |
| | w/ | 0.047% | 0.025% | 0.028% | 0.018% | 0.038% | 0.001% | 0.026% |
| C2 | w/o | 0.392% | 0.131% | 0.107% | 0.067% | 0.212% | 0.187% | 0.183% |
| | w/ | 0.096% | 0.059% | 0.032% | 0.031% | 0.044% | 0.059% | 0.054% |
| C3 | w/o | 3.115% | 2.450% | 1.714% | 6.552% | 1.050% | 1.847% | 2.788% |
| | w/ | 0.604% | 1.026% | 0.338% | 0.423% | 0.354% | 0.539% | 0.547% |
| C4 | w/o | 2.394% | 3.640% | 2.888% | 4.869% | 1.723% | 3.418% | 3.155% |
| | w/ | 1.142% | 1.051% | 0.917% | 0.593% | 0.515% | 0.839% | 0.843% |

Tables 5.2 and 5.3 depict the performance of the interval sampling (for instance, M=3) implemented on original tables and compacted tables with different compaction schemes. The simulations are executed using IPv4 and IPv6 routing information, respectively.

We observe the port error problem is decreased using the interval sampling for IPv4 routing; actually, this sampling decreases port errors down to almost 20%. Since the port error ratios here are very small after interval sampling, we do not need to consider other improvements. With regard to IPv6, it works well for those compacted tables without routing conflicts, such as the tables compacted by the C1 and C2 schemes. However, this is not appropriate for schemes C3 and C4. Obviously, for some traces, the port error ratios are still over than 1%.  Some routing conflicts are permitted to exist in the routing tables by using these two schemes to achieve high hit rate and good compaction performance. However, if the entry causing routing conflicts stays in the cache longer, the possibility of the port errors is larger.

In order to control the port error problem further, we also develop selective sampling (Sl) and adaptive sampling (Ad) techniques on cache organization, and compare them with an every hit sampling (Eh) that always search the routing table when there is a cache hit. Obviously, this every hit sampling is an ideal situation, which provides the best performance could be achieved.

Table 5.4 shows the improvement of port error ratio by our error-control samplings. Comparing with Table 5.3, it can be observed that the selective sampling gets similar or less port error ratio, while adaptive sampling dramatically decreases the port error ratio. The performance of adaptive sampling is almost as good as the ideal every hit sampling. Figure 5.4 demonstrates a clean trend of the port error ratio’s variety.

Table 5.4: The port error ratio of improved sampling schemes.

| | | **Traces** | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **CS** | **Sampling** | **A** | **B** | **C** | **D** | **E** | **F** | **Average** |
| O | Sl | 0.023% | 0.003% | 0.008% | 0.014% | 0.016% | 0.000% | 0.011% |
| | Ad | 0.015% | 0.002% | 0.006% | 0.008% | 0.015% | 0.000% | 0.008% |
| | Eh | 0.014% | 0.001% | 0.005% | 0.008% | 0.012% | 0.000% | 0.007% |
| C1 | Sl | 0.048% | 0.025% | 0.028% | 0.017% | 0.038% | 0.001% | 0.026% |
| | Ad | 0.028% | 0.015% | 0.015% | 0.011% | 0.024% | 0.001% | 0.016% |
| | Eh | 0.027% | 0.015% | 0.015% | 0.010% | 0.021% | 0.001% | 0.015% |
| C2 | Sl | 0.098% | 0.059% | 0.033% | 0.029% | 0.045% | 0.060% | 0.054% |
| | Ad | 0.063% | 0.033% | 0.017% | 0.018% | 0.030% | 0.032% | 0.032% |
| | Eh | 0.059% | 0.031% | 0.016% | 0.017% | 0.029% | 0.029% | 0.030% |
| C3 | Sl | 0.597% | 1.020% | 0.343% | 0.416% | 0.345% | 0.539% | 0.543% |
| | Ad | 0.347% | 0.541% | 0.186% | 0.278% | 0.232% | 0.356% | 0.323% |
| | Eh | 0.332% | 0.537% | 0.178% | 0.267% | 0.215% | 0.351% | 0.313% |
| C4 | Sl | 1.131% | 1.046% | 0.925% | 0.588% | 0.509% | 0.842% | 0.840% |
| | Ad | 0.736% | 0.607% | 0.906% | 0.477% | 0.340% | 0.806% | 0.645% |
| | Eh | 0.693% | 0.570% | 0.605% | 0.453% | 0.310% | 0.521% | 0.525% |

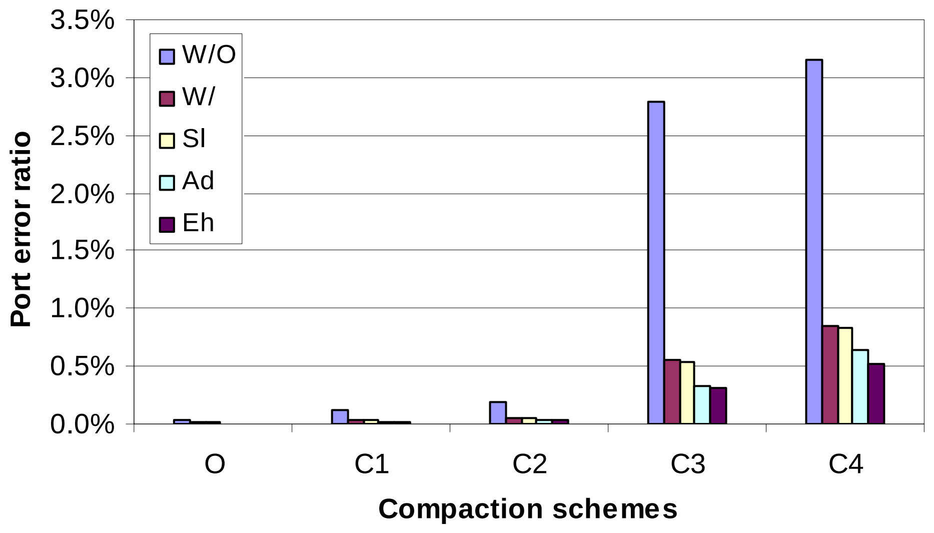


Figure 5.4: The average port error ratios of sampling techniques.

### 5.4.2 Number of searches in routing tables

Given an incoming trace, the number of searches in routing tables is another measurement of sampling technique. As we know, searching routing table is a time-consuming process due to the slow memory access time. The frequent searches in memory increase the complexity and decrease the efficiency of the solution.

