A Novel Hardware Cache Architecture to Support Layer-Four Packet Classification at Memory Access Speeds

Jun Xu Mukesh Singhal
Department of CIS
The Ohio State University
Columbus, OH 43210
jun,singhal@cis.ohio-state.edu

Joanne Degroat
Department of EE
The Ohio State University
Columbus, OH 43210
degroat@ee.eng.ohio-state.edu

February 11, 1999

Abstract

Existing and emerging layer-4 switching technologies such as firewalling, RSVP, DiffServ, and QoS routing all require packet classification to be performed not only on destination address, but also on source address, port numbers, and protocol as well. This is known as the layer-4 lookup problem. The fastest algorithm available today delivers a throughput of only 1 Million Lookups Per Second (MLPS), far away from 25 to 75 MLPS needed to support a 50 to 150 Gbps layer-4 router. We propose to use layer-4 route caching to speed up the process of layer-4 packet classification and design an implementation-ready hardware cache architecture for this purpose. The design is based on our novel cache management algorithm called near-LRU which best exploits the locality behavior exhibited by the Internet traffic. To better approximate the fully-associative near-LRU algorithm using set-associative hardware, we invent a dynamic set-associative scheme based on our novel statistically independent parallel hashing scheme. The cache architecture achieves a high stable hit ratio above 90 percent and a fast throughput from 22.5 to 75 MLPS at a reasonable cost (700 to 1700 US dollars).

1 Introduction

Existing and emerging layer-4 switching technologies such as packet-filtering firewall, RSVP, differentiated services, QoS routing, VPN, and multicasting all require the forwarding decision to be made not only on destination address, but also on source address, port numbers, and protocol as well. This problem is known as *layer-4 packet classification* or *layer-4 lookup*. The motivation behind layer-4 packet classification and its various applications are well documented in [14] and [30]. Today, exponential growth in demand for Internet bandwidth has driven the throughput of Internet routers to 50 Gbps [20]. To support such a high bandwidth at layer-4, layer-4 packet classification needs to be performed at 25 Million Lookups Per Second (MLPS), assuming an average packet size of 2000. However, the fastest scalable hardware scheme for layer-4 packet classification [14] only delivers 1 million layer-4 lookups per second, which is about 4% of the layer-4 lookup power needed for a 50 Gbps layer-4 router.

As route caching was successful at layer-3 in improving the speed of IP lookup [5, 10, 20], we would like to find out if the same technique can be applied at layer-4. However, layer-4 caching is not looked upon as a promising approach in the layer-4 lookup research community. Researchers are critical or skeptical about its scalability, reliability, and cost-effectiveness[14, 30]. Amid these criticisms and skepticism, we have designed a layer-4 route cache architecture that achieves a high stable hit ratio of 92% with only 0.6% standard deviation. It also meets all the technical challenges listed by the researchers as arguments against layer-4 route caching. Such a high and stable hit ratio is achieved by using our novel cache management algorithm called *near-LRU* that best exploits the *locality behavior* exhibited by the Internet traffic at layer-4 [3, 11, 6]. However, near-LRU is still a fully-associative algorithm which is not scalable to large cache size. Use of relatively inexpensive set-associative hardware to approximate may result in a large amount of *collision misses*. For better approximation at almost the same cost, we have invented a technique called *dynamic set-associative scheme* based on our novel *statistically independent parallel hashing* scheme. This technique cuts the amount of collision misses by 60% to 90%. The cache architecture is built upon an emerging DRAM technology called SLDRAM [9, 26] that achieves the same high throughput as SRAM at the price of DRAM. The throughput of our cache architecture is about 22.5/45 MLPS using 400/800 MHz SLDRAM technology, about 45% of the throughput of the underlying memory technology. Using our proposed *lazy-writeback* scheme, the throughput of the cache architecture is improved to 37.5/75 MLPS, which is able to support 75/150 Gbps layer-4 routers.

This paper is organized as follows. Section 2 formulates the problem of layer-4 packet classification. Section 3 presents the issues and challenges with layer-4 caching. Section 4 proposes the concept of near-LRU cache that our cache architecture tries to approximate. Section 5 presents our cache architecture and Section 6 evaluates its performance. Section 7 presents the simulation results of the cache architecture. Section 8 proposes the lazy-writeback scheme that can dramatically improve the throughput of the cache. Section 9 talks about the denial-of-service attack and section 10 concludes the paper.

2 The Problem of Layer-4 Lookup and Related Works

2.1 The Problem of Layer-4 Lookup

The problem of layer-4 packet classification can be formulated as follows. A forwarding table of a layer-4 router typically consists of a large number (say N) of forwarding rules called *filters*. The schema of the forwarding table consists of d+1 attributes, among them d attributes are used as key for search. They correspond to the packet header fields based upon which the forwarding decision is made. The other attribute contains the forwarding decision that should be applied to the packet, such as access privilege, resource reservation, Type-Of-Service (TOS) assignment, queue assignment, and next-hop(s). Each key attribute of a filter allows three types of matches: exact match, prefix match (e.g., 164.107.60.88 matches 164.107.60.*), or range match (e.g., port 8080 matches the range [1024, 65535]) [30]. When a new packet arrives, its corresponding header fields are matched with the filters using all three types of matches described above if applicable. A packet may match multiple filters and hence the ambiguity. This problem is solved by assigning a priority value to each filter. The objective of layer-4 lookup is to find the matching filter with the highest priority, called *best matching filter*, for an incoming packet [30].

In layer-4 applications like firewalling, RSVP, DiffServ, and QoS routing, the five-tuple < destination-address, source-address, destination-port, source-port, protocol >, is typically used as key to search for the forwarding information. We call this five-tuple a *layer-4 address*. However, in firewalls, the ACK bit of TCP flags also participates in making the forwarding decision. We argue that with the rewriting of forwarding rules, the checking of ACK bit can be combined into the forwarding decision like "pass if ACK bit is on" or "block if ACK bit is off". Such a conversion is always possible because we should be able to tell whether a TCP connection, uniquely identified by a layer-4 address, is allowed only one-way (hence ACK) or both ways. Other layer-4 switching

applications like VPN and multicasting use <destination-address, source-address> as the search key. That problem is a special case of packet classification based on layer-4 address and is much simpler[14, 30]. Due to the lack of space, this paper will focus on the packet classification based on the layer-4 address.

Layer-4 packet classification based on d header fields can be viewed as a d-dimensional *range match* problem [14], which is equivalent to a classical problem in computational geometry called *point location problem*[14]. The *point location problem* is to find the object that a point belongs to among a large number (say N) of d-dimensional objects. The general form of the problem when $d > 3$ does not have a nice algorithmic solution that is low in both time and space complexities. On the one hand, the best algorithm in terms of time complexity requires $O(lgN)$ computation, but needs $O(N^d)$ working space[14]. On the other hand, the best algorithm in terms of space complexity has a memory requirement of $O(N)$, but needs $O(lg^{d-1}N)$ computation time[14]. When N is as large as tens of thousands and d is as large as 5, neither algorithm is realistic[14].

