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

Jun Xu (Dept. of CIS) Mukesh Singhal (Dept. of CIS) Joanne Degroat (Dept. of EE)
The Ohio State University, Columbus, OH 43212, USA
{jun,singhal}@cis.ohio-state.edu degroat@ee.eng.ohio-state.edu

Abstract— **Existing and emerging layer-4 switching technologies require packet classification to be performed on more than one header fields, known as layer-4 lookup. Currently, the fastest general layer-4 lookup scheme delivers a throughput of 1 Million Lookups Per Second (MLPS), far off from 25/75 MLPS needed to support 50/150 Gbps layer-4 router. We propose the use of route caching to speed up layer-4 lookup, and design and implement a cache architecture for this purpose. We investigated the locality behavior of the Interent traffic (at layer-4) and proposed a near-LRU algorithm that best harness this behavior. In implementation, to best approximate fully-associative near-LRU using relatively inexpensive set-associative hardware, we invented a dynamic set-associative scheme that exploits the nice properties of N-universal hash functions. The cache architecture achieves a high and stable hit ratio above 90 percent and a fast throughput up to 75 MLPS at a reasonable cost ($700/1700 for 50/150 Gbps router).**

Keywords— **Layer-4 Switching, Packet Classification, Route Caching, Set-Associative Cache, Parallel Hashing, Universal Hash Functions.**

I. Introduction

Existing and emerging layer-4 switching technologies such as packet-filtering firewall, RSVP, differentiated services, QoS routing, Virtual Private Networking (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. This problem is known as *layer-4 packet classification* or *layer-4 lookup* [1], [2]. The motivation behind layer-4 packet classification and its various applications are well documented in [1] and [2]. Today, exponential growth in demand for Internet bandwidth has driven the throughput of Internet routers to 50 Gbps [3]. To support such a high bandwidth at layer-4, packet classification needs to be performed at 25 Million Lookups Per Second (MLPS), assuming an average packet size of 2000 bits. However, the fastest general scheme for layer-4 lookup [1] only delivers 1 MLPS, which is only about 4% of what is needed for a 50 Gbps layer-4 router.

As a high degree of *temporal locality* can be observed at transport layer [4], [5], [6], we propose to employ route cache to improve the speed of layer-4 lookup. We designed a layer-4 route cache architecture that achieves a high and stable hit ratio of 92% with only 0.6% variation. This is achieved using our novel cache management algorithm called *near-LRU* that best exploits the *locality behavior* exhibited by the Internet traffic at layer-4 [4], [5]. However, near-LRU is still a fully-associative algorithm which is not scalable to large cache size. Using traditional set-associative hardware to approximate near-LRU may result in a large amount of *collision misses*. For better approximation, we 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 75% to 90%. The cache architecture is built upon an emerging DRAM technology called SLDRAM [7], [8] that achieves the same high throughput as SRAM at the unit price of DRAM. The throughput of our cache architecture is 22.5/45 MLPS using 400/800 MHz SLDRAM technology. Using our *lazy-writeback* scheme, this throughput is further improved to 37.5/75 MLPS, which is able to support 75/150 Gbps layer-4 routers. The VLSI implementation of the cache architecture is laid out and is ready for fabrication. The cost of such a cache architecture is in the worst case 700/1700 dollars for a 50/150 Gbps router.

This paper is organized as follows. Section II formulates the problem of layer-4 packet classification. Section III presents the issues and challenges with layer-4 caching. Section IV proposes the concept of near-LRU cache. Section V presents the design and implementation of the proposed cache architecture. Section VI describes how dynamic set-associative scheme helps approximate near-LRU through detailed performance analysis. Section VII proposes the lazy-writeback scheme that can dramatically improve the throughput of the cache. Section VIII concludes the paper.

II. Layer-4 Lookup and Related Works

The problem of layer-4 packet classification can be formulated as follows. The forwarding table of a layer-4 router typically consists of a large number of forwarding rules called *filters*. The schema of the forwarding table consists of d+1 attributes, among them d attributes are used as the key for search. They correspond to the packet header fields based upon which a forwarding decision is made. The last 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*, or *range match* [2]. When a new packet arrives, its corresponding header fields are matched with the filters using all three types of matches if applicable. A packet may match multiple filters and hence the ambiguity. This ambiguity is resolved 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 [2].

In layer-4 applications like firewalling, RSVP, DiffServ, and QoS routing, the five-tuple <destination-IP, source-IP, 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*. In firewalls, the ACK bit of TCP flags also participates in making the forwarding decision [9]. However, 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 such as VPN and multicasting use <destination-IP, source-IP> as the search key. That is a special case of packet classification based on layer-4 address and is much simpler [1], [2]. This paper will focus on the general case: 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* [1]. The latter is in turn equivalent to the *point location problem* in computational geometry, which is to find the object that a point belongs to among L d-dimensional objects[1]. 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 $O(log(L))$ computation, but needs $O(L^d)$ working space [1]. On the other hand, the best algorithm in terms of space complexity has a memory requirement of $O(L)$, but needs $O(log^{d-1}(L))$ computation time [1]. When N is as large as tens of thousands and d is as large as 5, neither algorithm is realistic [1].