Table 5.5 shows the number of searches with improved sampling techniques on a 128-entry cache. Although the selective sampling obtains the similar port error ratio to the interval sampling as mentioned earlier, it only requires smaller amount of searches in routing tables for scheme C3 and C4. The adaptive sampling decreases this searching amount greatly for all of these four compaction schemes, with an effective reduction of port errors at the same time. Figure 5.5 illustrates the average amount of searches with these sampling techniques.

Table 5.5: The number of routing table searches of sampling schemes.

| **CS** | **SA** | **IPv6 Traces** | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | **A** | **B** | **C** | **D** | **E** | **F** | **Average** |
| O | interval | 913,445 | 872,981 | 863,270 | 820,180 | 813,926 | 801,972 | 847,629 |
| | Sl | 913,452 | 872,982 | 863,263 | 820,178 | 813,906 | 801,972 | 847,626 |
| | Ad | 829,543 | 696,596 | 668,762 | 563,112 | 550,531 | 509,484 | 636,338 |
| | Eh | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 |
| C1 | interval | 897,242 | 860,179 | 850,762 | 808,469 | 805,837 | 793,469 | 835,993 |
| | Sl | 897,252 | 860,184 | 850,771 | 808,467 | 805,818 | 793,470 | 835,994 |
| | Ad | 802,379 | 673,546 | 647,914 | 536,504 | 526,086 | 491,105 | 612,922 |
| | Eh | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 |
| C2 | interval | 806,933 | 771,879 | 765,958 | 757,212 | 750,482 | 741,965 | 765,738 |
| | Sl | 807,170 | 771,656 | 765,960 | 757,935 | 750,350 | 741,807 | 765,813 |
| | Ad | 613,943 | 462,762 | 462,384 | 422,892 | 402,716 | 409,431 | 462,355 |
| | Eh | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 |
| C3 | interval | 795,943 | 763,131 | 750,559 | 754,817 | 745,509 | 734,501 | 757,410 |
| | Sl | 501,873 | 396,689 | 394,088 | 391,593 | 467,866 | 430,141 | 430,375 |
| | Ad | 529,784 | 561,398 | 468,073 | 434,444 | 513,238 | 518,079 | 504,169 |
| | Eh | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 |
| C4 | interval | 781,781 | 746,542 | 729,673 | 740,083 | 733,439 | 720,645 | 742,027 |
| | Sl | 453,377 | 454,024 | 379,182 | 402,487 | 405,725 | 409,210 | 417,334 |
| | Ad | 463,855 | 641,810 | 574,359 | 479,562 | 457,646 | 420,849 | 506,347 |
| | Eh | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 | 2,000,000 |

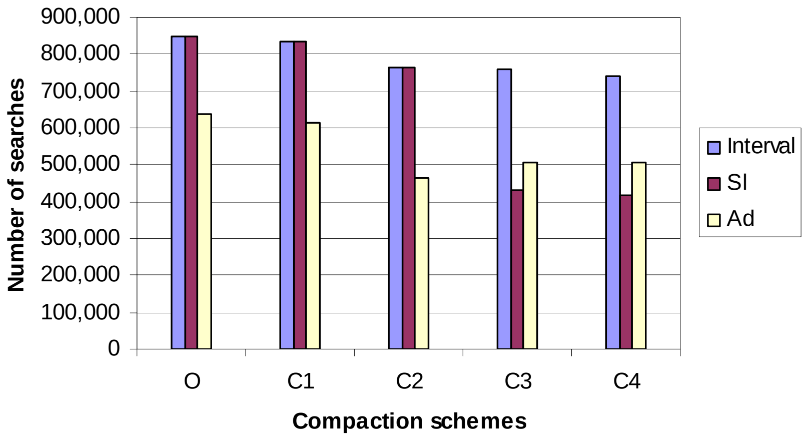


Figure 5.5: The average amount of searches.

### 5.4.3 Cache hit rate

Table 5.6: Cache hit rate of 128-entry cache with sampling techniques

| CS | Sampling | Trace file | | | | | |
|---|---|---|---|---|---|---|---|
| | | A | B | C | D | E | F |
| O | w/o | 81.50% | 84.53% | 85.26% | 88.49% | 88.96% | 89.85% |
| | w/ | 81.49% | 84.53% | 85.25% | 88.49% | 88.96% | 89.85% |
| | Sl | 81.49% | 84.53% | 85.26% | 88.49% | 88.96% | 89.85% |
| | Ad | 81.49% | 84.53% | 85.25% | 88.49% | 88.96% | 89.85% |
| | Eh | 81.49% | 84.53% | 85.25% | 88.49% | 88.95% | 89.85% |
| C1 | w/o | 82.72% | 85.49% | 86.20% | 89.37% | 89.58% | 90.49% |
| | w/ | 82.71% | 85.49% | 86.19% | 89.36% | 89.56% | 90.49% |
| | Sl | 82.71% | 85.49% | 86.19% | 89.37% | 89.56% | 90.49% |
| | Ad | 82.70% | 85.48% | 86.19% | 89.36% | 89.56% | 90.49% |
| | Eh | 82.70% | 85.48% | 86.19% | 89.36% | 89.56% | 90.49% |
| C2 | w/o | 89.49% | 92.14% | 92.56% | 93.19% | 93.77% | 94.39% |
| | w/ | 89.48% | 92.11% | 92.55% | 93.21% | 93.71% | 94.35% |
| | Sl | 89.46% | 92.13% | 92.55% | 93.15% | 93.72% | 94.36% |
| | Ad | 89.44% | 92.11% | 92.55% | 93.19% | 93.70% | 94.35% |
| | Eh | 89.44% | 92.09% | 92.54% | 93.15% | 93.71% | 94.34% |
| C3 | w/o | 90.59% | 93.28% | 93.84% | 93.57% | 94.28% | 95.23% |
| | w/ | 90.30% | 92.77% | 93.71% | 93.39% | 94.09% | 94.91% |
| | Sl | 90.36% | 92.76% | 93.71% | 93.38% | 94.09% | 94.91% |
| | Ad | 90.16% | 92.54% | 93.65% | 93.29% | 94.00% | 94.67% |
| | Eh | 90.26% | 92.53% | 93.65% | 93.27% | 93.99% | 94.66% |
| C4 | w/o | 91.92% | 94.68% | 95.86% | 94.77% | 95.34% | 96.70% |
| | w/ | 91.37% | 94.01% | 95.27% | 94.49% | 94.99% | 95.95% |
| | Sl | 91.34% | 94.01% | 95.27% | 94.49% | 94.99% | 95.94% |
| | Ad | 91.09% | 93.74% | 95.01% | 94.17% | 94.88% | 95.77% |
| | Eh | 90.96% | 93.73% | 94.81% | 94.15% | 94.83% | 95.58% |

Cache hit rate is an important performance metric for cache organizations. It affects the address lookup speed significantly during routing process. From Table 5.6, there are minor changes of hit rate with the implementation of our sampling techniques.