Two papers in SIGCOMM'98 ([14] and [30]) proposed new algorithm solutions to packet classification according to layer-4 address. Using a hardware implementation, Lakshman and Stiliadis' scheme [14] was able to perform a worst case of 1 MLPS against a few thousand filters. Two algorithms, *extended grid of tries* and *crossproducting* were proposed by Srinivasan et al. in [30]. The *extended grid of tries* algorithm handles the special case in which port numbers are either wildcards or an exact number. *Cross-producting* algorithm aims to solve the general layer-4 lookup problem and its throughput with small number of filters (no more than 50) is about 2.47 MLPS. However, the space complexity of this algorithm grows exponentially with the number of filters. Therefore, it is not scalable to a large number of filters. "On-demand cross-producting" was proposed to solve the scalability problem but no serious performance study has been conducted on the worst case as well as on the average case. Their 480 ns average search time and 540 ns worst case search time is based on a questionable assumption that every search hits the cache. Therefore, we find that the Lakshman and Stiliadis' scheme [14] is the only solution for the layer-4 classification that is both fast and scalable.

3 The Caching Approach: Issues and Challenges

It has been shown in the literature that a high degree of *temporal locality* can be observed at the transport layer; that is, the arrival of a packet implies a high probability of the arrival of another

packet with the same layer-4 address in the near future [3, 6, 31]. This locality can be explained by the fact that a network object such as an FTP file or an HTTP is broken into a number of packets with the same layer-4 address for transit. This temporal locality implies that if we save a recently-used lookup result, called a *layer-4 route*, in a cache, there is a high probability that an incoming packet will hit the cache and will be forwarded without a full-fledged lookup.

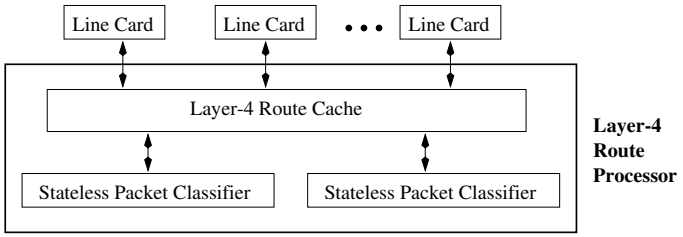

Figure 1: System Model of Our Layer Caching Approach

The system model of a layer-4 router with *route cache* is shown in Fig. 1. The *route processor* of a layer-4 router consists of a *route cache* and a few *stateless packet classifiers*. When a packet arrives at a port of a router, the corresponding line card at that port will extract its layer-4 address and put it in a route lookup request to be sent to the route processor. The route processor will first search the layer-4 address in the route cache for a match. If a hit occurs, the corresponding forwarding decision will be sent back to the line card immediately. If a miss occurs, the packet will be forwarded to stateless packet classifier. The stateless packet classifier will perform a full-fledged layer-4 lookup using the fastest layer-4 packet classification scheme, such as Lakshman and Stiliadis' scheme[14]. The result will be sent to the line card and saved into the route cache. Let $H_{classifier}$ be the total throughput of all *stateless packet classifiers*, R be the miss ratio of the route cache, H_{cache} be the throughput of the route cache, then the lookup throughput of the route processor is $H_{processor}$=$MIN(H_{cache}, \frac{H_{base}}{R})$. This states that a speedup of $\frac{1}{R}$ in layer-4 lookup throughput can be achieved if the throughput of the cache itself does not become a bottleneck. If the miss ratio is as low as 8%, with two 1 MLPS stateless packet classifiers (e.g., Lakshman and Stiliadis' scheme [14]), the lookup throughput of the route processor can be as high as 25 MLPS, enough to support a 50 Gbps layer-4 router.

The only existing work on layer-four caching that was reported in literature is ATLAS [12], an ATM packet filtering firewall working at the rate of OC-3c (155 Mbps). A 128-bit-wide Content Addressable Memory (CAM) is used to cache the recently-used layer-4 addresses that were found

safe. The size of the CAM and the hit ratio achieved are not reported. The problem of this approach is that CAM can not scale to a large size due to technological constraints and is extremely expensive. Commercially available 128-bit-wide CAM comes at the size of 1K entries and costs about $30 [25]. We will show that in a 50 Gbps Layer-4 router, millions of layer-4 routes need to be cached to achieve a decent hit ratio. It is not only that we cannot afford that much CAM, but such a huge CAM is hard to build as well.

In this paper, for studying the miss ratio, the average size of a packet is always considered as a constant unless specified otherwise. We argue that it is "misses/Kbytes" rather than "misses/Kpackets" that should be the metric for miss ratio when traffic volume rather than packet volume are used predominantly as the metric for throughput. However, the "miss ratio" we felt comfortable with is "misses/Kpkts". Fixing the packet size allows us to talk about "misses/Kpkts" but to mean "misses/Kbytes". In contrast, introducing packet size as a variable will make our following discussion unnecessarily complicated and confusing.

Due to a number of technical challenges, caching is not looked upon as a promising technology in the layer-4 lookup research community. Both layer-4 packet classification papers [14, 30] in the SIGCOMM'98 are skeptical whether full-header layer-4 caching can be a stable, reliable, and cost-effective approach. Looking at these skepticism in a positive way, they are posing a number of challenges that an effective layer-4 caching scheme must meet in order to be technologically and economically viable. Two major challenges are posed by these criticisms. First, the cache architecture should deliver a high, stable, and predictable hit ratio on average as well as in the worst case. Srinivasan et al. doubt whether layer-4 caching can achieve a decent hit ratio when layer-3 route cache can only achieve 60% to 80% hit ratio in backbone routers[19]. Lakshman and Stiliadis [14] are concerned that the hit ratio may not be stable enough so that when the hit ratio is low, the stateless packet classifiers will be temporarily overloaded. We meet this challenge by delivering a cache architecture that achieves 92% hit ratio with 0.6% percent standard deviation. Second, it should withstand denial-of-service attack. Lakshman and Stiliadis [14] are concerned that the caching scheme can not withstand the denial-of-service attack. Their argument is that "a malicious user or group of users discovering the limitations of the hash algorithms or caching techniques, can generate traffic patterns that force the router to slow down and drop a large portion of the packets arriving at a particular interface[14]." We will address this issue in Section 9. In addition to these two challenges, we pose an additional one to ourselves. The cache should synchronize with route updates from time to time without disrupting its service. In the layer-3 50 Gbps

router designed by BBN, the entire layer-3 route cache has to be invalidated periodically because the Internet route is unstable and keeps changing[13]. This leads to a "cold start" of the cache periodically and the miss ratio during that period is expected to be high. Our cache architecture will automatically invalidate cache entries that are $D_{lifespan}$ seconds old without affecting its miss ratio and throughput. $D_{lifespan}$ is set to 2 minutes in our implementation.