A few solutions were proposed recently for packet classification using layer-4 address. Using a hardware implementation, general layer-4 lookup scheme proposed in [1] is able to perform a worst case of 1 MLPS against a few thousand filters. Srinivasan et al. [2] proposes *extended grid of tries* that handles a special case in which port numbers are either wildcards or an exact number. Their *Cross-producting* algorithm aims to solve the general 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. A caching-like (different from route caching) approach, "on-demand cross-producting" [2] was proposed to solve the scalability problem but requires "non-deterministic classification time" [10]. Recursive Flow Classification (RFC) proposed in [10] achieves a lookup speed of 30 MLPS using pipelined hardware implementation. However, it does not appear to be a general lookup scheme because its high performance is achieved on real-life enterprise firewall filtering tables by exploiting their inherent "structure and redundancy." It is not clear whether this exploitation will continue to be possible in other layer-4 switching applications and/or in backbone routers where forwarding rules can be much more random (less structure-rich).

III. L4 Route Cache: 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 [4], [5], [6]. This can be explained by the fact that a network object such as a file or a homepage is broken into a number of packets with the same layer-4 address for transit. This 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.

Similar locality behavior was observed at layer-3 [11] and caching has been used in commercial routers to speed up the IP lookup [12], [3]. However, recent studies show that the hit ratio of layer-3 route cache tends to be low (60% to 80%) and unstable [12], but none of them seriously explains why. After an in-depth study of the route caching practices at layer-3 [12], [3], we discovered that this poor performance is caused by the way route cache is implemented rather than the caching approach per se. Current layer-3 route cache all uses L1 cache of a general-purpose CPU, which is the control processor of the router. For obvious economical reasons, the design of general-purpose CPU and its L1 cache is targetting computer manufacturers, not router designers. L1 cache is not ideal for route cache due to two major problems. First, L1 tends to be small in size (typically less than 100k) so that when the number of concurrent active flows is large, which is often the case in backbone routers, the hit ratio tends to be low due to "thrashing" among different flows/sessions. Use of larger L2 cache does not solve this problem because the huge block size of L2 is too wasteful for caching layer-3 routes. In general, a block can only cache one address due to lack of spatial locality between consecutive addresses, e.g., a packet destined for 164.107.60.88 is not necessarily followed by a packet destined for 164.107.60.89 in the near future. Second, the program memory reference and IP address reference (viewed as 32-bit virtual address) have totally different locality behavior, and L1 cache caters to the former. L1 cache is set-associative which typically uses random replacement among different sets. This is a right choice for CPU caching in which relatively expensive LRU outperforms random replacement very little [13]. However, for route caching, our simulation study found that LRU significantly outperforms random replacement in terms of miss ratio at both layer-3 and layer-4. Our layer-4 cache architecture is designed in such a way that it will not "inherit" either problem: it employs decent amount of DRAM so that the cache size is not an issue; it employs our near-LRU cache replacement algorithm which achieves almost the same low miss ratio as LRU (provably optimal) with much lower implementation complexity.

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 (or possibly one) *backup packet classifiers*. When a packet arrives at a port of a router, the line card at that port will extract its layer-4 address and put it in a route lookup re-

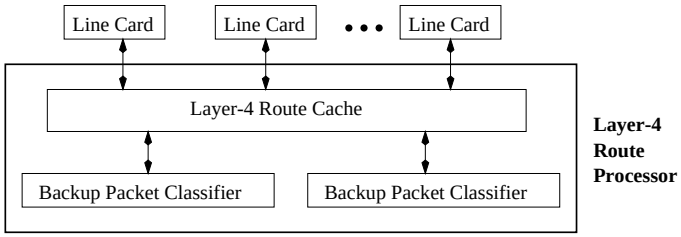

Fig. 1. System Model of Our Layer Caching Approach

quest to be sent to the route processor. The route processor will first search the route cache for a match. If a hit occurs, the corresponding forwarding decision will be sent back to the line card immediately. Otherwise, the packet will be forwarded to a backup packet classifier. The backup packet classifier will perform a full-fledged layer-4 lookup using the fastest general layer-4 packet classification scheme. The result will be sent to the line card and saved into the route cache. Let H_{backup} be the total throughput of all *backup packet classifiers*, R be the miss ratio of the route cache, and H_{cache} be the throughput of the route cache. Then the lookup throughput of the route processor is $H_{processor}=MIN(H_{cache}, \frac{H_{backup}}{R})$. 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.

This system model is most cost-effective for layer-4 routers that employ *centralized route processing* [14], in which all lookup requests from each and every line card are handled by one route processor, for two reasons. First, the 50/150 Gbps lookup throughput of the route cache can be fully utilized in the central route processor of a 50/150 Gbps router. Multiple route cache modules can work in parallel if higher bandwidth is needed. Second, when the amount of layer-4 switched traffic at each port is not evenly distributed, centralized approach achieves the best utilization of the lookup engine by statistically multiplexing the lookup requests into a single stream. A design alternative to centralized route processing is to place one route processor on each line card, known as *distributed route processing* (The relative advantages and disadvantages of either approach and the commercial router products that adopt either approach are well discussed in [14].). As the highest per-link bandwidth that will soon be commercially available is only 9.6 Gbps (OC-192), route cache will be underutilized under distributed route processing.

Due to a number of technical challenges, caching is not looked upon as a promising technology in the layer-4 lookup research community. Layer-4 packet classification papers [1], [2] 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 [12]. Lakshman et al. [1] are concerned that the hit ratio may not be stable enough so that when the hit ratio is low, the backup packet classifiers will be temporarily overloaded. We meet this challenge by delivering a cache architecture that achieves 92% hit ratio with only 0.6% variation. We showed above why our layer-4 cache architecture will not "inherit" the hit ratio problem from layer-3 caching practices [12]. Second, it should withstand denial-of-service attack. Lakshman et al. [1] are concerned 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 [1]." We will address this issue in Section VI.A. An additional challenge we pose to ourselves is that 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 [15]. This leads to a "cold start" of the cache periodically and the miss ratio during that period is expected to be high [3]. Our cache architecture will automatically invalidate cache entries that are $D_{lifespan}$ (set to 120s in our cache) seconds old without affecting its miss ratio and throughput.