This validates our samplings are help to control port error problem with no significantly degrading the original caching performance.

## 5.5 Summary

Port errors are side-effects of using a cache memory in the manner proposed in this dissertation. These errors occur when the port selected by the cache is different from the port that would have been selected using the routing table directly. Due to the implementation of compacting routing table, this problem could be aggravated. In order to reduce port errors, we have proposed two new sampling techniques that better match the requirements of the new compaction schemes. The simulation results using IPv4 and IPv6 routing information have shown that our sampling can alleviate the port error problem effectively without negatively impacting caching performance [42]. The summary of our sampling schemes is as follows.

- **Two new sampling techniques to reduce port errors.** The selective sampling only executes routing table searches for those entries with high probability of causing port errors with a given sampling rate. The adaptive sampling adjusts the sampling rate automatically based on the particular entry's pervious performance. If it has caused port errors frequently, then set a low rate to permit searching routing table and updating cache in time.
- **Smaller port error ratio**. The samplings we proposed are effective ways to alleviate the port error problems. Given a sample trace A, the port error ratio is decreased from 3.115 % to 0.347 % of C3 by the adaptive sampling and its

performance is very close to the ideal *every hit sampling*. The small port error ratio is extremely beneficial to reduce the possibility of incorrect routing.

- **Few routing table searches.** Both of these advanced sampling schemes decrease the number of searches in routing tables as compared with the common interval sampling. This in turn saves search time and decreases the route lookup complexity.

# Chapter 6

# Set Associative Caching Implementation

## 6.1 Introduction

As we presented in Chapter 4, routing table compaction schemes have demonstrated the potential in improving cache performance for IP routing and forwarding operations. The tables are reduced in size by combining many route entries using the *don't care* element in Ternary CAM. Thus, an entry in the compacted routing table may contain more than one entry of the original table internally. Consequently, such small compacted routing tables can be contained into fewer TCAM chips, which in turn save memory cost and improve route lookup speed by parallel searching.

Each compaction scheme becomes practicability only when it can be implemented effectively based on current conditions. For example, one IPv4 routing and compaction scheme by ternary CAM based on bit – pattern associative array is achieved in [15]. The cache is divided into sets. Each set is associated with a network address prefix value. Since the address length of IPv4 is 32 bit, there are 32 possible prefix values.

We also proposed several novel compaction schemes in Chapter 4. All of them can be implemented based on TCAM in theory as well. However, considering the IP address format, which needs 32(for IPv4) or 128(for IPv6) sets in conventional TCAM caching, we take advantage of the features in routing tables after compacted, and propose new hardware implementations for these compaction schemes in this chapter. These implementations are based on simple set associative caching. Using the simple associative TCAM caching architecture associated with our compaction schemes, the system achieves similar or even higher hit rate without overhead cost in hardware, as compared with complex hardware needed for fully associative caches. This high hit rate in turn keeps fast address lookups.

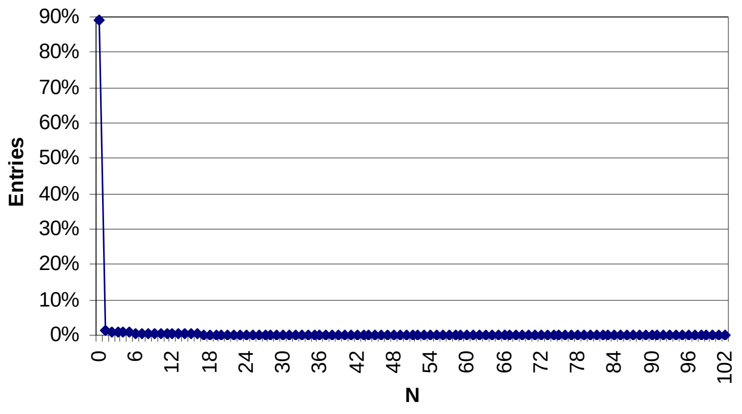
This chapter is organized as follows. In Section 6.2, the features of IP address space in routing table contents and cache hit distribution are analyzed and described for both IPv4 and IPv6 protocols. Section 6.3 presents proposed set associative caching techniques based on the above features. Section 6.4 evaluates cache performance and port error control on these new implementations. Section 6.5 includes a conclusion of this chapter.

## 6.2 Features of address space in routing table entries

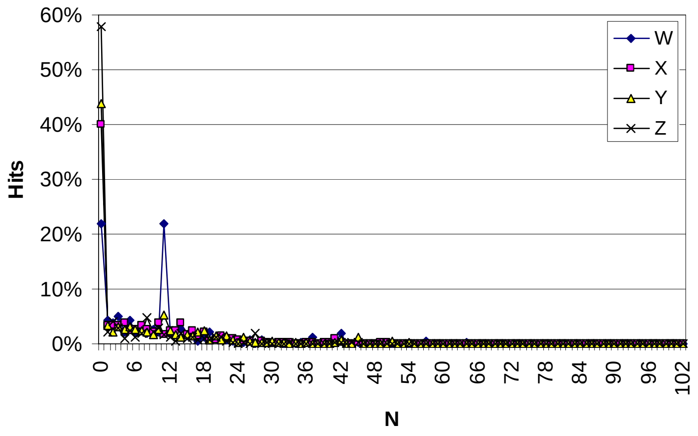
According to the procedure of compaction schemes, we notice that there exists such an entry that has a destination address space overlapping other entries' space with different port assignments in the compacted routing table. For example, entry *a* with the address of *10**11* and entry *b* with the address of *101*11* overlap their address space by *don't care* elements in TCAM, if the port numbers associated with them are different. In this section, we extract the features of this kind of address overlapping, which provides the motivation for the hardware implementation later. In order to describe concisely, we choose the compaction scheme C2 as an example to make analysis and explanation.

