Effects of a Hash-based Scheduler on Cache Performance in a Parallel Forwarding System

Weiguang Shi, Mike H. MacGregor and Pawel Gburzynski
Department of Computing Science,
University of Alberta,
Edmonton, AB, T6G 2E8, Canada
{wgshi, macg, pawel}@cs.ualberta.ca

Keywords Temporal Locality, Cache, Parallel Forwarding, Load Balancing

Abstract
We investigate the confrontation of load splitting and caching in high-performance parallel network forwarding systems. Our study demonstrates that hash-based load splitting schemes tend to significantly improve the temporal locality of the address stream submitted to a single routing engine (RE), which in turn greatly facilitates caching as a means of increasing system performance. We also show that the impact of locality on the efficiency of load balancing cannot be ignored: load balancing in a parallel forwarding system cannot be studied in isolation from the caching issues.

INTRODUCTION

The widening gap between the performance of processors and storage in many areas of computing stimulates caching as the standard approach to improving the performance of many systems. In consequence of this trend, many practical performance problems reduce to caching issues, e.g., the organization and size of cache storage and/or cache replacement policies. The underlying premise of caching is locality, which is present in many diverse workloads, including traces of machine instructions executed by a CPU, sequences of document requests in HTTP transactions, or streams of packets in a network. Significant temporal locality has been observed in network traffic at various levels [1–6]. In this work, we are concerned with the temporal locality in the stream of IP destination addresses extracted from packets arriving at a router and being input to a routing table lookup algorithm.

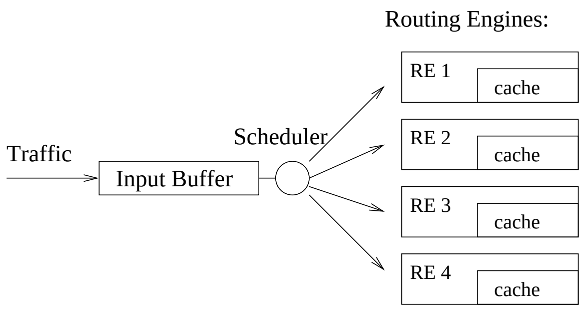

Figure 1: A Multi-processor Forwarding System

Routing table lookup is one of the bottlenecks to the performance of an Internet router, due to the "longest-prefix match" required by the IP addressing scheme and the growing size of routing tables. In response to the growing bandwidth of communication networks and the need for flexibility, network forwarding systems have evolved to employ multiple processors working in parallel to speed up the destination lookup and packet classification.

With a single processor, routing table lookup can be seen as a single server FCFS (First-Come-First-Served) queuing system, where packets enter the system, possibly spend some time waiting in the queue, and are processed sequentially in the order of their arrival.

A multiprocessor forwarding system can be viewed as an FCFS multi-server queue. Figure 1 shows an example with four routing engines (RE's). The scheduler is responsible for dispatching the next packet in the input queue to an RE, where it is processed, i.e., its output port is determined. Two important factors affecting the performance of the forwarding subsystem (encompassing a single RE) are the lookup speed of the RE, which in turn depends on the cache performance, and the efficiency of the RE scheduler. The scheduler's job is to find the next available routing engine and assign to it the next packet from the input queue. Under heavy load, a system with such a scheduler maintains a nearly work-conserving discipline and keeps all RE's busy as long as there is continuous supply of incoming packets.

Our goal is to compare the temporal locality in the workload of the forwarding sub-system under different scheduling schemes and evaluate the performance benefits of caching. As it turns out, by ignoring temporal locality in incoming packet traces, the simple scheduler, though "efficient in itself," may not yield the maximum performance of the forwarding system shown in Figure 1.

To simplify the discussion, we assume that the caches in the RE's are simple "*route caches*", with each entry consisting of a destination address paired with the output port over which all packets addressed to that destination should be forwarded. The cache replacement policy is LRU, and all RE's are equipped with caches of the same size.

Notably, an alternative approach would be to store address prefixes, rather than full IP addresses, in cache entries. This could result in some decrease of the cache size required to maintain the same abstract level of performance viewed as the cache hit rate. However, a realistic implementation of this idea would incur a tradeoff resulting from the fact that the longest prefix match (needed to locate entries in the cache) is a more complex operation than a direct lookup of a specific value. Moreover, a performance study of the prefix-based caching would have to consider the actual configuration of the routing tables at the router. This would complicate the experiments (by introducing additional parameters) and, more importantly, greatly reduce our experimental base, i.e., the collection of available IP traces. This is because published IP traces are routinely "sanitized" for privacy reasons, which operation conceals the identity of the specific IP addresses (thus removing any relationship between them, including common prefixes) while preserving their uniqueness. On the other hand, one can reasonably expect that the results obtained for address caching (as opposed to prefix caching) will qualitatively hold for prefix caching as well, because the issues of temporal locality in both approaches are essentially the same. ¿From some point of view, address based caching can be viewed as the worst case of prefix based caching. Furthermore, although our studies have been carried out within the particular context of routing table lookup, their results are applicable to other problems involving packet classification using multiple-NPU's (Network Processing Units) operating in parallel.