IV. The Notion of Near-LRU Cache

A. Flow Theory

The concept of a *flow* is introduced in Claffy's Ph.D thesis [4] to characterize the locality behavior of the Internet traffic. 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}. A flow is said to be *active* from the time it was created until the time it expires. Flow theory [4] introduces some important metrics defined over an Internet traffic trace, among them two are critical 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* $F(D_{expire}, t)$. 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 [5], [6]. From the figures and data presented in [5] and [6], we found that the values of F and ρ are almost *stationary* (not a function of t) provided 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 achieving a stable hit ratio under the bombardment of the sustained maximum (but constant) traffic volume U_{max}. The stationarity of these two metrics is further corroborated by our trace-driven simulation shown next.

Table I contains seven FIXWEST Internet backbone traces obtained from NLANR's FTP site. Since the con-

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

TABLE I
THE STATISTICS OF FIXWEST TRACES

cept of layer-4 address is only relevant to TCP/UDP packets, we exclude other packets (less than 10%) from these traces, and packet volume (Avg. Pkts/s) in the table reflects this exclusion. These traces will be used in several measurements and trace-driven simulations throughout this paper. We choose FIXWEST traces because backbone traffic contains much higher number of *active flows* per Megabit of traffic (higher multiplexing level) than traffic in a campus or a corporate gateway, a fact well corroborated in [4], [16], [6]. As the cache performance is much worse under higher multiplexing level (as in backbone router), backbone traces offer a better touch stone that tests how well our cache works. FIXWEST traces, on the other hand, record 100% of the traffic. Among the seven traces listed in Table I, trace 1 contains the heaviest packet/traffic volume and has the longest duration. The simulation and measurement results from this trace are presented throughout this paper. Results from other traces produced similar conclusions.

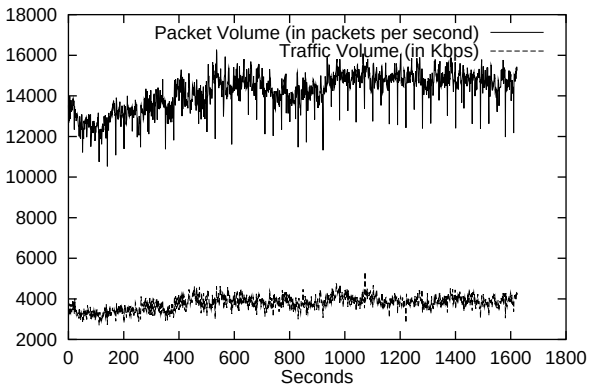

Fig. 2. Packet Volume and Traffic Volume

Fig. 2 shows the traffic volume U and packet volume V over the 1625-second time period. Fig. 3 shows the number of active flows F and the arrival rate of new flows ρ over this period, when D_{expire} is set to 5 seconds. Visually, it can be seen that F is no more bursty than V, and ρ is much less bursty than U. This comparison is fair because we compare the upper/lower curve in Fig. 2 with the upper/lower curve in Fig. 3 so that the curves in comparison are of similar scale. Statistically, $C_X^2 = \frac{Var(X)}{(E[X])^2}$, *coefficient of variation* of a random variable X, measures the *variability* of X [17]. 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. Therefore, the stationarity of both F and ρ is corroborated in this measurement. Other traces offer the same conclusion on this issue.

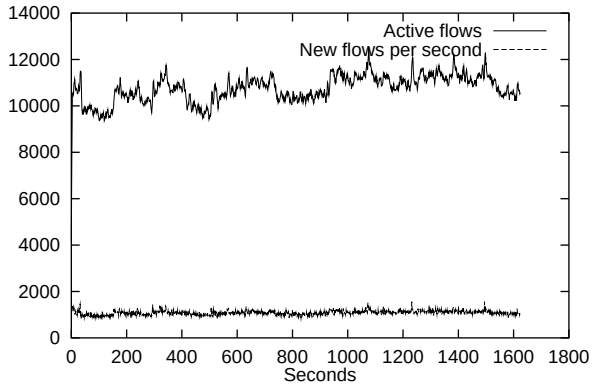

Fig. 3. Active Flows and New Flows Per Second

B. Near-LRU Cache Management Algorithm

The stationarity of F and ρ in the flow theory framework motivates us to propose a cache management algorithm called *near-LRU*, which aims to exactly harness this locality behavior. In near-LRU, each cache entry is associated with a timestamp that is updated whenever this cache entry is accessed. When there is a need for cache replacement, a victim is randomly chosen among cache entries that have not been accessed for more than D_{expire} seconds. Assume 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 because each new flow causes exactly one miss when its first packet arrives. The stationarity of ρ indicates that $R_{near-LRU}$ is fairly stationary. From the same set of data used in Fig. 2 and 3, we measured that $E[R_{near-LRU}] = 9.261\%$ and $\sqrt{Var(R_{near-LRU})} = 0.006044$, that is, near-LRU algorithm achieves a hit ratio of 90.7% with only 0.6% standard deviation! The number of active flows F indicates the approximate number of entries the cache architecture needs to have. The stationarity of F is extremely important for the "well-definedness" of our near-LRU algorithm in the sense that it allows us to provision the cache size for the worst scenario without much waste in the average case.