### 6.2.1 IPv4 routing table

In order to assess the address overlapping in compacted tables, we count the number of overlapping entries (N) that are covered by a particular entry. For the above example, entry *b* is one entry covered by entry *a*. The number of such entries covered by entry *a* might be more than 1.



(a) Number of routing entries with overlapping entries.



(b) Hit distribution with compacted tables.

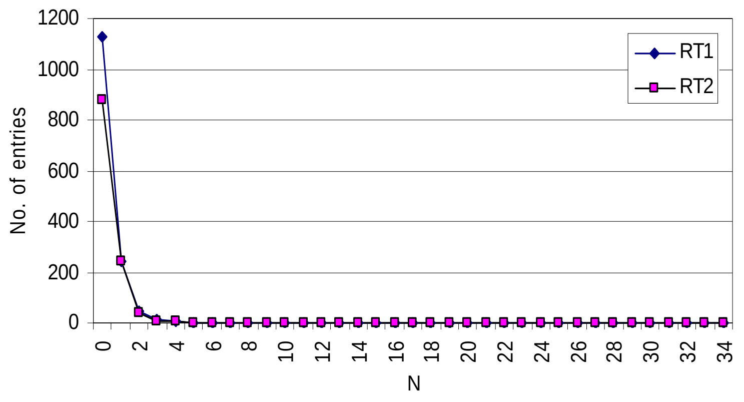
Figure 6.1: Distribution of overlapping entries (IPv4).

Figure 6.1(a) shows the percentage of the entry number in a IPv4 routing table with the same value of N. Obviously there are few entries with overlapping destination address space for N larger than 20. The 1K-entry cache hit distribution of this compacted routing table with scheme C2 is plotted in Figure 6.1 (b). The y-axis depicts the percentage of hits, which is the ratio of the hits among the group entries with the same number of N to the total hits. Almost 60% or even more hits are contributed by the entries with N less than 20. Thus, it provides the possibility that we can treat the routing entries differently by their values of overlapping entries N, when fetching them in cache. For those entries have no much contribution to cache hits, we can assign few cache space with less priority to them, while assigning more cache space with high priority to those significant entries. This flexible assignment will not degrade the whole system performance.

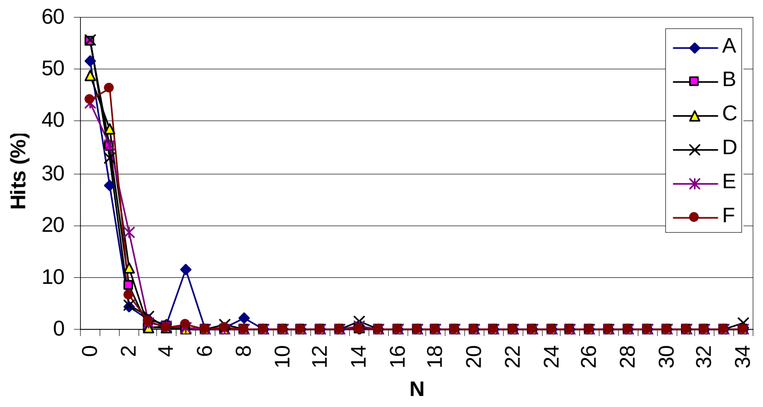
### 6.2.2 IPv6 routing table

Because of IPv6's long address format, it requires more memory to store the large routing table, which makes the compaction schemes much more important in saving memory usage and improving the lookup performance. It also brings a challenge in hardware implementation of compaction schemes. Similarly to the above procedure of IPv4, we analyze IPv6 address overlapping status to simplify the implementation in hardware.

Figure 6.2 depicts the feature of overlapping address and cache hit distribution of a 128-entry cache. There are few entries with overlapping destination address space for N larger than 3. A great number of the cache hits concentrate onto the entries with small number of N, which consists with the trend of IPv4.



(a) Number of routing entries with overlapping entries.



(b) Hit distribution with compacted tables.

Figure 6.2: Distribution of overlapping entries (IPv6).

## 6.3 Set associative caching implementation

In this section, we develop new implementations for our compaction schemes on set associative caching, which are applied to IPv4 and IPv6, respectively. A set-associative cache is considered as a reasonable compromise between a complex fully associative cache and simplistic direct mapped cache. These implementations are motivated by the features of overlapping addresses and cache hit distribution in compacted routing tables.

For IPv4 routing, we have observed those entries with the value of N less than 20 have most of cache hits from Figure 6.1. Thus, the cache is divided into 21 sets. The entries in routing tables with the different value of N (i.e. N=0, 1, 2, …19, >19) are filled in the corresponding set. The size of each set in the associative cache array is initially made proportional to the distribution of hits in each group. The priority of each set is opposite to the value of N. Figure 6.3 illustrates our scheme. When the destination address searched is found a match or multi matches in cache, the entry with higher priority is chosen and the corresponding port number is selected to deliver the packet. If there is a miss in cache, an entry found in the routing table memory is needed to write into cache. First which set the entry should be written is decided according to its number of overlapping entries N, then fill the entry into free space in cache or replace some other entry in the same set by Least Recently Used replacement policy.

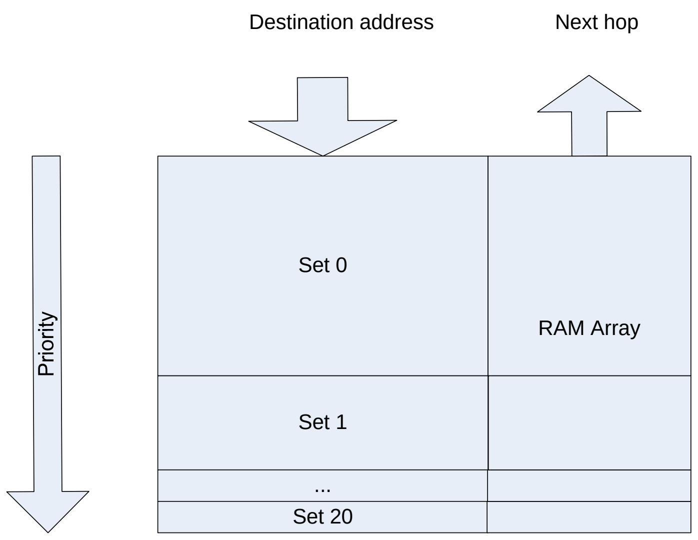


Figure 6.3: Set associative TCAM caching for IPv4.