4 The Notion of Near-LRU Cache

4.1 Flow Theory

Over the past thirty years, to investigate how to effectively perform paging and CPU caching, a huge amount of effort has been put into studying the temporal and spatial *locality behavior* of the program data and the results have shed light on how the cache should be structured and managed to best exploit the *locality behavior* under many technological and economical constraints[1, 4, 7, 8, 21, 24, 27, 28, 29]. Likewise, locality of the network traffic at layer-4 has been studied in Claffy's Ph.D thesis on Internet traffic characterization [3]. The concept of a *flow* is introduced in the thesis to characterize the locality behavior of the Internet traffic. Many types of flows can be defined by varying the aspects like directionality, granularity, and protocol layer[3]. In this paper, a flow is defined as a series of unidirectional packets that share the same layer-4 address. A flow is started when the first packet of the flow arrives and is considered *expired* when there has been no activity for a timeout period D_{expire}. This concept of flow is actually an extension of Jain's *packet train model* [11] at layer-4, and D_{expire} is the counterpart of MAIG (Maximum Allowed Inter-car Gap) in that model. A flow is said to be *active* from the time it was created until the time it expires.

Flow theory [3] introduces some important metrics defined over an Internet traffic trace, among them two are important to the design and performance evaluation of our layer-4 route cache architecture. One is the arrival rate of new flows $\rho(D_{expire}, t)$ and the other is the number of *active flows* $F(D_{expire}, t)$. Both are functions of D_{expire} and t. Based on the flow theory, a large amount of measurement work has been conducted by NLANR (National Laboratory for Applied Network Research) to measure the metrics introduced in the flow theory[6, 31]. From the figures and data presented in [6] and [31], we find that the values of F and ρ are almost *stationary* (not a function of t) provided that the *traffic volume* U is fairly stationary. The assumption of the stationarity of U is acceptable because we aim to design a route cache that is capable of achieve a stable hit ratio under the bombardment of the sustained maximum (but constant) traffic volume U_{max}. The

Trace	1	2	3	4	5	6	7
Date	6/21/95	2/28/96	9/18/96	9/26/96	1/9/97	5/17/97	11/20/97
Time	15:00	21:45	20:17	19:17	17:51	10:02	19:45
Duration(min)	1625	770	298	1233	1084	298	1155
Avg. Pkts/s	14414	13625	13939	9995	8088	3856	9494

Table 1: The FIXWEST Traces Used in Our Trace-Driven Simulation

stationarity of these two metrics is further corroborated by our trace-driven simulation.

4.2 Traces used in the simulation

Table 1 contains seven FIXWEST traces obtained from NLANR's FTP site. Since the concept of layer-4 address is only relevant to TCP/UDP packets, we exclude other packets from these traces. The packet volume (Avg. Pkts/s) in the table reflects this exclusion. These traces will be used intensively in our trace-driven simulations. We choose FIXWEST traces for two reasons. First, we want to choose backbone traces because the number of *active flows* (D_{expire} fixed) per Mbps will be much higher in backbone traces than in campus or corporate gateway traces [3, 15, 31]. Second, AIX traces, the other group of backbone traces available from NLANR, only sample 25% of the back-to-back packets, which would render our simulation study meaningless. Among the seven traces, trace 1 contains the heaviest packet volume, traffic volume and has the longest duration. The simulation results from this trace will be presented throughout this paper. Simulation results from other traces produce similar conclusions.

4.3 The stationarity of ρ and F

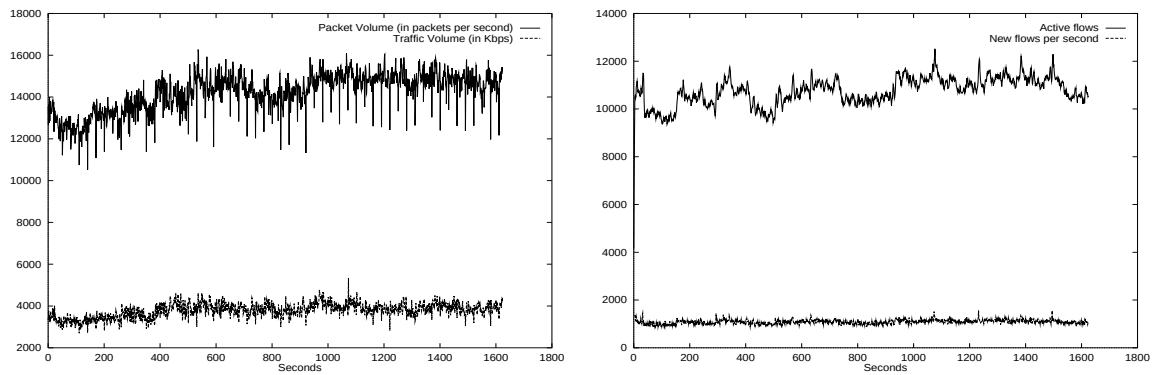

Figure 2: (a)Packet Volume and Traffic Volume (b)Active Flows and New Flows Per Second

Fig. 2(a) shows the traffic volume U and packet volume V over the 1625-second time period. Fig. 2(b) shows the number of active flows F ($D_{expire} = 5s$) and the arrival rate of new flows ρ over this period. Visually, it can be seen that F is no more burster than V, and ρ is much less bursty than U. Statistically, $C_X^2 = \frac{Var(X)}{(E[X])^2}$ *coefficient of variation* of a random variable X, measures the *variability* of X[18]. The smaller the C_X^2, the less variable is X. We measured that C_U^2=0.006183, C_V^2=0.002817, C_F^2=0.002675, and C_ρ^2=0.004882. So $C_F^2 < C_V^2$ and $C_\rho^2 < C_U^2$ verify our visual observations from the figure. Other traces offer the same conclusion on this issue.

Based on these observations, we propose a fully-associative cache management algorithm called *near-LRU*, which works as follows. Each cache entry is associated with a timestamp and the timestamp is updated whenever this cache entry is accessed. When there is a need for cache replacement, cache entries that have not been accessed for more than D_{expire} seconds (considered *expired*) are candidates for replacement. Assuming no collision happens and the expired entries are not allowed to produce a hit, then average miss ratio achieved by near-LRU cache is $R_{near-LRU} = \frac{\rho}{V}$, the arrival rate of new flows divided by the packet volume. The reason is that each new flow causes exactly one miss when it arrives. So when the packet volume is almost stationary, the stationarity of ρ indicates that $R_{near-LRU}$ is fairly stationary. From the same set of data used in Fig. 2, we measured that $E[R_{near-LRU}] = 9.261\%$ and $\sqrt{Var(R_{near-LRU})} = 0.006044$. We can see that this is a very high and stable miss ratio. F indicates the approximate number of entries the cache architecture needs to have. The stationarity of F allows us to provision the cache size for the worst scenario without much waste.

4.4 The near-optimality of near-LRU

We have shown that near-LRU delivers a high stable hit ratio. We now show that it is actually a near-optimal cache management algorithm in the sense that it achieves almost the lowest hit ratio among nonlookahead (no knowledge about future) cache management algorithms. We prove this in two steps. First, we show that near-LRU is almost equivalent to LRU in caching Internet traffic. Suppose the number of active flows F is constant over the time and we have an F-entry near-LRU cache. Then when a new flow arrives, exactly one cache entry should expire (otherwise, we have F+1 active flows) and it is selected for replacement. However, this entry is obviously the least-recently used entry. So given that F is almost stationary, this management algorithm is almost identical to LRU algorithm.