Near-LRU is actually a near-optimal cache management algorithm in the sense that it achieves almost the lowest hit ratio among all 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 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 [18]. 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 denote the probability of i_{th} stack position (1st position is the stack top) being referenced. 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 the *weak locality condition* at size S is satisfied [18].

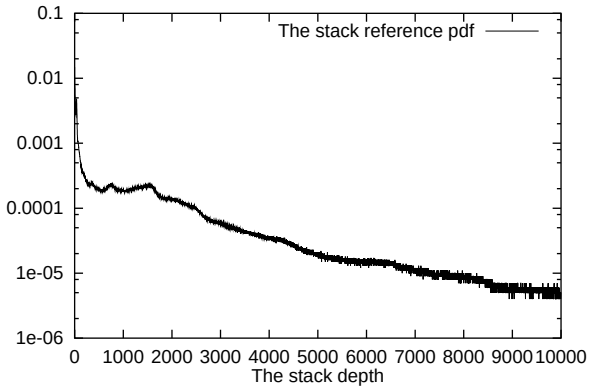

Fig. 4. Stack Reference pdf

Fig. 3 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 of other traces produce the similar results. We conclude that when the cache size is fairly large (> 2000), LRU should be the optimal cache management algorithm.

C. The Growth of F and ρ in the Future

So far, we get exciting results from FIXWEST traces. However, the TCP/UDP traffic volume of the heaviest trace (trace 1) is only 30.8 Mbps. Can the same or higher hit ratio be achieved in the future by near-LRU cache when the traffic volume is as high as 150 Gbps? To answer this question, we need to either test our near-LRU algorithm on traces of much higher traffic volume (if not 50 to 150 Gbps) or prove that our results will continue to hold with much higher traffic volume. Unfortunately, the former is not possible because FIXWEST traces are the heaviest backbone traces that records 100% of the back-to-back traffic and are publicly available to us. So, we resort to the latter alternative. We 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. So the hit ratio will be higher in the future if the average amount of the traffic per flow is larger. Using the same insights in arguing for the packet train model [19], a flow is typically a series of packets that are transporting a network object such as an file or a 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. Therefore, the near-LRU cache should produce an equal or higher hit ratio in the future.

Now that the same high hit ratio can be guaranteed in the future, then how many cache entries do we need to achieve that? It is worth noting that the number of active flows F actually reflects the number of active users (the multiplexing level). 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). Since it is obvious that future Internet users should 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. In other words, though the *absolute multiplexing level* (F) increases with the traffic volume, the *relative multiplexing level* ($\frac{F}{U}$) is not. We found through our simulation that 10,000 entries are needed for FIXWEST trace 1 (30.8 Mbps) traffic to achieve a hit ratio over 90%. Note that this number is much smaller than the number of active flows reported in [6] and [16] in traffic stream of similar bandwidth. The difference is due to the fact that they assumed a timeout value of 64 seconds while we use a timeout of approximately 5 seconds, which is good enough for achieving a decent hit ratio. As we have shown that the number of active flows grows at most linearly with the traffic volume, a 50/150 Gbps layer-4 router will in the worst case need 16/48 million entries of cache. As each layer-4 cache entry is 128-bit long (explained later), we are looking at 256/768 Mbyte cache memory in the worst case.

In summary, we showed that we can continue to achieve the same or higher hit ratio in the future when the traffic volume is thousands of times higher, at the cost of larger amount of cache memory which is at most proportional to the increases of the traffic volume.

V. Proposed Cache Architecture

Generally speaking, a cache is chunks of RAM augmented by control 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 series of accesses to consecutive memory locations). 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 data every 2.5/1.25 ns and therefore can read/write a 128-bit layer-4 route every 20/10 ns. Inside a SLDRAM chip there are 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 [8], we expect that the price of the SLDRAM will drop to about 2 dollars per Mbyte (The price of DRAM is now about 1.5 dollars per Mbyte) in the near future. As in the worst case 256/768 MB route cache is needed for a 50/150 Gbps router, we are looking at a memory cost of 500/1500 dollars.

The architecture of the route cache is shown in Fig. 5. It consists of N independent SLDRAM banks, each storing $M = 2^r$ entries. When a lookup request arrives, the 97-bit layer-4 address in the request will be processed by N different hardware hash functions in parallel to produce N r-bit hash values and $(97-r)$-bit tags. 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 $(97-r)$-bit tag, a 32-bit forwarding decision ("next-hop") field, and a 16-bit "timestamps" field. As r is typically between 20 and 24, we can fit the whole record into 16 bytes while reserving three to seven excess bits for control signals. The "tag" fields in these entries will be compared with the tag values outputed from the hardware hash functions in parallel. If one entry matches, we have a hit and its "next-hop" field will be output as the final "next-hop" result from the multiplexer. Otherwise, we have a miss, which is handled by the following approximation of near-LRU replacement policy.