With regard to IPv6 routing, those entries with the value of N less than 3 contribute a large number of hits from Figure 6.2. Thus, we only need 4-set associative caching. The entries in routing tables with the different value of N (i.e. N=0, 1, 2, >2) are filled in the corresponding set. Figure 6.4 is the diagram of this implementation.

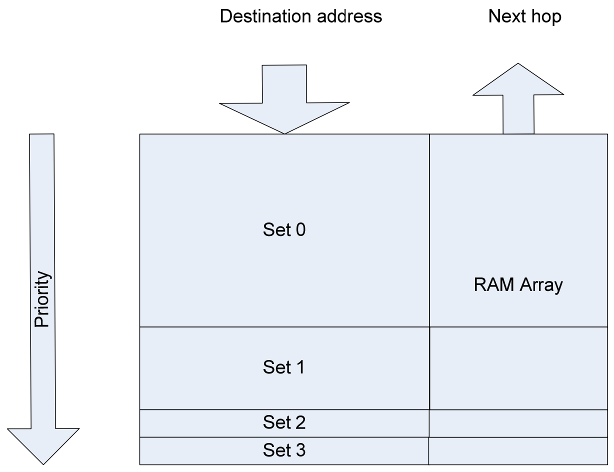


Figure 6.4: 4-set associative TCAM caching for IPv6.

## 6.4 Performance analysis and evaluation

In this section, our proposed set associative caching techniques are implemented and tested based on TCAM. We use IPv4 and IPv6 destination traces to analyze the address lookup performance. Both cache hit rate and port error ratio are still used as the measurement criterions in our evaluation.

### 6.4.1 Cache hit rate of IPv4 routing

The cache is initially divided into 21 sets with the size proportional to the distribution of cache hits in each group N. The implementation of set associative cache results in a little bit of change of the hit rate, as compared with fully associative cache.

The hit rate of a 1K-entry cache with these two kinds of caching is shown on Table 6.1. Obviously, the hit rate of our set associative caching is very close to that of the fully associative cache. The difference between them is trivial and less than 0.1%. However, set associative caching is beneficial to build simpler and less expensive architecture. All of the performances depend on the trace files and the number of cache entries.

Table 6.1: Hit rate of 1K-entry cache.

| Cache | Traces | | | |
|---|---|---|---|---|
| | W | X | Y | Z |
| Fully | 96.99% | 96.68% | 97.53% | 98.67% |
| Set | 96.98% | 96.63% | 97.47% | 98.70% |

### 6.4.2 Port error ratio and sampling schemes for IPv4 routing

As we mentioned before, a port error caused by caching is a disadvantage to route lookups. It occurs when the port selected using the cache does not match the port that would be selected using the routing table. Interval sampling technique is usually used to alleviate this problem. Table 6.2 and Figure 6.5 depict the difference of port error ratio between without and with sampling. The port error ratio of set associative caching is a little higher than that of full associative caching without sampling technique. However, the average difference is small and less than 0.03%. Moreover, the port error ratio is greatly alleviated down to less than 0.3% with sampling. The difference is cut down to a trivial value as well. Since the port error ratios here are already small after interval sampling, we do not need to analyze other two advanced sampling schemes, namely selective, adaptive sampling, which we have been proposed and compared with every hit sampling in Chapter 5.

Table 6.2: The port error ratio without (w/o) and with (w/) sampling (SA).

| Cache | SA | Trace file | | | | Average |
|---|---|---|---|---|---|---|
| | | W | X | Y | Z | |
| FU | w/o | 0.783% | 1.215% | 1.882% | 0.914% | 1.199% |
| | w/ | 0.239% | 0.321% | 0.255% | 0.097% | 0.228% |
| Set | w/o | 0.823% | 1.228% | 1.929% | 0.925% | 1.226% |
| | w/ | 0.241% | 0.308% | 0.264% | 0.095% | 0.227% |

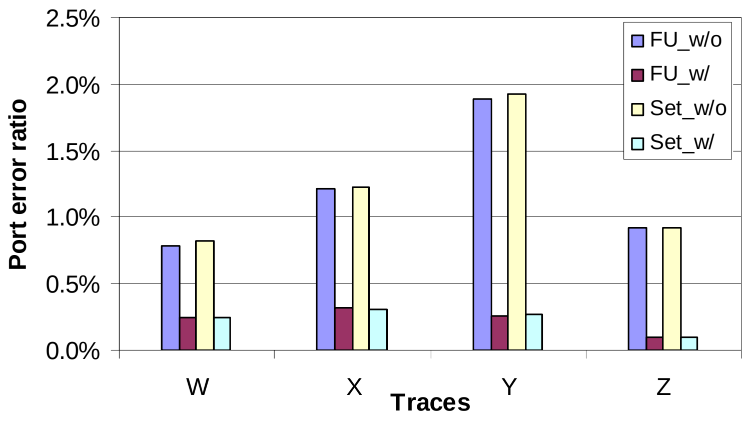


Figure 6.5: The performance of port error ratio.

### 6.4.3 Cache hit rate of IPv6 routing

According to the features of IPv6 routing table, we implement 4-set associative caching based on TCAM. Since the size of each set is required to be proportional to the distribution of hits in each group, three different divisions of four sets based on Figure 6.2 (b) are analyzed and compared. The following Table 6.3 shows the information about these set divisions.

The different divided sets might result in the varying hit rate in the cache architecture. The hit rate of a 128-entry cache with different set divisions and fully associative caching is shown on Table 6.4 and plotted in Figure 6.6. We use the six groups of traces to evaluate the address lookup performance. There is minor change of cache hit rate between the set associative caching and fully associative caching, especially for division D2 and D3. The difference between them is less than 0.5%.

