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

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 away 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. 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 as well. 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, 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 million layer-4 lookups per second, which is about 4% of that 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 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 [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 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. Section 5 presents the design and implementation of the proposed cache architecture. Section 6 describes how dynamic set-associative scheme helps approximate near-LRU through detailed performance analysis. Section 7 proposes the lazy-writeback scheme that can dramatically improve the throughput of the cache. Section 8 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 key for search. They correspond to the packet header fields based upon which the 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, Diff-Serv, 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 like 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]. 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* [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. 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 $O(lgL)$ 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 $O(lg^{d-1}L)$ 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 new solutions were proposed recently for packet classification using layer-4 address. Using a hardware implementation, general layer-4 lookup scheme proposed in [1] was able to perform a worst case of 1 MLPS against a few thousand filters. Srinivasan et al. [2] proposed *extended grid of tries* that handles the 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" was proposed to solve the scalability problem but requires "non-deterministic classification time"[10]. Using pipelined hardware implementation, 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 (so will not drive us out of business) because it its high performance is achieved on real-life enterprise firewall filtering table by exploiting their inherent "structure and redundancy." It is questionable whether this exploitation will continue to be possible in other layer-4 switching applications, or in backbone routers where forwarding rules are much more random (less structure-rich) due to much higher multiplexing level.

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.

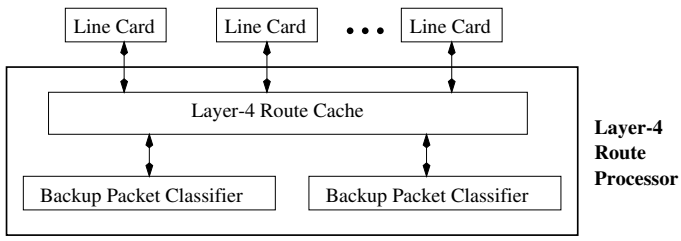
The only existing work on layer-4 caching that was reported in literature is ATLAS [11], an ATM packet filtering firewall operating at the rate of OC-3c (155 Mbps). A 128-bit-wide Content Addressable Memory (CAM) is used to cache the recently-used layer-4 ad-

dresses 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[12]. 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.

Similar locality behavior was observed at layer-3 [13] and caching has been used in commercial routers to speedup IP lookup[14], [3]. However, the days of caching at layer-3 is numbered [3] due to two major challenges. Fortunately, our layer-4 caching scheme will not "inherit" either. The first challenge is that the recently developed pipelined hardware schemes like [15], [16] can perform IP lookup in one memory access, as fast as a cache hit! Layer-4 caching does not "inherit" this problem because the speed of *general* layer-4 lookup scheme without caching (1 MLPS) is still much slower compared to that can be achieved with caching (up to 75 MLPS). The second challenge is that the hit ratio of known layer-3 route caches tends to be low (60% to 80%) and unstable. After a study of these route caches, we found that they were all L1 caches on general-purpose CPUs. For obvious economical reasons, the design of L1 cache is optimized for CPU caching, not for route caching. L1 cache tends to be small in size so that when the number of concurrent 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[1]. Also, the program memory reference and IP address reference (viewed as 32-bit virtual address) has 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. 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 caching won't inherit this hit ratio problem either. In our cache architecture, a huge amount of DRAM is used so that the cache size is not an issue. Also, 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.

[1]In general, a block can only cache one address due to lack of spatial locality between consecutive addresses(e.g., 164.107.60.88 and 164.107.60.89).

Fig. 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 (or possibly one) *backup 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 a backup packet classifier. The backup packet classifier will perform a full-fledged layer-4 lookup using the fastest general-purpose 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. For example, if the miss ratio is as low as 8%, with two 1 MLPS backup classifiers (e.g., scheme proposed in [1]), the lookup throughput of the route processor can be as high as 25 MLPS, enough to support a 50 Gbps layer-4 router.