Second, we need to show that, given the number of cache entries, LRU cache achieves the

lowest miss ratio among nonlookahead algorithms. LRU stack model is typically used to measure the locality behavior of a memory or traffic trace [29]. In the stack model, the stack contains the addresses that are referenced in the past. When an address is referenced, it is taken out from its current location and pushed to the top of the stack. Let P_i to denote the probability of i_{th} stack position (1st position is the stack top) being referenced. Two *locality conditions*, *strong locality condition* and *weak locality condition*, are defined in the virtual memory literature [29]. The address references are said to satisfy *strong locality condition* if this *stack reference probability* is monotonically decreasing ($P_1 > P_2 > P_3 > \cdots$). The address references are said to satisfy *weak locality condition* at size S if $MIN(P_1, P_2, \cdots, P_S) > MAX(P_{S+1}, P_{S+2}, \cdots)$. It is proved in the virtual memory literature that with S entries of cache, LRU is the best replacement algorithm if (1) the *strong locality condition* is satisfied, or (2) the *weak locality condition* at size S is satisfied[29].

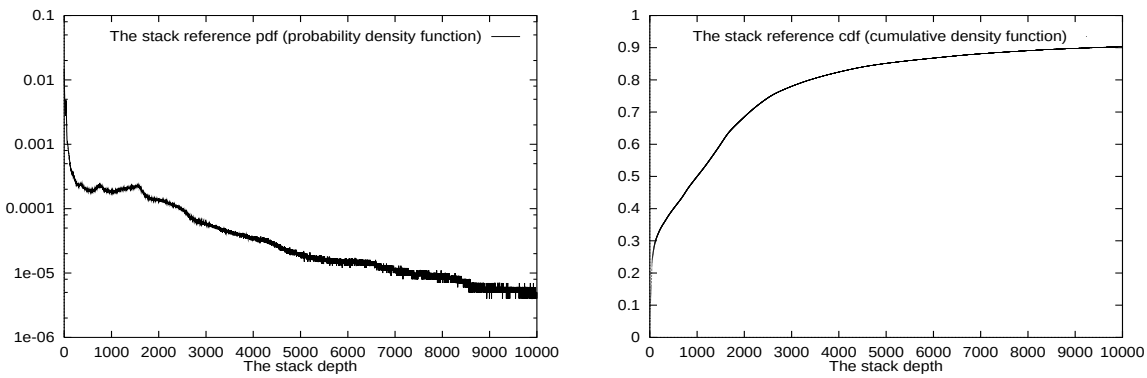

Figure 3: (a)Stack Reference pdf (b)Stack Reference cdf

Fig. 3(a) shows the stack reference *probability density function* (pdf) measured from FIXWEST trace 1. It shows that except for some humps before the stack distance 2000, the pdf keeps decreasing afterwards. The *weak locality condition* is satisfied for any $S > 2000$. Measurements after other traces produce the similar results. According to (2) we conclude that when the cache size is fairly large (> 2000), LRU should be the optimal cache management algorithm. Jain has performed a similar measurement at layer 3 and also located such a hump. Jain uses (1) to show that LRU is the optimal for the noninteractive traffic[10].

Fig. 3(b) is the corresponding stack reference *cumulative distribution function* (cdf). According to the definition of the stack model, we know that a hit occurs if and only if the stack distance being referenced is less than S, the size of the cache. So this curve shows the relationship between the cache size and the hit ratio achievable by LRU cache of this size. This figure shows that it takes about 10000 entries to achieve a hit ratio above 90%.

4.5 The growth of F and ρ in the future

The average TCP/UDP traffic volume inside FIXWEST trace 1 is only about 30.8 Mbps, and as shown in Fig. 3(b), about 10,000 entries of cache are needed to achieve a hit ratio above 90%. This section provides the answers to two important questions. First, can the same or higher hit ratio be achieved in the future by near-LRU cache when the traffic volume is as high as 50 to 150 Gbps? Second, if the same high hit ratio can be guaranteed, how many cache entries do we have to provide?

To answer the first question, we need to notice that the miss ratio of near-LRU cache is equal to $\frac{1}{\sigma}$, σ being the average number of packets in a flow. The reason is that in near-LRU cache, exactly one miss is encountered for each and every flow. A closer look at σ reveals that it reflects the average amount of traffic inside a flow (under the constant packet size assumption). So the miss ratio will be higher in the future if the average amount of the traffic (or number of packets) in a flow is larger. Using the same insights in arguing for the packet train model [11], a flow is typically a series of packets that are transporting an FTP file or an HTTP homepage. As the average size of the networking object keeps increasing thanks to the fancier homepages and larger application files, the average number of packets in a flow is at least not going to decrease. So with the same D_{expire} value, the near-LRU cache should have an equal or higher hit ratio in the future.

To answer the second question, it is worth noting that F actually reflects the number of active users at any given time. When the total traffic volume increases, it will certainly accommodate more active users and hence higher F. However, the growth of F should not be faster than the growth of the traffic volume U, that is, $\frac{F(future)}{F(now)} \leq \frac{U(future)}{U(now)}$. The reason is that the per-user bandwidth is actually $\frac{U}{F}$, the total traffic volume divided by the number of active flows (active users). Assuming that future Internet users will enjoy an equal or higher per-user bandwidth, we have $\frac{U(now)}{F(now)} \leq \frac{U(future)}{F(future)}$, which is equivalent to the formula above.

We assume that, in the worst case, F grows as fast as U. As we have seen, 10,000 entries are needed to achieve a hit ratio above 90% for a 30.8 Mbps trace. Therefore, a 50 Gbps layer-4 router will need at least 16 million entries of cache. As each layer-4 cache entry is 128-bit long (explained later), we are looking at 256 Mbyte cache memory.

5 Proposed Cache Architecture

A cache is nothing more than chunks of RAM augmented by some auxiliary logic for search and replacement. The RAM we will use in our route cache is an emerging DRAM technology called Synchronous Link DRAM (SLDRAM). SLDRAM can deliver a very high throughput for read/write in *burst mode*. A burst read of length L starting from address A is $read(A)$, $read(A+1)$, $\cdots$, $read(A+L-1)$. SLDRAM is perfect for layer-4 route caching because accessing a 128-bit layer-4 route is exactly burst reads/writes that SLDRAM is geared toward. A 400/800 MHz SLDRAM can deliver 16 bits[1] every 2.5/1.25 ns and therefore can read/write a 128-bit layer-4 route every 20/10 ns! A SLDRAM chip consists of a number of independent DRAM banks augmented by interleaving logic to sustain high bandwidth and pipelining logic to allow a new read/write to be issued before the previous read/write is finished. Since the core technology inside SLDRAM is DRAM and the pipelining and interleaving logic only adds 10% to 20% cost penalty, we expect that the price of the SLDRAM will drop to about 2 dollars per Mbyte (DRAM is about 1.5 dollars per Mbyte) in the near future. As in the worst case 256 to 784 MB route cache are needed for a 50/150 Gbps router, we are looking at a memory cost of 500/1500 dollars.