Table 6.3: Set divisions (Consider the total cache as one).

| N | Set divisions | | |
|---|---|---|---|
| | D1 | D2 | D3 |
| 0(Set 0) | 0~0.50 | 0~0.55 | 0~0.55 |
| 1(Set 1) | 0.51~0.80 | 0.56~0.90 | 0.56~0.88 |
| 2(Set 2) | 0.81~0.85 | 0.91~0.95 | 0.89~0.95 |
| >2(Set 3) | 0.86~1 | 0.96~1 | 0.96~1 |

Table 6.4: Hit rate (%) of 128-entry cache with different set divisions.

| Divisions | Traces | | | | | |
|---|---|---|---|---|---|---|
| | A | B | C | D | E | F |
| FU | 90.59 | 93.28 | 93.84 | 93.57 | 94.28 | 95.23 |
| D1 | 90.35 | 91.93 | 92.01 | 93.1 | 93.93 | 94.45 |
| D2 | 90.6 | 93 | 93.31 | 93.31 | 94.12 | 95.17 |
| D3 | 90.73 | 93.15 | 93.53 | 93.43 | 94.28 | 95.2 |

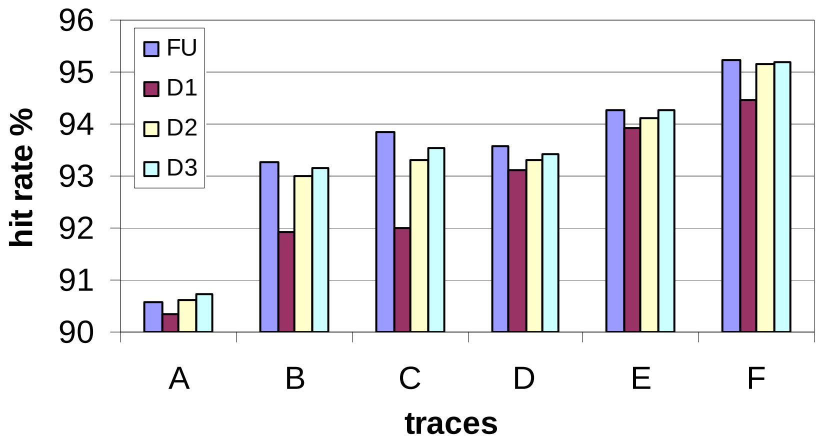


Figure 6.6: Different set divisions' hit rate using 128-entry cache.

### 6.4.4 Port error ratio and sampling schemes for IPv6 routing

Table 6.5 provides the port error ratio without and with sampling techniques. By interval sampling, the port error ratio is alleviated down to less than 1%, almost 0.5% for most cases. With advanced samplings, the port error ratio is decreased greatly and even close to every hit sampling, which is the ideal sampling in theory.

Table 6.5: The port error ratio (%) without (w/o) and with (w/) sampling (SA).

| Divisions | SA | Trace file | | | | | | Average |
|---|---|---|---|---|---|---|---|---|
| | | A | B | C | D | E | F | |
| FU | w/o | 3.12 | 2.45 | 1.71 | 6.55 | 1.05 | 1.85 | 2.79 |
| | w/ | 0.6 | 1.03 | 0.34 | 0.42 | 0.35 | 0.54 | 0.55 |
| | Sl | 0.6 | 1.02 | 0.34 | 0.42 | 0.34 | 0.54 | 0.54 |
| | Ad | 0.35 | 0.54 | 0.19 | 0.28 | 0.23 | 0.36 | 0.33 |
| | Eh | 0.33 | 0.54 | 0.18 | 0.27 | 0.22 | 0.35 | 0.32 |
| D1 | w/o | 2.57 | 2.47 | 1.89 | 6.92 | 1.67 | 1.32 | 2.81 |
| | w/ | 0.46 | 0.53 | 0.23 | 0.34 | 0.23 | 0.26 | 0.34 |
| | Sl | 0.47 | 0.52 | 0.24 | 0.34 | 0.23 | 0.26 | 0.34 |
| | Ad | 0.27 | 0.26 | 0.14 | 0.21 | 0.16 | 0.15 | 0.20 |
| | Eh | 0.25 | 0.25 | 0.13 | 0.2 | 0.14 | 0.14 | 0.19 |
| D2 | w/o | 2.35 | 2.58 | 1.86 | 7.39 | 0.99 | 1.5 | 2.78 |
| | w/ | 0.44 | 0.54 | 0.25 | 0.33 | 0.31 | 0.34 | 0.37 |
| | Sl | 0.44 | 0.52 | 0.25 | 0.33 | 0.31 | 0.34 | 0.37 |
| | Ad | 0.26 | 0.27 | 0.15 | 0.2 | 0.2 | 0.18 | 0.21 |
| | Eh | 0.24 | 0.26 | 0.14 | 0.18 | 0.19 | 0.17 | 0.20 |
| D3 | w/o | 2.93 | 2.6 | 1.81 | 7.2 | 0.96 | 1.79 | 2.88 |
| | w/ | 0.42 | 0.55 | 0.26 | 0.35 | 0.31 | 0.35 | 0.37 |
| | Sl | 0.43 | 0.54 | 0.26 | 0.34 | 0.3 | 0.34 | 0.37 |
| | Ad | 0.25 | 0.27 | 0.15 | 0.21 | 0.2 | 0.18 | 0.21 |
| | Eh | 0.24 | 0.27 | 0.14 | 0.2 | 0.18 | 0.18 | 0.20 |

Given an example of Trace A, the port error ratio of set division D2 and D3 is almost the same and both of them less than fully associative cache (See Figure 6.7). The other traces share the similar trend. Furthermore, division D3 obtains the best performance among these three different divisions by considering both cache hit rate and port error ratio.

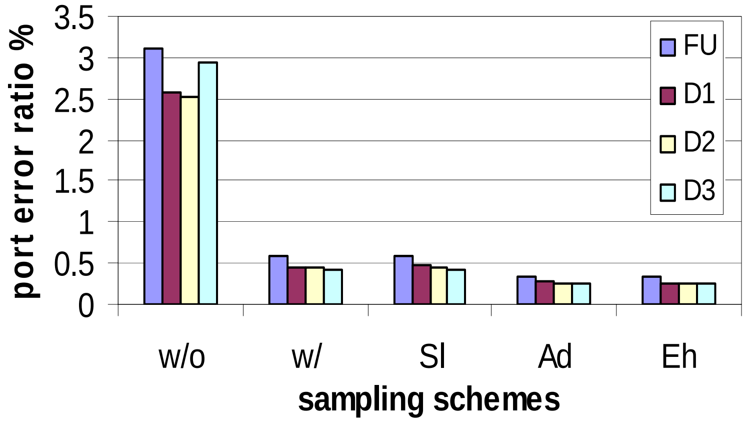


Figure 6.7: The set associative caching improvement on Trace A.

## 6.5 Summary

This chapter presented new set associative caching to implement the compaction schemes proposed in Chapter 4. This set associative caching is achieved based on the practical routing table address space and cache hit distribution. As compared with fully associative cache, it has advantages of simple and inexpensive implementation. We have evaluated different set associative caching architectures with sampling techniques, which are used to alleviate port error problems caused by the cache technique. The simulations on IP routing information have shown that our set associative caching achieves the similar cache hit rate as the fully associative caching and can control port error effectively without impacting the system performance. The summary of our conclusion is as follows.