In this system model, centralized route processing is assumed in which lookup requests from line cards are handled by one route processor. Centralized route processing is criticized for wasting a decent portion (25% as reported in BBN's 50 Gbps router[3]) of the internal bandwidth of the switching fabric for forwarding lookup requests and responses. A design alternative that eliminates this bandwidth tax is to place

one route processor on each line card, known as the distributed route processing. However, at least in the near future, the centralized approach will be preferred over distributed approach in layer-4 switching for three reasons. First, at the time of writing, the typical per-link bandwidth that is commercially available is 2.4 Gbps (OC-48), which is far smaller than the layer-4 lookup bandwidth of 50 to 150 Gbps. Distributed approach will inevitably lead to the underutilization of the route processor. Second, the switching bandwidth of 0.5 Tbps [17] can be achieved, but at present layer-4 lookup bandwidth is only 50 to 150 Gbps (thanks to this paper). So the lookup speed will still be the bottleneck even if 25% of the switching bandwidth is used for internal communication. Third, when the amount of layer-4 switched traffic at each port is not evenly distributed, centralized approach achieves the best utilization of the lookup throughput by aggregating lookup requests into a single unit.

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 [1], [2] 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[14]. 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 over the time. Second, it should withstand denial-of-service attack. Lakshman et al. [1] 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[1]." We will address this issue in Section VI. 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[18]. 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 $D_{lifespan}$ (set to 120 in our cache) seconds old without affecting its miss ratio and throughput.

IV. The Notion of Near-LRU Cache

A. Flow Theory

Over the past thirty years, 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 virtual memory and CPU cache should be structured and managed to best exploit the *locality behavior* under many technological and economical constraints[19], [20], [21], [22], [23], [24], [25], [26], [27]. Likewise, locality behavior of the network traffic at layer-4 has been studied in Claffy's Ph.D thesis on Internet traffic characterization [4]. 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[4]. 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* [28] 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. These flow will be used throughout this paper.

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* $F(D_{expire}, t)$. Both are functions of both D_{expire} and 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

TABLE I
THE STATISTICS OF FIXWEST TRACES

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

[5] and [6], we found 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 stationarity of these two metrics is further corroborated by our trace-driven simulation shown next.

B. Traces used in the simulation

Table 1 contains seven FIXWEST Internet backbone 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 (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* (higher multiplexing level) than traffic in campus or corporate gateway, a fact well corroborated in [4], [29], [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. The other set of backbone traces from NLANR are AIX traces. However, they are not useful for our simulation study because they only sample 25% of the back-to-back packets in the traffic. FIXWEST traces, on the other hand, records 100% of the traffic. Among the seven traces we list in Table I, trace 1 contains the heaviest packet/traffic volume and has the longest duration. The simulation and measurement results from this trace will be presented throughout this paper. Results from other traces produced similar conclusions.