The "timestamps" field consists of an 8-bit timestamp LIFESPAN that records when the entry was created, and an 8-bit timestamp EXPIRE that records when the entry was last updated. LIFESPAN is used to check whether the entry is more than $D_{lifespan}$ seconds old, which indicates that the cache entry should no longer be trusted. EXPIRE is used to check whether the entry has expired, which indicates that it is a potential candidate for replacement. The value of EXPIRE field is incremented by 1 every $\frac{1}{8}$ second in $0\rightarrow1\rightarrow2\rightarrow\cdots\rightarrow255\rightarrow0\rightarrow\cdots$ rounds. To check if an entry expires is to check if T $\in [EXPIRE, EXPIRE+D_{expire}]$ (T is the current time), when the threshold 255 is not crossed, or to check if $T \geq EXPIRE$ or $T \leq EXPIRE + D_{expire} - 256$, when the threshold is crossed. Since D_{expire} is usually between 4 to 6 seconds in our architecture and each "round" is 32 seconds, we estimate that when the cache is in active use, the chance for "ambiguity", in which a "very old" (more than 16 seconds old) flow to be identified as an active flow, is infinitesimal. EXPIRE is incremented by 1 every 2 seconds as it needs longer representable time duration but can tolerate less precision. These checks are performed by N Preprocessing (PP) modules in parallel and the results are fed to the replacement logic. If at least one entry expires, one among the expired entries will be chosen for replacement. Otherwise, we have a "collision," and one among the N entries will be randomly chosen as victim. This random replacement is emulated in the replacement logic module as follows. A $log(N)$-bit state vector records the index (say I) of the bank that is most recently replaced. The replacement logic is to to search the corresponding cache entries in bank I+1, I+2, ..., and I+N (modulo N) in sequence, and the first one that expires will the replaced. If no banks expire, the item in bank I+1 will be replaced. The state vector will be updated with the index of the bank that the replaced entry belongs to. This logic can be implemented using a $(N+log(N))*2^N$ ROM. In our simulation study, we found that this emulated randomness is indistinguishable (in terms of hit ratio) from the true random replacement.

In our cache architecture, the number of independent memory banks N is 8. The cache architecture can be implemented as a printed circuitry board attached to a port of the switching fabric. As the implemenation of other components (chips) such as equal-only comparators, replacement logic and PP modules (merged into one chip), and multiplexer is straightforward, we only need to explain the implmentation of the N hash functions. It will be clear in Section VI.A that these N hash functions can be made out of N identical generic chips (by initializing their internal state registers to different values), and each such generic chip is very amenable to hardware implemenation. We estimated that a 97-in 97-out generic chip occupies an area of 2.5mm by 2.5mm using 0.25 μm CMOS process and its cost is no more than 10 dollars (including packaging) assuming reasonable level of mass production (e.g., 3000+) [20]. We estimated that the total cost of the control logic is within 200 dollars. So the total cost of the cache architecture is in the worst case 700/1700 dollars (for 50/150 Gbps router) including the cost of memory. As such a cost is per-router (not per port), it is justified for a 50/150 Gbps layer-4 router.

We have shown in Section III that the desired speedup $\frac{1}{R}$ can not be achieved by caching if the throughput of the cache, H_{cache}, becomes a bottleneck. We estimated that the propagation delay along the critical path of the control logic is well below 10ns if implemented using commodity VLSI technology (e.g., 0.25 μm CMOS process), the time needed for 800 MHz SLDRAM to access a 128-bit layer-4 route. With proper pipelining, the hashing, comparison, and replacement operations can be done in parallel with the memory read/write. Therefore, the lookup operation can be performed as fast as the bandwidth of SLDRAM divided by the number of memory accesses needed for a lookup. One read is needed for each layer-4 lookup. If a hit occurs, one write is needed to update the EXPIRE bits. If a miss occurs, two writes are needed to write an "interim entry" and a final entry. The interim entry, which is written into the entry to be replaced before the lookup result from the backup classifier is available, avoids write/write race condition by indicating that "the forward-

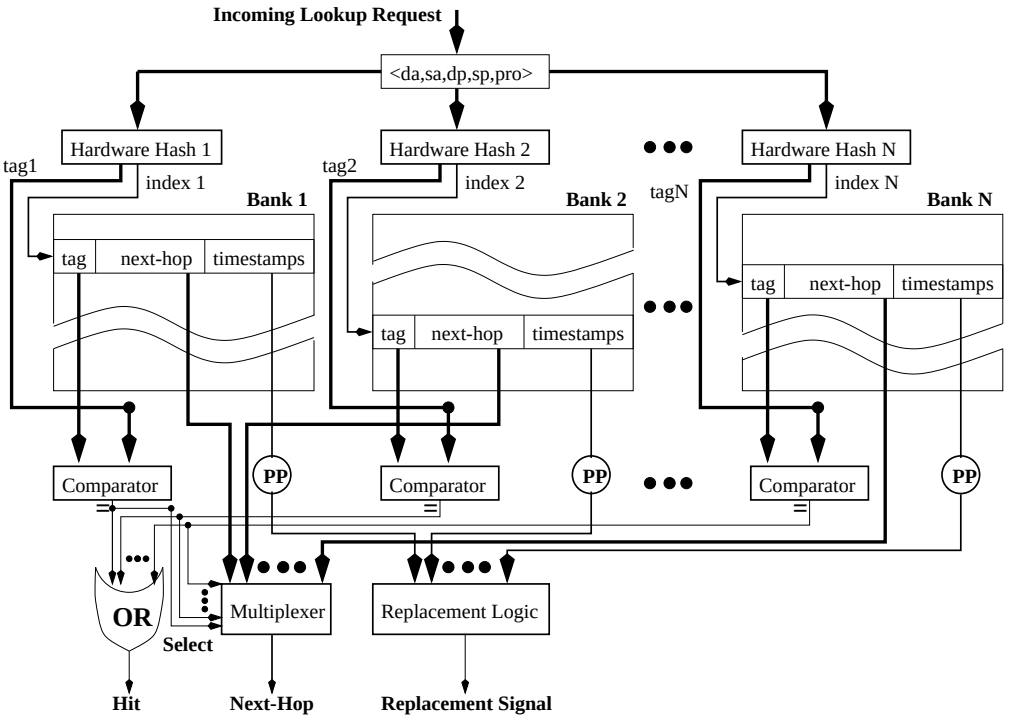

Fig. 5. Hardware Design of Our Caching Approach

ing decision is in search." 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 46% of the available throughput of the memory. 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. Using our lazy-writeback strategy (described in Section VII), this throughput can be further improved to 75/150 Gbps.