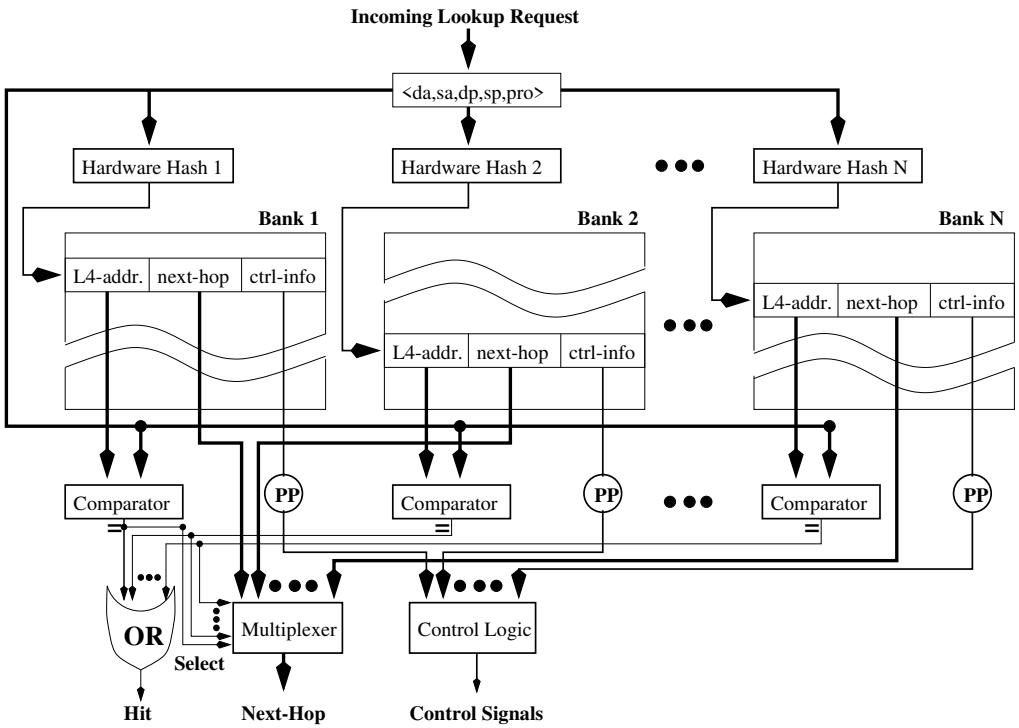

Figure 4: Hardware Design of Our Caching Approach

[1] It is actually 18 bits but 2 bits are used for error correction.

The architecture of the route cache is shown in Fig. 4. It consists of N independent SLDRAM banks, each storing $M = 2^r$ entries. When a lookup request arrives, the layer-4 address in the request will be processed by N different hardware hash functions in parallel and will produce N r-bit hash values. Each hash value can be viewed as a memory location indexed into an SLDRAM bank where the route cache that matches the layer-4 address may potentially be stored. Each cache entry contains three fields, a 97-bit layer-4 address ("L4-addr."), a 16-bit forwarding decision ("next-hop") field, and a 15-bit control information ("ctrl-info") field. The "ctrl-info" field contains three subfields, a 1-bit VALID, a 7-bit CREATE, and a 7-bit ACCESS. The VALID bit indicates whether the entry contains a valid route cache. The CREATE stores a timestamp indicating when the entry was created. This field is used to check whether the entry is created before $T - D_{lifespan}$, $D_{lifespan}$ being the maximum allowable lifespan of a cache entry (we set it to 120s) and T being the current time. If this is true, the cache entry can no longer be trusted. The ACCESS field stores the time when this entry was last accessed. This field will be compared with $T - D_{expire}$ to check whether the entry has expired. If it has, then it is a potential candidate for replacement. All these comparisons are done in the Preprocessing (PP) modules in parallel and the results are fed to the control logic. The coding and manipulation of ACCESS and CREATE bits are detailed in Appendix B.

The "L4-addr." fields in these N entries will be compared with the layer-4 address in the incoming lookup request in parallel. If one entry matches, we have a hit and its "next-hop" field will be outputted as the final "next-hop" result from the multiplexer. When a hit occurs, the ACCESS field will be updated to record the time of this recent access. If a miss happens, the request will be sent to a stateless packet classifier for a full-fledged lookup. To avoid write/write race condition, an interim cache entry indicating that "the forwarding decision is in search" is written into the entry to be replaced. In case no entry expires, a *collision* happens and an entry is randomly chosen for replacement.

If these N hardware hash functions were the same, this architecture would be equivalent to an N-way set-associative cache. However, we will show in Section 6 that N-way set-associative cache will bring about a high amount of *collision miss* when its load ($\frac{F}{MN}$) is high. We have invented a new set-associative scheme, called *dynamic set-associative scheme*, and will refer to regular set-associative scheme as *static set-associative scheme*. Under the same load, the dynamic set-associative scheme can cut the *collision miss* by 60% to 90%. In the *dynamic set-associative scheme*, these N hardware hash functions, h_1, h_2, $\cdots$, and h_N are not only different, but also have

to satisfy three nice properties. First, every function can perform a hash within 10ns. Second, each is a uniform hash function in the sense that given a random layer-4 address X, $h_j(X)$ is randomly distributed among $\{0, 1, 2, \cdots, 2^r - 1\}$, j= 1,2,$\cdots$,N. Third, these hash functions are statistically independent in the sense that given a random layer-4 address X, $h_1(X)$,$h_2(X)$,$\cdots$,and $h_N(X)$ are statistically independent random variables. Counterintuitively, such hash functions are not hard to find. Choosing N hash functions randomly from a class of hashing functions (subject to some restrictions as specified in [22, 23]) called H_3 defined by Carter and Wegman[2] readily satisfy the first and the second requirements. H_3 functions are linear transformations over GF(2) from a w-dimensional vector space to a r-dimensional vector space [2] and its detailed description is in Appendix A. Ramakrishna et al. showed in [22, 23] that an H_3 function is easily implementable in hardware using two-level combinatorial logic. Even implemented using the low-end 1 micron CMOS, such an H_3 function can compute the hash value well within 10ns. Ramakrishna et al. also demonstrated that a randomly generated H_3 hash function achieves exactly the same performance as uniform hashing according to frequently-used metrics to measure the performance of a hash function such as expected length of successful search with separate chaining. Ramakrishna also mentioned that the H_3 functions satisfy the third requirement, but offered no definition, explanation, or proof[22, 23]. We proved a theorem that states if $N * r \leq w$ then N carefully-chosen hash functions will satisfy the third requirement (see Appendix A for details). However, in our cache architecture this requirement is not satisfied ($N \times r$=8$\times$17=136>97=w) and the N(=8) hash functions are indeed dependent. Fortunately, the metric we are interested in (*collision ratio* as will be discussed in Section 6) is virtually not affected by the weak dependence among the hash functions. It will be shown that the analytical results of the *collision ratio* obtained by assuming that $h_1(X)$,$h_2(X)$,$\cdots$,$h_N(X)$ are independent random variables perfectly match our simulation results!