C. The Stationarity of ρ and F

Fig. 2. Packet Volume and Traffic Volume

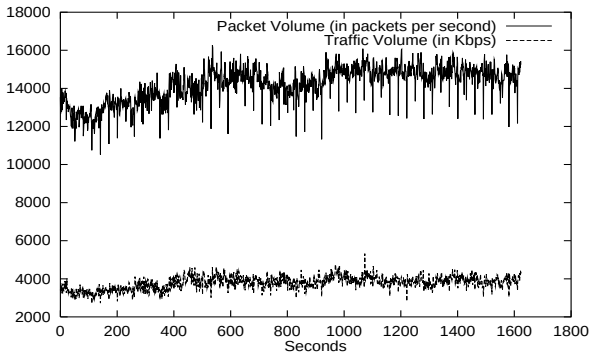


Fig. 3. Active Flows and New Flows Per Second

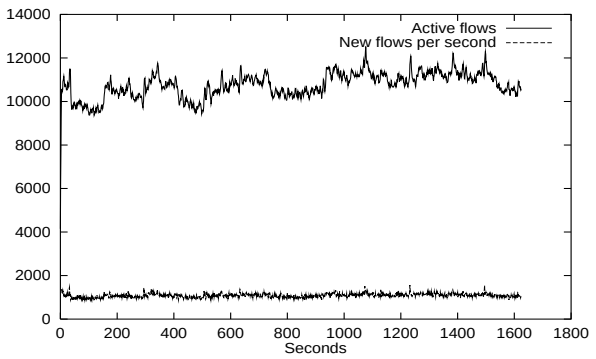


Fig. 2 shows the traffic volume U and packet volume V over the 1625-second time period. Fig. 3 shows the number of active flows F ($D_{expire} = 5s$) and the arrival rate of new flows ρ over this period. Visually, it can be seen that F is no more bursty than V, and ρ is much less bursty than U. Statistically, $C_X^2 = \frac{Var(X)}{(E[X])^2}$, *coefficient of variation* of a random variable X, measures the *variability* of X[30]. 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.

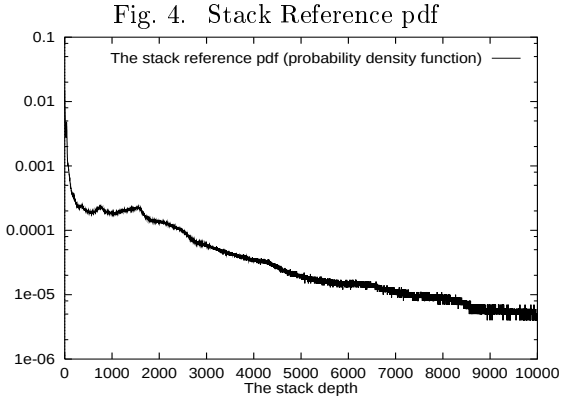
The stationarity of F and ρ in the flow theory framework motivates us to propose a cache management algorithm called *near-LRU*, which aims to exactly harness this locality behavior. In near-LRU, each cache entry is associated with a timestamp that is updated whenever this cache entry is accessed. When 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 92.6% 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 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 [27]. 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[27].

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 after-

Fig. 4. Stack Reference pdf



wards. The *weak locality condition* is satisfied for any $S > 2000$. Measurements of other traces produced the similar results. We conclude that when the cache size is fairly large (> 2000), LRU should be the optimal cache management algorithm.

D. 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 50 to 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 for us because FIXWEST traces are the only 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 [28], 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. But 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[2] are needed for FIXWEST trace 1 (30.8 Mbps) traffic to achieve a hit ratio over 90%, 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