VI. Dynamic Set-Associative Cache

A. The Conceptual Overview

Our cache architecture would be a regular N-way set-associative scheme (called *static set-associative scheme*) if the N hardware hash functions were the same. However, as we will show, static set-associative cache will bring about a high volume of *collision miss* when its load is high. We cut the collision miss down by 75% to 90% using our novel *dynamic set-associative scheme*, in which these N hash functions $h_1, h_2, \cdots, h_N$ satisfy the following property. Given a random hash key X, $h_1(X)$,$h_2(X)$,$\cdots$,and $h_N(X)$ are identical independent uniformly distributed random variables. In hashing literature, this type of hash function is called *N-universal hash function* [21]. It is well established that N hash functions randomly chosen from a function class called H_3 [22] are N-universal hash functions. A fair amount of measurement is done in the literature [21], [23] to demonstrate that the actual performance of such hash functions on real life data as well as on random data satisfies the N-universal property. Also, it was shown in [23] that H_3 hash functions are very amenable to hardware implementation.

Each hash function in H_3 class is a linear transformation $B^T = QA^T$ 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:

$$\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_r \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1w} \\ q_{21} & q_{22} & \cdots & q_{2w} \\ \cdots & \cdots & \cdots & \cdots \\ q_{r1} & q_{r2} & \cdots & q_{rw} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_w \end{pmatrix} \quad (1)$$

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 $r \times w$ matrix defined over GF(2) and each hash function in H_3 is uniquely corresponding to such a Q. So each hash function in the class can be configured from a generic chip by initializing its internal state registers to the row vectors of 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:
$b_i = (a_1 \circ q_{i1}) \oplus (a_2 \circ q_{i2}) \oplus \cdots \oplus (a_w \circ q_{iw}) \quad i = 1, 2, \cdots, r$

The tag is obtained as follows. Denote the row vectors of Q as V_i, $i = 1, 2, \cdots, r$. We restrict the choice of V_i so that they are *linearly independent*. This is not a severe restriction because the probability for r vectors randomly chosen from $\{0,1\}^w$ being *linearly independent* is $\prod_{i=0}^{r-1}(1 - \frac{2^i}{2^w}) \geq 1 - \sum_{i=0}^{r-1} \frac{2^i}{2^w} \geq 1 - 2^{r-w}$, $(1 - \frac{2^i}{2^w})$ being the probability that v_i does not belong to the vector space spanned by $\{v_1, v_2, \cdots, v_{i-1}\}$. (The first inequality is a special case of Bernoulli's inequality.) In our system design, where $w = 97$ and r is typically between 20 and 24, this probability is greater than 0.999999. Now that they are linearly independent, we can expand these r vectors into

a basis V_1, V_2,$\cdots$,V_r,V_{r+1},V_{r+2},$\cdots$,V_w in the vector space $\{0,1\}^w$. Let $\overline{Q}$ be the $(w-r)*w$ matrix, the i_{th} row of which is V_{r+i}, i=1,2,...,$w-r$. Then $\overline{B}^T = \overline{Q}A^T$ is the tag. This procedure has not been reported in hashing literature because H_3 hash functions have never been used for caching.

The novelty of the dynamic set-associative scheme is best assessed by comparing with existing parallel hashing schemes that explicitly or implicitly take advantage of the nice property of N-universal hash functions [21], [24], [25], [26]. The scheme in [21], which is closest to ours, employs the same hash functions (H_3) and is interested in the same metric (number of collisions). However, our scheme is fundamentally different from [21] in two ways. First, the targeted application of [21] is *dictionary* in which each entry (e.g., English word) has to be inserted without evicting the existing entries. So it has to rehash the existing entries when a collision occurs, an issue our scheme does not need to address. Second, scheme in [21] operates in *prioritized mode* while our scheme operates in *random mode*. We will explain both modes in Section VI.C and show that they have totally different performance characteristics. Schemes in [24], [25], [26] are so different from ours in operation settings and targeted metrics that a meaningful comparison requires far more space than we can afford here. In summary, the novelty of our scheme is reflected in three aspects. First, our scheme is not a simple application, adaptation, or modification of any of these schemes. Second, to the best of our knowledge, no performance modeling that is similar to that of our scheme (for the random mode) has appeared in literature of any sort. Third, no close scheme has been proposed for the caching purpose.

The dynamic set-associative scheme is much more robust than static set-associative cache in withstanding the denial-of-service 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 hashed into these sets will undergo severe thrashing. However, with 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.

B. Performance Analysis

Now we are in a position to show how *dynamic set-associative scheme* achieves a lower miss ratio than *static set-associative scheme*. In both dynamic and static schemes, the total miss ratio R can be broken down as R=R_{new}+$R_{collision}$-$R_{stealing}$. R_{new} refers to the misses caused by the arrival of the new flows that do not cause a collision (at least one among N replacement candidates expires). The value of R_{new} is slightly smaller than $\frac{1}{\sigma}$ (σ is the average number of packets in a flow). The difference between them is explained by the fact that some cache misses that causes collisions are actually new flows (should be counted in R_{new}), but are recorded in $R_{collision}$ instead. $R_{collision}$ refers to the portion of miss ratio that is caused by collision, which our scheme tries to minimize. In our cache, when an entry expires, there is a high probability for it to be replaced by a new entry. However, there is still a certain chance for it to be accessed again 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 $R_{stealing}$. Through both performance analysis and simulation study, it can be shown that under the same system parameters, R_{new} and $R_{stealing}$ are almost the same in static scheme as in dynamic scheme.