- **Address space overlapping.** One feature of the compacted routing entries is the address space overlapping. Different overlapped addresses have different contribution to the cache hits. A great number of hits are obtained by those entries with the count of overlapped address less than a value, which depends on the IP routing tables. This value also determines the number of set needed in set associative caching.
- **Set associative caching.** In order to implement the compaction schemes under practical and economical conditions, a cache is divided into sets according to the count of overlapped addresses, instead of fully associative caching. The size of each set is initially made proportional to the cache hit distribution.
- **Keep high cache hit rate.** The cache hit rate achieved by set associative caching is close to that of fully associative cache, either for IPv4 or IPv6. The difference between these two kinds of caching is trivial for most of cases. This set associative caching does not have much damage on the cache performance.
- **Smaller port error ratio.** The interval sampling and the advanced samplings are work effectively to alleviate the port error problems in the set associative caching. The port error ratio is decreased less than 0.5%. There is no much difference on port error ratio between set associative caching and fully associative caching. Small number of port errors help to keep the routing process correct.

# Chapter 7

# Concluding Remarks

IP routing is an important task in the transmission of packets through the Internet. It determines which output port the incoming packet should be delivered to by searching routing tables. The speed of this process is extremely important, especially, with the increasing growth of link speed in Internet router. A high speed line requires fast route lookup to match in order to enhance the overall performance of network processors. However, the routing table lookup is becoming a time-consuming process because the growing size of the routing table that is stored in main memory with a slow access time. A cache memory is generally used to accelerate this operation. That is, caching recently used route entries and achieving a large cache hit rate yield a short average access time.

This dissertation has proposed high performance cache architectures for IP routing, which involves the following areas:

- Improved cache techniques
- Optimized cache replacement policies
- Route entry compaction based on caching
- Advanced sampling techniques for port errors
- Set associative caching implementation

The remainder of this chapter will address the contributions in each of these arrears and discuss the research directions for future work.

## 7.1 Contributions

The research includes several schemes and techniques to enhance IP routing process. They are implemented and estimated though extensive simulation using IPv4/IPv6 routing information. The features of our work are presented as follows.

- **Two new replacement policies: LAR and RLAI.** Both of the policies tend to remove an inactive entry by evaluating its previous access references. The LAR evicts the entry with the smallest access count among N entries that have not been accessed for a long while. As for RLAI, if an entry has not been referred longer than its previous access interval, it can be potentially evicted. Due to the higher cache hit rate yielded by the LAR and RLAI policies, fewer accesses to routing table memory are needed for address lookups. Consequently, this helps to reduce the average memory access time, which is often used as measurement for cache-memory performance.
- **Four compaction schemes based on TCAM.** They obtain high compaction performance. The number of routing entries after compacted is between 30% to 65% of the original ones. The cache hit rate is improved dramatically by performing searches in compacted routing tables.
- **Two new sampling techniques to reduce port errors.** The selective sampling only executes routing table searches for those entries with high probability of causing port errors with a given sampling rate. The adaptive sampling adjusts the

sampling rate automatically based on the particular entry's pervious performance. They lower the port error, which is beneficial to reduce the possibility of incorrect routing.

- **Set associative caching.** In order to implement the compaction schemes under practical and economical conditions, a cache is divided into sets according to the count of overlapped addresses. The size of each set is initially made proportional to the cache hit distribution. This scheme keeps the high cache hit rate close to that of fully associative cache.
- **Victim caching and randomly selected indexing:** The VC increments cache hit rate by reducing miss penalty. A simple direct-mapped cache cooperating with a 16-entry victim cache can achieve a hit rate better than 2-way or 4-way set associative cache of the same size. Randomly selected indexing scheme reduces conflict misses by searching the entries prone to conflict in direct-mapped cache more than one place.
- **Pipelining:** A high route lookup throughput is achieved by proposed pipeline structure. The route lookup process is divided into three independent stages: index selection, cache access and getting port. The pipeline has no data hazards.

All the proposed scheme and technologies have been shown to improve the caching effectiveness. The high cache hit rate makes average memory access time shorter; this in turn speeds up the address lookup process.

## 7.2 Future work

The following topics are potential works that could be done in future. They would be beneficial for speedup route entry lookup.

- Clustering: Clustering is the process that organizing objects into groups whose members are similar in some way. If we can use clustering to analyze route entries according to the incoming packets, and generate the pattern of route entries with “closer access relationship”, then it could be possible to reduce cache misses by pre-fetching route entries, at the same time when fetch the real missed entries from routing table to cache. The entry pre-fetched should be the same group as the missed one.
- IPv6 efficient routing: one of the advantages of IPv6 is its hierarchical provider-based global unicast address architecture, which has potential to allow efficient routing. If we can decompose the large size tables into independent small ones based on the multi-level of IPv6 addressing and make each route lookup in hierarch method, then it is not necessary to search the whole IP address, which in turn saves searching time. Furthermore, small tables are benefit to speedup lookup process too.
- Warm Start Cache: The above cache architectures begin from a “cold start”. This is, the cache is initialized empty. This method does not maintain any previous forwarding information. By contrary, warm start means saving some relative entries into cache before lookups. If we know a database of those route entries with high access frequency in a long-time, and then storing such entries in cache as “warm start” might be helpful to reduce the number of misses.

- More routing information resource: Try to find large tables and more traces to optimize our proposed schemes. This might be difficult since these traces are not made public in most cases.

# BIBLIOGRAPHY

[1]A.S. Tanenbaum, *Computer Networks*, 4th Ed, New Jersey: Prentice Hall publishers inc. 2003.

[2]RFC 1180, “A TCP/IP Tutorial,” http://www.ietf.org/rfc/rfc1180.txt, January 1991.

[3]RFC 2821, “Simple Mail Transfer Protocol,” http://www.ietf.org/rfc/rfc2821.txt, April 2001.

[4]RFC 959, “File Transfer Protocol,” http://www.ietf.org/rfc/rfc0959.txt, October 1985.

[5]RFC 791, “Internet Protocol,” http://www.ietf.org/rfc/rfc0791.txt, September 1981.