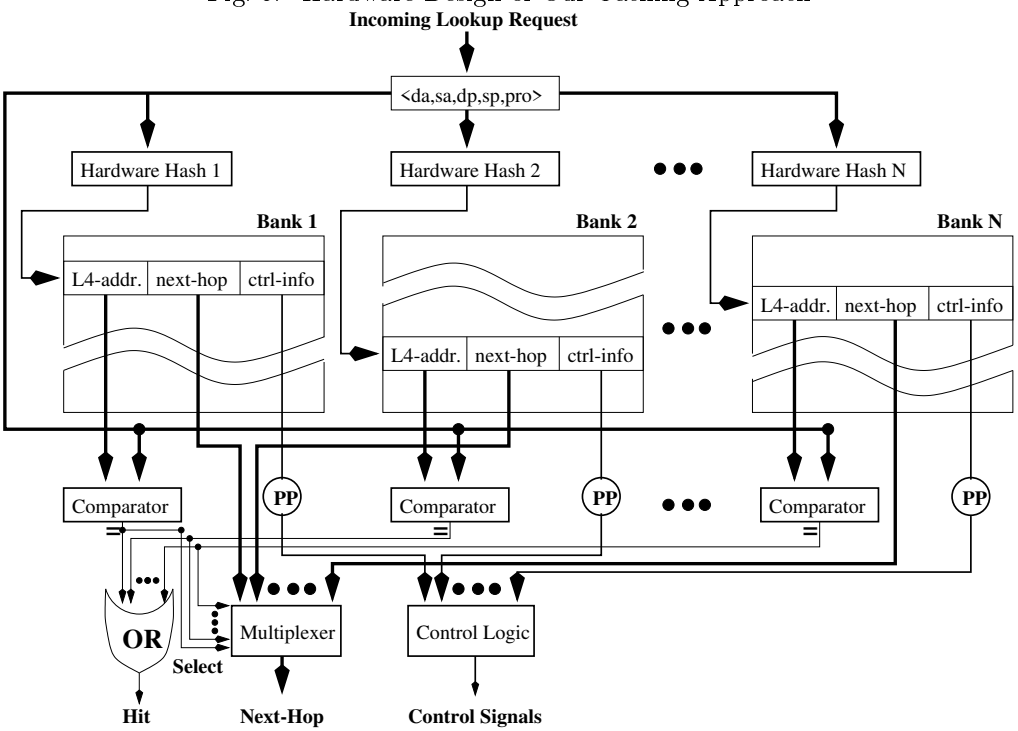
A cache is nothing more than chunks of RAM augmented by some auxiliary logic for search and replacement. The RAM we will use in our route cache is an emerging DRAM technology called Synchronous Link DRAM (SLDRAM). SLDRAM can deliver a very high throughput for read/write in *burst mode* (a series of accesses 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. We prefer SLDRAM over its competitor RDRAM/RAMBUS because the latter operates at a lower frequency (200 MHz). 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 (DRAM is about 1.5 dollars per Mbyte) in the near future. As in the worst case 256/768 MB route cache are 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 layer-4 address in the request will be processed by N different hardware hash functions in parallel to 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 LIFESPAN, and a 7-bit EXPIRE. The VALID bit indicates whether the entry contains a valid route cache. The LIFESPAN 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 and T being the current time. If this is true, as we have explained, the cache entry should no longer be trusted. The EXPIRE 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.

In hardware implementation, the value of the 7-bit EXPIRE field is incremented by 1 every $\frac{1}{8}$ second in $0 \rightarrow 1 \rightarrow 2 \rightarrow \cdots \rightarrow 127 \rightarrow 0 \rightarrow \cdots$ rounds. Two 7-bit 3-state (¡¿=) comparators are needed to check if T $\in$ [EXPIRE, EXPIRE+D_{expire}] (T is the current time) in parallel, when the threshold 127 is not crossed. Otherwise these two comparators will check if T$\in$[EXPIRE, 127] or T$\in$[0,EXPIRE+D_{expire}-128].

[2]This number is much smaller than the number of active flows (per Mbps) reported in [6] and [29] in traffic stream of similar bandwidth because they assume a timeout value of 64 seconds while the number in our analysis reflects a timeout of approximately 5.3 seconds, which is good enough for achieving a decent hit ratio.

Fig. 5. Hardware Design of Our Caching Approach



Since D_{expire} is usually between 4 to 6 seconds in our architecture and each "round" is 16 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. As to the coding of EXPIRE, the increment will be 1 every 2 seconds and $D_{lifespan}$ is set to 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" (more than 256 seconds old) flow to be identified as a living one is infinitesimal. The "L4-addr." fields in these N entries will be compared with the layer-4 address in the incoming lookup request using N 97-bit equal-only comparators 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. Also, the EXPIRE field will be updated to record the time of this recent access. If a miss happens, the request will be sent to a backup 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. In our cache architecture, the number of independent memory banks N is 8. This control logic that implements this random replacement consists of a 2048 $\times$ 11 ROM and a 3-bit state vector that records the index (say I) of the bank that is most recently replaced. The input to the control logic is the 8 bits that indicate whether a particular bank is replaceable. These 8+3 bits are the input to the ROM. The algorithm is to search bank I+1, I+2, ..., and I+8 (modulo 8) in sequence, and the first one that expires will the replaced. If there is a "collision" then the item in bank I+1 will be replaced. The state vector will be updated with the index of the bank that gets replaced. In our simulation study, we found that this emulated randomness is indistinguishable (in terms of hit ratio) from the true random replacement.