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

TABLE II
THE RELATIONSHIP BETWEEN E[O] AND C

Dynamic set-associative scheme achieves a much smaller miss ratio than static set-associative scheme because it cuts $R_{collision}$ by 75 to 90 percent. We demonstrate this in two steps. In the first step, we show (till the end of this paragraph) that in both dynamic and set-associative schemes, $R_{collision}$ is the same linear function of a variable Γ. In this step, the discussion and calculation is unified for both static and dynamic cases. In the second step, we show this Γ is much smaller under dynamic set-associative scheme ($\Gamma_{dynamic}$) than under static set-associative scheme (Γ_{static}). Let λ denote the number of collisions that occurs in the cache 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 a set-associative cache (dynamic or static) and let O be the average number of collisions that occurs in the processes. We can see that O actually represents the average number of active flows that are not cached at any given time. We further find that there is a linear relationship between O and λ, that is, $\lambda = cO$, c being called the *collision rate*. This fact is indirectly proved in CPU caching literature [27] and is corroborated by our experimental results from FIXWEST trace 1. We can see from Table II that c almost remains constant 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 F, that is, $\Gamma = \frac{O}{F}$. Then $R_{collision}$ is expressed as $R_{collision}$= $\frac{\lambda}{V}$=$\frac{cO}{V}$=$\frac{cF\Gamma}{V}$=$\left(\frac{cF}{V}\right)\Gamma$. In both dynamic and set-associative schemes, $\frac{cF}{V}$ is a constant when the traffic trace and system parameters like 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 $R_{collision}$ is about 30 to 50 percent of Γ.

Now we show that under the same load ratio α, defined as $\alpha = \frac{F}{MN}$, $\Gamma_{dynamic}$ is much smaller than Γ_{static}. Both $\Gamma_{dynamic}$ and Γ_{static} are found to be a function of N (number of banks) and α, but not a function of M

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

TABLE III
$\Gamma_{dynamic}$ VS. Γ_{static}

(size of a bank). Γ_{static} is estimated as follows. Let Y be the random variable that denotes the number of entries mapped to an arbitrary row of N entries with the same index (called a *set*). When M is fairly large (> 1024), P can be approximated by Poisson distribution $P[Y = i] = \frac{(N\alpha)^i exp(-N\alpha)}{i!}$. Since the collision ratio from an arbitrary set (randomly chosen) is exactly the overall collision ratio, $\Gamma_{static} = 1 - \sum_{i=0}^{N} \frac{(N\alpha)^i exp(-N\alpha)}{i!}$.

$\Gamma_{dynamic}$ is estimated as follows. Let us assume $p(x, s)$ denotes the probability that x data items are put into cache and s items settle (do not cause a collision). Then assume that all the items that settle is evenly distributed in the N banks. We have the following recurrence condition: $p(x,s) = \left(\frac{s}{NM}\right)^N p(x-1,s) + \left[1 - \left(\frac{s-1}{MN}\right)^N\right] p(x-1,s-1)$. 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} item 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. Then $\Gamma_{dynamic} = \frac{E[O]}{F} = \sum_{s=1}^{F} \frac{(F-s)p(F,s)}{F} = 1 - \sum_{s=1}^{F} \frac{sp(F,s)}{F}$. As this recurrence condition is complex, we can not get a closed form expression for Γ. Fortunately, $p(F, s)$, 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 III shows $\Gamma_{dynamic}$ and Γ_{static} under various load conditions (from 0.5 to 0.9). The theoretical results are calculated using aforementioned equations. 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. Randomly-generated 97-bit layer-4 addresses are used as hash keys. Ten experiments were conducted to obtain 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 deviation is very small. This independently corroborates the N-universal property of H_3 hash functions. From Table III, 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 $\Gamma_{dynamic}$. This is to say that the collision miss in the dynamic scheme is only 25% of the collision miss ratio in the static scheme!

The benefit of using near-LRU algorithm and dynamic set-associative cache becomes obvious when we simulated the miss ratio of our cache architecture under FIXWEST trace 1. The cache architecture consists of $N = 8$ banks with $M = 2048$ entries each, and D_{expire} is set to 5.3 seconds. We obtained a total miss ratio of 8.0 percent with a standard deviation of only 0.6%. This miss ratio is very close to the theoretical minimum miss ratio 7.7 percent that is achieved using 16384 ($16834 = 8 \times 2048$) entries of LRU cache. In contrast, simulation on an 8-way 2048-set static set-associative cache with random replacement policy produced a 11.9 percent miss ratio, which is almost 50% higher.

C. Design Alternatives and Parameter Tuning

Numerous design alternatives were attempted and their performance measured/simulated before we settled with the design choices made in this architecture. Once we had made these design choices, we then needed to finely tune the system parameters to achieve the highest and most stable hit ratio possible. In this section, we present the rationale behind one such design choice and an example of parameter tuning.