We have shown in Section 3 that the desired speedup $\frac{1}{R}$ can not be achieved by caching if the throughput of the cache, H_{cache}, becomes a bottleneck. With proper pipelining, the hashing, comparison and replacement operations can be done in parallel with the memory read/write. None of these operations will take more than 10ns, the time needed for 800 MHz SLDRAM to access a 128-bit layer-4 route. The SLDRAM is the bottleneck of our cache architecture. A read is needed for each layer-4 lookup. If a hit occurs, one write is needed to update the ACCESS bits. If a miss occurs, two writes are needed to write an interim entry and a final entry. With a miss ratio of 10%, each layer-4 lookup will access the memory 2.1 times on average. So the throughput of the cache in terms of MLPS is the available throughput of the memory divided by 2.1. Considering the factor

that memory refresh takes away about 6% of the bandwidth, a 400/800 MHz SLDRAM can deliver 22.5/45 MLPS, enough to support 45/90 Gbps layer-4 routers.

The hardware logic used in the proposed architecture is quite simple. We estimate that the cost of the auxiliary logic should not exceed 200 dollars.

6 Static vs. Dynamic Set-Associative Scheme

In this section, we present how our *dynamic set-associative scheme* achieves a lower miss ratio than *static set-associative scheme*. The miss ratio of our architecture R can be broken down as R=R_{new}+$R_{collision}$-$R_{stealing}$. This formula also holds for both static and dynamic associative schemes. $R_{collision}$ refers to the portion of miss ratio that is caused by collision. R_{new} refers to the misses caused by the arrival of the new flows that do not cause a collision. Given D_{expire}, R_{new} is slightly smaller than $R_{near-LRU}$ because some cache misses due to collision are actually new flows, but they are counted in $R_{collision}$. As the the number of active flows is usually smaller than the number of cache entries, a certain percentage of cache entries is expired. However, there is still a good chance for one of them to be accessed and "reinstated" into an active flow. This entry is counted as a new flow in R_{new} but it causes no miss. So some miss ratio should be deducted from R_{new} and we call it *stealing ratio*, denoted as $R_{stealing}$.

When D_{expire} is fixed, R_{new} and $R_{stealing}$ are almost the same in static scheme and in dynamic scheme. The difference between the miss ratio that they can achieve comes from $R_{collision}$. Let λ denote the number of collisions per second, then we know that $R_{collision} = \frac{\lambda}{V}$, the number of collisions per second divided by the number of packets per second.

Let us put F distinct items into the dynamic set-associative cache, and let $O_{dynamic}$ be the average number of collisions that occurs in the process. O_{static} is defined in the same way. For both static and dynamic set-associative scheme, We can see that O ($O_{dynamic}$ or O_{static}) actually represents the average number of active flows that is not cached at any given time. We further find that there is a linear relationship between O and λ as such that $\lambda = cO$, c being called the *collision rate*. This fact is indirectly proved in a CPU caching paper[1] and is corroborated by our experimental results from FIXWEST trace 1.

We can see from Table 2 that c almost remains constants while O and λ vary. We define *collision ratio* Γ as the average number of active flows in collision divided by the number of active flows. Then $\Gamma = \frac{O}{F}$. Then $R_{overflow}$ is expressed as $R_{overflow} = \frac{\lambda}{V} = \frac{cO}{V} = \frac{cF\Gamma}{V} = \left(\frac{cF}{V}\right)\Gamma$. $\frac{cF}{V}$ is a constant

D_{expire} (s)	4.0	4.4	4.8	5.2	5.6	6.0
$E[F]$	10751	11283	11787	12287	12773	13245
$E[O]$	53	77	110	160	226	312
$E[\lambda]$	16.676	25.426	37.909	54.067	74.924	101.272
Collision rate (c)	0.3146	0.3302	0.3446	0.3379	0.3315	0.3246

Table 2: The relationship between Overflow and Collision Miss

when the traffic trace, M, N, and D_{expire} are fixed. This constant is measured to be between 0.3 and 0.5 from FIXWEST traces when D_{expire} is between 4 to 6 seconds. Therefore $R_{overflow}$ is about 30 to 50 percent of Γ. In the following we demonstrate that under the same load condition, $\Gamma_{dynamic}$ is much smaller than Γ_{static}. We define $\alpha = \frac{F}{MN}$ as the load ratio of the cache architecture. Both $\Gamma_{dynamic}$ and Γ_{static} are found to be a function of N and α, but not a function of M.

Γ_{static} is estimated as follows. Let Y_j be the random variable that is equal to the number of entries that is mapped to set j, $j = 1, 2, \cdots, M$. A set is a row of N entries with the same index. Then $P[Y_j = i] = \binom{F}{i}(1/M)^i(1-1/M)^{F-i} \quad i = 1, 2, \cdots, F$. When M is large enough, this can be approximated by Poisson distribution $P[Y_j = i] = \frac{(N\alpha)^i exp(-N\alpha)}{i!}$. Then the expected number of overflows O_j from any set j ($j = 1, 2, \cdots, F$) is

$$E[O_j] = \sum_{i=N+1}^{\infty} P[Y_j = i](i-N) = N\alpha - \sum_{i=1}^{N} i\frac{(\alpha N)^i exp(-\alpha N)}{i!} - N\left(1 - \sum_{i=0}^{N}\frac{(\alpha N)^i exp(-\alpha N)}{i!}\right) \quad (1)$$

Let O be the number of overflows, then the expectation of the overflow ratio Γ_{static} is:

$$\Gamma_{static} = \frac{E[O]}{F} = \frac{\sum_{j=1}^{M} E[O_j]}{MN\alpha} = 1 - \frac{1}{N\alpha}\sum_{i=1}^{N}\frac{i(\alpha N)^i exp(-\alpha N)}{i!} - \frac{1}{\alpha}\left(1 - \sum_{i=0}^{N}\frac{(\alpha N)^i exp(-\alpha N)}{i!}\right) \quad (2)$$

$\Gamma_{dynamic}$ is estimated as follows. We use the same analysis method used in Mckenny's paper[17, 16]. Let us assume $p_{x,y}$ denotes the probability that x data items are put into the system and y items settle. Then assume that all the items that settle is evenly distributed in the N banks. We have the following recurrence condition:

$$p(x,s) = \left(\frac{s}{NM}\right)^N p(x-1,s) + \left[1 - \left(\frac{s-1}{MN}\right)^N\right] p(x-1,s-1) \quad (3)$$

On the right-hand side, the first term is the probability that the x_{th} item collides with other items in all N banks. The second term is the probability that the x_{th} does not collide with another item in at least one bank and gets inserted. The initial condition is p(1,1)=1 and p(1,s)=0, s=1,2,$\cdots$,F.