The VLSI implementation of the cache architecture has been completely laid out using VHDL and is ready for fabrication. The cost (reflected by die area) and performance of each logic component (chip) is analyzed according to the prevalent 0.25 μm process. The cache architecture can be implemented as a printed circuitry board attached to a port of the switching fabric. There are 8 (N=8) SLDRAM banks on this board. There are two chips that are associated with each memory bank. One (called hash chip) implements a 97-bit input 22 to 24-bit output hash function. The other (called local logic chip) implements

the timestamping logic (PP module), the 97-bit equal-only comparator, and other control logic such as FIFO buffer and multiplexer. The N local logic chips are identical. The N hash chips are also identical as it will be clear next that different hash functions can be realized using the same chip by initializing state registers to different values. The number of transistors needed for each chip is smaller than 100,000, making them readily implementable in VLSI. To estimate their cost, die area is a better indicator because both chips are I/O-bound in the sense that the area needed for placing transistors is smaller than the area needed for accommodating the number of I/O pins (100 to 150). Each chip occupies an area of 1mm by 2mm using 0.25 μm process and its cost is about 5 dollars (including packaging) assuming reasonable level of mass production (e.g., 3000+). One last chip implements the random replacement logic (the 2048×11 ROM) and other global control signals, with an area much smaller than 1mm by 1mm (cost a couple of dollars). Therefore, we estimate 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 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. We synthesized the aforementioned chips under 0.25 μm process and found that each and every of them has a propagation delay well low 10ns, the time needed for 800 MHz SLDRAM to access a 128-bit layer-4 route. Therefore, the lookup operation can perform as fast as the bandwidth of SLDRAM. The throughput of the cache can thus be estimated as follows. 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 (explained above) 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 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 later), 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 N hardware hash functions were the same. However, as we will show, static set-associative cache will bring about a high amount of *collision miss* when its load is high. We cut the collision miss down by 60% 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*[31]. It is well established that N hash functions randomly chosen from a function class called H_3[32] are N-universal hash functions. A significant amount of measurement is done in the literature [33], [34], [31] to demonstrate that the actual performance of such hash functions on real life data as well as random data satisfy the N-universal property.

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:

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

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

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[31], [35], [36], [37]. The scheme in [31], which is closest to ours, employs the same hash functions (H3) and is interested in the same metric (number of collisions). However, our scheme is fundamentally

different from [31] in two ways. First, the targeted application of [31] 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 [31] 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. Mckenny's, Goto's, and Hiraki's schemes 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.

A salient benefit of this scheme is that it can withstand 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 hashed into these sets will undergo severe thrashing. 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. 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 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 a rate higher than T3, it can be shown that overprovisioning the backup lookup power by 5% and the size of route cache by 2.5% will effectively guard against this type of attack.

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, the portion of the miss ratio 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.

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 (until the end of this paragraph) that in both dynamic and set-associative schemes, $R_{collision}$ is the same linear function (also same values in parameters) of a variable Γ. In this step, the discussion and calculation is unified for both static and dynamic cases schemes. 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 is 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[19] 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 num-

TABLE II
THE RELATIONSHIP BETWEEN E[O] AND C

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

ber of active flows in collision divided by the number of active flows F, that is, $\Gamma = \frac{O}{F}$. Then $R_{overflow}$ is expressed as $R_{overflow} = \frac{\lambda}{V} = \frac{cO}{V} = \frac{cF\Gamma}{V} = \left(\frac{cF}{V}\right)\Gamma$. In both dynamic and set-associative scheme, $\frac{cF}{V}$ is a constant when the traffic trace and system parameters like M, N, and D_{expire} are fixed. This constant c 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.