(1) We explained that when there is a cache miss and there are more than one entries in the current cache set that have expired, the victim is randomly chosen from these entries. We call this mode of operation *random mode*. An alternative is to choose victim from the lowest index among this set, which we call *prioritized mode*. Recall that the objective of dynamic cache is to reduce Γ, which ultimately leads to the reduction of $R_{collision}$. We can show that prioritized mode leads to even smaller Γ (we prove that it is actually optimal) than random mode. For example, using the same data set as used to generate Table III, prioritized mode can further reduce the Γ by 75%. So at the early stage of our study, we expect that prioritized mode would produce lower miss ratio, only to be defied by the simulation results. The reason is that on the one hand, a large portion of negative miss ratio comes from $R_{stealing}$, hits produced by expired (but not dead) cache entries. The stealing ratio suffers in the prioritized mode because the replacement activity is now concentrated in low-index banks so that an expired entry is much less likely to steal a hit. On the other hand, the saving from the prioritized mode is tiny because the collision miss in the random mode has already been very small.

(2) D_{expire} is an important parameter that needs to be finely tuned. D_{expire} directly affects F, the number of active flows. We have shown that $\frac{F}{MN}$ is the load factor. If this load factor is too large (close to 1), the miss ratio will be high because there will be too many collisions. If this factor is too small, the miss ratio will also be high because the cache will not be making the best choice in cache re-

placement. The parameter tuning is to find the best F in the middle. D_{expire} is set accordingly to achieve this F.

VII. Lazy-Writeback

We have seen in Section V 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 EXPIRE bits. However, if a series of packets (say 10) hit a layer-4 route entry within a very short period ($<< D_{expire}$), there is really no need to perform a *writeback* every time a hit happens. A writeback is needed only after the 1st and the 10th 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 a 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, $M = 2048$) that are used in the last section. We set the $D_{writeback}$ to 1 second. We found 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 now improves to 37.5/75 MLPS, enough to support 75/150 Gbps layer-4 routers.

VIII. Summary

We designed and implemented a novel layer-4 route cache architecture to support layer-4 lookup at memory access speeds. It implements our near-LRU cache management algorithm using our novel dynamic set-associative scheme. We demonstrated through trace-driven simulation that near-LRU algorithm is able to achieve high and stable hit ratio and is near-optimal. We showed that the same high and stable hit ratio can be achieved in the future when the Internet traffic volume becomes much larger, at the cost of larger cache size that grows at most linearly with the traffic volume. To best approximate fully-associative near-LRU using relatively inexpensive set-associative hardware, we invented a dynamic set-associative scheme that exploits the nice properties of N-universal hash functions. Using both statistical analysis and simulation study, we showed that dynamic set-associative scheme cuts the collision miss ratio of traditional set-associative cache by 75% to 90%. We described a VLSI implementation of the cache architecture and analyzed the function, performance, and hardware complexity of the components. Through our simulation study using FIXWEST traces, we demonstrated that this architecture achieves 92% hit ratio with only 0.6% standard deviation. Using our lazy-writeback scheme that further improves the throughput by 70% percent, the cache architecture can support 75/150 Gbps (using 400/800 MHz SLDRAM) layer-4 switching at a worst-case cost of 700/1700 dollars. In conclusion, this cache architecture demonstrates that layer-4 caching is a viable and cost-effective solution for supporting high-speed layer-4 switching.

References

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

[2] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *Proc. of ACM SIGCOMM'98*, Sept. 1998.

[3] C. Partridge et al., "A 50-gb/s ip router," *IEEE/ACM Trans. on Networking*, vol. 6, no. 3, pp. 237–248, June 1998.

[4] K. Claffy, *Internet Workload Characterization*, Ph.D. thesis, UC San Diego, June 1994.

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

[6] K. Thompson, G. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE Network*, pp. 10–23, Nov. 1997.

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

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

[9] W. Cheswick and S. Bellovin, *Firewalls and Internet Security*, Addison Wesley, Reading, MA, 1994.

[10] P. Gupta and N. Mckeown, "Packet classification on multiple fields," in *Proc. of ACM SIGCOMM'99*, Sept. 1999.

[11] R. Jain, "Characteristics of destination address locality in computer networks: A comparison of caching schemes," *Computer Networks and ISDN Systems*, , no. 18, pp. 243–254, 1990.

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

[13] D. Patterson and J. Hennessy, *Computer Architecture, A Quantative Approach*, Morgan Kaufmann, San Francisco, CA, 1996.

[14] H. Chan, H. Alnuweiri, and V. Leung, "A framework for optimizing the cost and performance of next-generation ip routers," *IEEE JSAC*, vol. 17, no. 6, pp. 1013–1029, June 1999.

[15] C. Labovitz, G. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Trans. on Networking*, vol. 6, no. 5, pp. 515–528, Oct. 1998.

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

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

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

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

[20] Semiconductor Industry Association: Technology Needs, *The National Technology Roadmap for Semiconductors*, 1997.

[21] A. Broder and A. Karlin, "Multilevel adaptive hashing," in *Proc. of First Symposium of Discrete Algorithms*, 1990, pp. 43–53.

[22] J. Carter and M. Wegman, "Universal classes of hash functions," *J. of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[23] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Trans. on Computers*, vol. 46, no. 12, pp. 1378–1381, Dec. 1997.

[24] E. Goto, T. Ida, and T. Gunji, "Parallel hashing algorithms," *Information Processing Letters*, vol. 6, no. 1, Feb. 1977.

[25] K. Hiraki, K. Nishida, and T. Shimada, "Evaluation of associative memory using parallel chained hashing," *IEEE Trans. on Computers*, vol. 33, no. 9, pp. 851–855, Sept. 1984.

[26] P. McKenney, "High-speed event counting and classification using a dictionary hash technique," in *Proc. of ICPP'89*, 1989, pp. 71–75.

[27] A. Agarwal, M. Horowitz, and J. Hennessy, "An analytical cache model," *ACM Trans. on Computer Systems*, vol. 7, no. 2, pp. 184–213, May 1989.