Load	$\Gamma_{dynamic}$			$\Gamma_{dynamic}$		
Ratio	Theory	Experiment	Stdev	Theory	Experiment	Stdev
0.6	1.851e-3	1.825e-3	5.772e-5	2.031e-2	2.029e-2	1.668e-4
0.7	6.237e-3	6.265e-3	8.001e-5	3.971e-2	3.973e-2	3.246e-4
0.8	1.730e-2	1.730e-2	1.015e-4	6.682e-2	6.673e-2	2.297e-4
0.9	4.006e-2	4.014e-2	2.395e-4	1.007e-1	1.007e-1	3.601e-4

Table 3: Theoretical and Experimental Results of the Two organizations

The expectation of the overflow ratio $\Gamma_{dynamic}$ is

$$\Gamma_{dynamic} = \frac{E[O]}{F} = \sum_{s=1}^{F} \frac{(F-s)p(F,s)}{F} = 1 - \sum_{s=1}^{F} \frac{sp(F,s)}{F} \tag{4}$$

However, this recurrence condition is not as simple as the recurrence condition in Mckenney's paper, and we can not get a mathematically neat expression for Γ. However, $p(F,s)$, s=1,2,$\cdots$,F can be calculated using a two-dimensional dynamic programming algorithm and through optimization the space complexity of the algorithm is only $O(F)$.

Table shows $\Gamma_{dynamic}$ and Γ_{static} under various load conditions (from 0.5 to 0.9). The theoretical results are calculated using equation (2) and (4). To obtain the experimental results, we simulated a route cache architecture in which $N = 8$ and $M = 2^17$. It uses eight hash functions randomly generated from H_3 to simulate dynamic set-associative scheme. Eight identical hash functions (also from H_3) are used to simulate static set-associative scheme. We use randomly-generated 97-bit layer-4 addresses as the traffic source in this simulation.

Ten experiments are conducted with each experimental data entry and hence the standard deviation ("Stdev"). We can see that the theoretical and simulation results are very close and the standard deviations are very small. This not only corroborates all the properties of our hash functions, but also proves the accuracy of our analysis study. Also from these data, we can see that given a load, $\Gamma_{dynamic}$ is much smaller than Γ_{static}. For example, when the load ratio (α) is 0.8, Γ_{static} is 4 times larger than $\Gamma_{dynamic}$.

7 Simulation Results

We simulated the miss ratio of our cache architecture under the traffic load of FIXWEST trace 1. The cache architecture consists of $N = 8$ banks with $M = 2048$ entries each. We obtained a

total miss ratio of 8.0 percent (D_{expire}=5.3s) with a standard deviation of 0.6%. This miss ratio is very close to the theoretical minimum miss ratio 7.7 percent that is achieved using 16384 entries of LRU cache. In contrast, we ran the simulation on an 8-way set-associative cache with random replacement policy with same M and N, and got a 10.8 percent miss ratio.

Our results are nice because our hash functions perform in exactly the same way as the analytical results predict. However, we are faced with the charges that the hashing result is based on the "sanitized trace" rather than true layer-4 addresses. We have following arguments against these charges. First, Ramakrishna et al. have shown that H_3 hash functions perform well on real-life data that are not necessarily random[22, 23]. Second, the IP contained in "sanitized trace" looks like (1.0.0.1, 1.0.0.2, ...), which is far from random. Third, we obtain 32921 distinct IP addresses from an IP trace of 2146573 addresses (from NLANR) and use them to test the hash functions we use. The simulation result is very close to the experimental result.

8 Lazy-Writeback: Double the Throughput at No Cost

We have seen in Section 5 that the throughput of the route cache is about $\frac{1}{2.1}$ of the memory throughput. A hit will need two memory accesses because a *writeback* is needed to update the ACCESS bits. However, if a series of packets (say 8) hit a layer-4 route entry within a very short period, there is really no need to perform a *writeback* every time a hit happens. A writeback is needed only after the 1st and the 8th access. We proposed a scheme, called *lazy-writeback* to reduce the number of such unnecessary writebacks as follows. When a hit happens, there will be writeback only if the entry is more than $D_{writeback}$ seconds old. We have simulated the effect of this scheme using the same setting ($D_{expire} = 5.3s$, N=8, $M = 1024$) as used in the last section. We set the $D_{writeback}$ to 1 second. We find that the miss ratio R is increased only by 0.04% (from 8.00% to 8.04%). However, this scheme cuts the percentage of writebacks by 83.3% (from 100% to 16.7%). Using this scheme, the average number of memory accesses per lookup is reduced to 1.25. The throughput of the cache using 400/800 MHz SLDRAM is now improved to 37.5/75 MLPS, enough to support 75/150 Gbps layer-4 routers.

9 Issues on Denial-of-Service Attack

In this section, we describe how our cache architecture guards against denial-of-service attack. We consider two types of attacks to a cache architecture. The first type is called "targeted attack",

in which a malicious party specifically targets particular cache sets in his attack. For example, if the cache were built on regular set-associative cache (using bit-extraction as the hash function), it would be very easy for a hacker to "overcrowd" particular cache sets so that legitimate flows that are hashed into these sets will undergo severe thrashing. However, with our dynamic hashing scheme, the malicious packets from the hacker will be uniformly distributed in the cache so that no particular set or sets will be hurt severely. The other type of attack is to reduce the overall throughput and amount of route cache available to the legitimate users. We propose a reasonable assumption called *limited hacker bandwidth assumption*. We assume that the bandwidth that a hacker can have access to is only a tiny percentage (e.g., 0.1 percent) of the throughput of a router. Suppose that a hacker can not send out bad packets at no more than T3 rate, it can be shown that overprovisioning the stateless lookup power by 5% and the size of route cache by 2.5% will effectively guard against this type of attack.

10 Conclusions

In this paper, we proposed a cost-effective layer-4 route cache architecture that achieves a high and stable hit ratio. It is able to support 45 to 150 Gbps layer-4 routing at a cost between 700 and 1700 dollars. The cache architecture satisfies all the challenges posed by other layer-4 lookup researchers. The dynamic set-associative scheme used in our architecture to reduce miss ratio is novel not merely to networking literature, but also to caching and hashing research. The design is complete by itself and implementation-ready.

References

[1] A. Agarwal, M. Horowitz, and J. Hennessy. An analytical cache model. *ACM Trans. on Computer Systems*, 7(2):184–213, May 1989.

[2] J. Carter and M. Wegman. Universal classes of hash functions. *J. of Computer and System Sciences*, 18(2):143–154, 1979.

[3] K. Claffy. *Internet Workload Characterization.* PhD thesis, UC San Diego, June 1994.

[4] P. Denning. The working set model for program behavior. *Comm. of ACM*, 11(5):323–333, May 1968.

[5] D. Feldmeier. Improving gateway performance with a routing-table cache. In *Proc. of IEEE INFOCOM'88*, pages 298–307, New Orleans, LA, March 1988.

[6] National Laboratory for Applied Network Research (NLANR). Flow statistics analysis for fix-west. http://www.nlanr.net/, June 1998.

[7] M. Hill. A case for direct-mapped caches. *IEEE Computer*, pages 25–40, December 1988.

[8] M. Hill and A. Smith. Evaluating associativity in cpu caches. *IEEE Trans. on Computers*, 38(12):1612–1630, December 1989.

[9] IEEE Computer Society. *Draft Standard for A High-Speed Memory Interface (SyncLink)*, 1996.

[10] R. Jain. Characteristics of destination address locality in computer networks: A comparison of caching schemes. *Computer Networks and ISDN Systems*, (18):243–254, 1990.