[6]M. A. Ruiz-Sanchez, E. Biersack, and W. Dabbous, “Survey and taxonomy of IP address lookup algorithms,” *IEEE Network*, vol. 15, pp. 8–23, Mar-Apr. 2001.

[7]RFC 1519, “Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy,” http://www.ietf.org/rfc/rfc1519.txt, September 1993.

[8]RFC 1631, “The IP Network Address Translator (NAT)”, http://www.ietf.org/rfc/rfc1631.txt, May 1994.

[9]C. Huitema, *IPv6: the New Internet Protocol*, 2nd edition, Prentice Hall, 1998.

[10]RFC 2373, “IP Version 6 Addressing Architecture”, http://www.ietf.org/rfc/rfc2373.txt, July 1998.

[11]P. Gupta, *Algorithms for Routing Lookups and Packet Classification*, doctoral dissertation, Dept. Computer Science, Stanford University, 2000.

[12]RFC 2453, “RIP Version 2”, http://www.ietf.org/rfc/rfc2453.txt, November 1998.

[13]RFC 2328, “OSPF Version 2”, http://www.ietf.org/rfc/rfc2328.txt, April 1998.

[14] J. J. Rooney, *An Associative Ternary Cache for IP Routing*, doctoral dissertation, State University of New York at Binghamton, NY 2002.

[15]J. J. Ronney, J. G. Delgado-Frias, and D. H. Summerville, “An associative ternary cache for IP routing,” *IEE Proceedings Computers and Digital Techniques*, vol.151, Issue 6, pp.409-416,November 2004.

[16]H. Chao, “Next generation routers,” *Proceedings of The IEEE*, Vol. 90, No. 9, pp.1518-1558, September 2002.

[17]P. Gupta, S. Lin, and N. Mckeown, “Routing lookups in hardware at memory access speeds”, *IEEE infoCom*, April 1998.

[18]N.Huang, S.Zhao,J.Pan, and C.Su , “A fast IP routing lookup scheme for gigabit switching routers, *Proc. IEEE INFOCOM’99*, 1999.

[19]M. Kobayashi, T. Murase, and A. Kuriyama, “A longest prefix match search engine for multigigabit IP processing,” *Proc. IEEE Int’l Conf. Comm. (ICC 00)*, IEEE Press, pp.1360-1364, 2000.

[20]V. Srinivasan and G. Varghese, “Fast IP lookups using controlled prefix expansion”, *Proc. ACM Sigmetrics*, ACM Press, pp.1-10, 1998.

[21]S.Nilsson and G.Karlsson, “IP - address lookup using LC-tries,” *IEEE Journal on Selected Areas in Communications*, 1999.

[22]V. Ravikumar and R. Mahapatra, “TCAM architecture for IP lookup using prefix properties,” *IEEE Micro*, vol. 24, no. 2, pp. 60-69, Mar-Apr. 2004.

[23]H. Liu, “Routing prefix caching in network processor design”, *Proc. International Conference on Computer Communications and Networks (ICCCN)*, Phoenix, AZ, 2001.

[24]J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd Ed, San Francisco: Morgan Kaufmann publishers inc. 2003.

[25]H.C.Chen and J.S.Chiang, “Design of an adjustable-way set-assoicative cache,” *Proc.2001 IEEE Pacific Rim Conference on Communication, Computers and Signal Processing*, Vol.1,pp.315-318,Aug. 2001.

[26]N. P. Jouppi, “Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers,” *Proc. of the 17th annual intl. Symposium on Computer Architecture*, May 1990.

[27]M.R.Zargham, *Computer architecture: single and parallel system*, Englewood Cliffs, NJ: Prentice-Hall.Inc., 1996.

[28]The Measurement and Analysis on the WIDE Internet (MAWI) Working Group, the web site is: http://tracer.csl.sony.co.jp/mawi/

[29]J. G. Delgado-Frias, and R. Guo, “A cache architecture for IPv6 lookups,” *The 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004)*, July 2004.

[30]R. Guo and J. G. Delgado-Frias, “A pipelined cache architecture for IPv6 lookups,” *International Conference on Communication and Computer Networks (CCN 2004)*, Cambridge, Mass., November, 2004.

[31]D. Feldmeier, “Improving gateway performance with a routing-table cache,” *Proceedings of InfoCom’88*, pp.298-307, March 1988.

[32]B. Talbot, T. Sherwood, and B. Lin, “IP caching for terabit speed routers,” *Proceedings of Globecom*, pp.1565-1569, 1999.

[33]T. C. Chiueh and P. Pradhan, “Cache memory design for network processors,” *Proceedings of High-Performance Computer Architecture,* pp.409-418,January 2000.

[34]R. Guo, J. G. Delgado-Frias, and S.Wong, “Cache replacement policies for IP address lookups,” *International Conference on Circuits, Signals and Systems (CSS 2007)*, July 2007.

[35]University of Oregon Route Views Archive Project. http://archive.routeviews.org/

[36]6TAP router information. http://www.6tap.net/

[37]Hurricane Electric Internet Services. http://lg.he.net/cgi-bin/index.cgi.

[38]J. G. Delgado-Frias, J. Nyathi and S. Tatapudi, “Decoupled dynamic ternary content addressable memories,” *IEEE Transactions on Circuits and Systems, I: Fundamental Theory and Applications,* pp.2139-2147,vol. 52, no. 10, October 2005.

[39]H. Liu, “Routing table compaction in ternary CAM,” *IEEE Micro,* vol. 22, no. 1, January-February, pp.58-64, 2002.

[40]J. G. Delgado-Frias and J. Nyathi, “A high-performance encoder with priority lookahead,” *IEEE Transactions on Circuits and Systems, I: Fundamental Theory and Applications,* vol. 47, no. 9, pp. 1390-1393,September 2000.

[41]R. Guo, J. G. Delgado-Frias, “A novel compaction scheme for routing tables in ternary CAM to enhance cache hit rate,” *International Conference on Communication, Internet and Information Technology (CIIT 2007),* July 2007.

[42]R. Guo, J. G. Delgado-Frias, “Table compaction and sampling schemes to enhance TCAM cache performance in IP routing,” 2007.

[43]T. Jamil, R. Stacpoole, “Cache memories”, *IEEE Potentials,* vol. 19, no.2, pp. 24 – 29, 2000.

[44]W. Shyu, C. Wu, T. Hou, “Efficiency analyses on routing cache replacement algorithms,” *Proceedings of the IEEE ICC '02,* pp. 2232-2236, April 2002.