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 (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 relatively 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. We use the similar analysis method used in Mckenny's paper[38], [37]. Let us assume $p(x, s)$ denotes the probability that x data items are put into the system and s items settle. Then assume that all the items that settle is evenly distributed in the N banks. We have the following recurrence condition: $p(x, s) = \left(\frac{s}{NM}\right)^N p(x-1, s) + \left[1 - \left(\frac{s-1}{MN}\right)^N\right] p(x-1, s-1)$ 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. Then $\Gamma_{dynamic} = \frac{E[O]}{F} = \sum_{s=1}^{F} \frac{(F-s)p(F,s)}{F} = 1 - \sum_{s=1}^{F} \frac{sp(F,s)}{F}$ This recurrence condition is complex so that 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
$\Gamma_{dynamic}$ VS. $\Gamma_{dynamic}$

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 shows $\Gamma_{dynamic}$ and Γ_{static} under various load conditions (from 0.5 to 0.9). The theoretical results are calculated using our 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 are 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 deviations are very small. This independently corroborates the N-universal property of H3 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 alternative design and parameter choices had been attempted and their performance measured/simulated before we settle with this architecture and its system parameters. In this section, we present two of these alternatives and explain why they are not chosen finally.

(1) We explained that when there is a cache miss and there are more than one entries in the current cache set that expires, 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 the 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 replacement. The parameter tuning is to find the best D_{expire} in the middle. In our experiment, we found that D_{expire} is so chosen that F is about 60% of the cache size.

VII. Lazy-Writeback

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 EXPIRE bits. However, if a series of packets (say 10) 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 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 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, $M = 2048$) 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. The cost to implement this is small, as we only need to add two more comparators to each local logic chip.

VIII. Summary

We designed and implemented a novel layer-4 route cache architecture to support layer-4 lookup at memory access speeds. It implemented 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 actually is near-optimal. We showed that the same high and stable hit ratio can be achieved in the future when the Internet traffic volume is 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 the VLSI implementation of the cache architecture, and analyzed the func-

tion, 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 demonstrated 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. Srinivasna, 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] B. Kowalski, "Atlas policy cache architecture," Tech. Rep., StorageTek Corp., http://www.network.com/, 1996.

[12] MUSIC Semiconductors, "Cam tutorial," 1998.

[13] 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.

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

[15] P. Gupta, S. Lin, and N. Mckeown, "Routing lookups in hardware at memory access speeds," in *Proc. of IEEE INFOCOM'98*, Apr. 1998.

[16] N. Huang, S. Zhao, and J. Pan, "A fast ip routing lookup scheme for gigabit switching routers," in *Proc. of IEEE INFOCOM'99*, Mar. 1999, to appear.

[17] N. McKeown et al., "The tiny tera: A small high-bandwidth packet switch core," *IEEE Micro*, vol. 17, pp. 26–33, 1997.

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

[19] 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.

[20] P. Denning, "The working set model for program behavior," *Comm. of ACM*, vol. 11, no. 5, pp. 323–333, May 1968.

[21] M. Hill, "A case for direct-mapped caches," *IEEE Computer*, pp. 25–40, Dec. 1988.

[22] M. Hill and A. Smith, "Evaluating associativity in cpu caches," *IEEE Trans. on Computers*, vol. 38, no. 12, pp. 1612–1630, Dec. 1989.

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

[24] G. Rao, "Performance analysis of cache memories," *J. of ACM*, vol. 25, no. 3, pp. 378–395, July 1978.

[25] A. Smith, "A comparative study of set associative memory mapping algorithms and their use for cache and main memory," *IEEE Trans. on Software Engineering*, vol. 4, no. 2, pp. 121–130, Mar. 1978.

[26] A. Smith, "Cache memories," *ACM Computing Surveys*, vol. 14, no. 3, pp. 473–530, Sept. 1982.

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

[28] 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.

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

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

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

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

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

[34] 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.

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

[36] 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.

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

[38] P. Mckenny and J. Sarvela, "High-speed event counting and classification using a dictionary hash technique," Tech. Rep., SRI International, 1989.