[11] R. Jain and S. Routhier. Packet trains: Measurements and a new modle for computer network traffic. *IEEE JSAC*, 4:986–995, September 1986.

[12] B. Kowalski. Atlas policy cache architecture. Technical report, StorageTek Corp., http://www.network.com/, 1996.

[13] C. Labovitz, G. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Trans. on Networking*, 6(5):515–528, October 1998.

[14] T. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proc. of ACM SIGCOMM'98*, September 1998.

[15] S. Lin and N. McKeown. A simulation study of ip switching. In *Proc. of SIGCOMM'97*, 1997.

[16] P. McKenny. High-speed event counting and classification using a dictionary hash technique. In *Proc. of ICPP'89*, pages 71–75, 1989.

[17] P. Mckenny and J. Sarvela. High-speed event counting and classification using a dictionary hash technique. Technical report, SRI International, 1989.

[18] R. Nelson. *Probability, Stochastic Processes, and Queueing Theory*. Springer-Verlag, NY, 1995.

[19] P. Newman, G. Minshall, and L. Huston. Ip switching and gigabit routers. *IEEE Communications Magazine*, January 1997.

[20] C. Partridge et al. A 50-gb/s ip router. *IEEE/ACM Trans. on Networking*, 6(3):237–248, June 1998.

[21] S. Przybylski. *Cache and Memory Hierarchy Design–A Performance Directed Approach*. Morgan Kaufmann, 1990.

[22] M. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient hardware hashing functions for high performance computers. Technical Report TR-96-13, Computer Science Dept., RMIT, Melbourne, Australia, 1996.

[23] M. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient hardware hashing functions for high performance computers. *IEEE Trans. on Computers*, 46(12):1378–1381, December 1997.

[24] G. Rao. Performance analysis of cache memories. *J. of ACM*, 25(3):378–395, July 1978.

[25] MUSIC Semiconductors. Cam tutorial, 1998.

[26] SLDRAM Inc. *400 Mb/s/pin SLDRAM*, 1998.

[27] A. Smith. A comparative study of set associative memory mapping algorithms and their use for cache and main memory. *IEEE Trans. on Software Engineering*, 4(2):121–130, March 1978.

[28] A. Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, September 1982.

[29] J. Spirn. *Program Behavior: Models and Measurements.* Elsevier, 1977.

[30] V. Srinivasna, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *Proc. of ACM SIGCOMM'98*, September 1998.

[31] K. Thompson, G. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, pages 10–23, November 1997.

Appendix A. Details about H_3 Hash Functions

A class of hashing functios called H3 are defined by Carter and Wegman [2]. Each hash function in H_3 class is a linear transformation that maps a w-bit binary string $A = a_1 a_2 \cdots a_w$ to an r-bit binary string $B = b_1 b_2 \cdots b_r$ as follows:

$$B^T = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_r \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1w} \\ q_{21} & q_{22} & \cdots & q_{2w} \\ \cdots & \cdots & \cdots & \cdots \\ q_{r1} & q_{r2} & \cdots & q_{rw} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_w \end{pmatrix} = QA^T \tag{5}$$

Here a k-bit string is treated as a k-dimensional vector over GF(2) = {0,1} and T stands for transposition. Q is a $w \times r$ matrix defined over GF(2) and each hash function in H3 is uniquely corresponding to such a Q. The multiplication and addition in GF(2) is boolean AND (denoted as $\circ$) and XOR (denoted as $\oplus$), respectively. Each bit of B is calculated as:

$$b_i = (a_1 \circ q_{i1}) \oplus (a_2 \circ q_{i2}) \oplus \cdots \oplus (a_w \circ q_{iw}) \qquad i = 1, 2, \cdots, r \tag{6}$$

As shown in [22, 23], such a function in H3 is easily implementable in hardware using two-level combinatorial logic. Let $\beta_1, \beta_2, \cdots, \beta_r$ stand for the row vectors of the matrix Q such that $\beta_i = (q_{i1}, q_{i2}, \cdots, q_{iw}) \quad i = 1, 2, \cdots, r$. Actually such a hashing hardware is dynamically reconfigurable by equiping the hashing hardware with r w-bit writeable registers $R_1, R_2, \cdots, R_r$ loaded with $\beta_1, \beta_2, \cdots, \beta_r$. Actually, a hash function is randomly generated when $\beta_1, \beta_2, \cdots, \beta_r$ are independently generated by random number generators.

A significant amount of measurement is done in [22, 23] to demonstrate that a randomly generated H3 function achieves exactly the same performance as uniform hashing according to four

common metrics that measure the performance of a hash function such as expected length of successful search with seperate chaining. Their conclusion is that by choosing functions at random from the class H3 the analytical performance of hashing can be achieved with real life data. Note that it is not necessary that the real life data are random. It is also claimed in the paper that different hash functions randomly chosen from H3 class is statistically independent. However, the concept of statistically independence is not defined in this case. Another paper by Mckenny [16] also use this concept without definition.

We prove the following theorem. Suppose there are N hash functions and each hash function generates a r-bit word and the key field is w-bit long. Then if $r \times N \leq w$, we can somehow properly choose functions from H_3 so that they are statistically independent if the key is uniformly distributed over the key space. The way to choose is that let $m = r \times N$, and $\beta_1, \beta_2, \cdots, \beta_m$ be all these row vectors, we need to make sure that these row vectors are linearly independent. The proof of this theorem only involves the elemarntary theorms of linear equations and the inferences of uniform distribution assumption. Due to the lack of space, we skip it.

However, when $w \leq m$, then these $\beta_1, \beta_2, \cdots, \beta_m$ are not linearly independent and these hash functions are not statistically independent. However, the dependence among these functions are weak. In calculating our metric, the collision ratio Γ, when $\frac{m}{w}$ is not too large we can treat $h_1(A)$, $h_2(A)$, $\cdots$, $h_N(A)$ as independent random variables and the analysis result will still perfectly match the simulation result!

Appendix B. Route Cache Hardware Implementation Details

Due to the limitation of space, we can only sketch some vital hardware implementation details. The 7-bit ACCESS field is incremented every 1/8 second in $0 \rightarrow 1 \rightarrow 2 \rightarrow \cdots \rightarrow 127 \rightarrow 0 \rightarrow \cdots$ rounds. Two comparisions are needed to check if T $\in$ [ACCESS, ACCESS+D_{expire}] (T is the current time), when the threshold 127 is not crossed. Otherwise four comparisons are needed to check if T$\in$[ACCESS, 127] or T$\in$[0,ACCESS+D_{expire}-128]. We will use four comparitors to perform these comparisions simultaneously. Since D_{expire} is usually between 4 to 6 seconds in our architecture and each "round" is 16 seconds, we estimat that when the cache is in active use, the chance for a "very old" (more than 16 seconds old) flow to be identified as an active flow is infinitestimal. Similar coding is used for EXPIRE. However, the increment will be 1 every 2 seconds and $D_{lifespan}$ is typically 120 seconds. So a cache entry that is more than 2 minutes old will be considered "dead". Since each "round" in this case is 256 seconds, we estimated that the the chance for a "dead man walking" senario is infinitestimal.