The rest of our paper is organized as follows. Section discusses previous related work. In section , we explain our locality measure and compare locality in scheduled traffic under Round-Robin and hash-based scheduling policies. We find that hash-based scheduling improves the locality in the stream of addresses submitted to a single RE, which in turn improves the performance of the entire forwarding system, even though Round-Robin distributes the load more "evenly" among the multiple RE's (in terms of the number of packets to process). This demonstrates that any discussion of load balancing in a parallel forwarding system must take into account the caching component, and include the impact of locality in the workloads leaving the scheduler. In section , we comment upon the role of caching in smoothing out the utilization levels of different RE's, whose loads, in terms of packet numbers, appear to be biased. In section , we discuss the performance metrics that account for the impact of caching, and then, in section , present the the simulation results illustrating the throughput and load distribution in hash-based parallel forwarding systems. Finally, section concludes this work.

RELATED WORK

As a case of temporal locality in network traffic, the packet "train" phenomenon was first discussed in [1] with reference to a token ring LAN. In [3], it was shown that the performance of an Internet router can be significantly improved by caching. Ref. [2] examines temporal locality of traffic in a corporate network environment and compares different cache replacement policies. Ref. [7] characterizes temporal locality in backbone network traffic and focuses on synthetic trace generation.

The effectiveness of different hashing schemes for network address lookup is compared in [8], where it is suggested to use high-order information bits of the address as a hash function for load distribution. When a packet arrives, its destination address is hashed to yield the forwarding engine to which the packet should be dispatched. Here the destination IP address is the key, and the number of processors determines the size of the hash table. Each entry of the hash table contains a sub-table, which yields the memory hierarchy containing the routing tables.

Along similar lines, [9] investigates load-balancing for hash-based traffic-splitting schemes. A table-based hash scheme is shown to perform as well as a CRC-based one, and also turns out to be an effective approach to load-balancing.

The problem of load distribution within a cluster of WWW proxy servers is considered in [10], and a a distribution scheme, dubbed HRW (Highest Random Weight), is proposed. It is shown that name-based mapping is an effective way of achieving good hit rate in proxy server caches. Given the huge difference in response time in the cases of a cache miss and hit, it is justified that the scheme balances load on the proxy servers in terms of objects rather than requests. Despite the drastic difference in the environment, our work shows that similar situations exist in routing.

Among the commercial network processor products implementing parallel routing, the IBM PowerNP [11] can be used with a "load balancer." The load balancing algorithm discussed in [12] is based on table-based hashing. To distribute the workload onto parallel NP's, the load balancer groups traffic flows into fine-grained bundles using information on the input queue length of the NP's. It also needs per packet timing information and per-bundle statistics to do the scheduling. The load balancer guarantees that the packets of a particular IP flow are passed to the same network processor.

Ref. [13] proposes an adaptive load-balancing mechanism based on the processor mapping described in [10]. The adaptive loop is necessary because even if the flows are distributed onto the processors evenly, imbalance can still occur due to different packet distributions within the flows.

In [14], a performance study is carried out for a forwarding system based on the Intel IXP1200 network processor [15]. Route caching is not explicitly exploited in the system, although the last referenced route does exist in each thread's register set.

Some routing table lookup algorithms have been designed with caching in mind. The solution proposed in [16] employs complex data structures to compress large routing tables to fit in the data cache. In [17], three routing table lookup algorithms are compared with respect to their cache performance. An interesting way to harness the standard virtual memory translation mechanism for IP route caching is discussed in [18].

Among the generic models and measures of temporal locality, the SLRUM (Simple LRU Stack Model), studied in [19] in the context of program page reference behavior, is simple, straightforward, and directly related to the natural LRU replacement policy implemented in many caching systems. As this model is of relevance to our study, we shall explain it briefly.

Imagine a stack as a one-dimensional array storing all different addresses that have been referenced so far. When a specific address is referenced, and it is present in the stack, its current location in the stack (assuming the top is at location zero), is output as the stack distance, and the LRU algorithm is used to update the stack, hence the name "LRU stack." The update consists in moving the referenced address to the top while shifting all the addresses preceding it one position down. When an address is referenced for the first time, it is assumed to be located below the last used entry of the stack. With this model, each entry in the address trace is directly transformed into a stack distance, and the entire trace is thus transformed into the corresponding "distance string." For example, the following trace: "$10, 1, 3, 1, 10, 2, 8, 3, 2, 3, 12, 1$" results in the following distance string: "$0, 1, 2, 1, 2, 3, 4, 4, 2, 1, 5, 4$".

TEMPORAL LOCALITY IN SCHEDULED TRAFFIC

From the viewpoint of IP address lookup, we prefer to view the stack distance in the SLRUM model, as an equivalent "reuse distance," defined as the number of distinct addresses preceding a given address in the trace string. To state this formally, consider an address trace "$a_1, a_2, \ldots, a_n, \ldots$". Let A_i be defined as the set containing all the unique addresses in the sequence "$a_1, a_2, \ldots, a_i$," with $A_0 = \emptyset$. The corresponding reuse distance sequence "$r_1, r_2, \ldots, r_n, \ldots$" is defined as follows:

$$r_i = \begin{cases} idx(a_i) & \text{if} \quad a_i \in A_{i-1} \\ |A_{i-1}| + 1 & \text{if} \quad a_i \notin A_{i-1}. \end{cases} \tag{1}$$

where $idx(a_i)$ yields the index of a_i in the LRU stack. Note that the first appearance of an address has the largest reuse distance so far plus one.

For a sufficiently long trace analyzed from the viewpoint of its statistically relevant properties, it makes sense to consider the frequency (or probability) of a given reuse distance n, interpreted as the the likelihood that an address that just appeared will be referenced again in n steps. Thus the frequency distribution of reuse distance describes the temporal locality of the address sequence. In [7], we show that reuse distance distributions of Internet traces tend to be

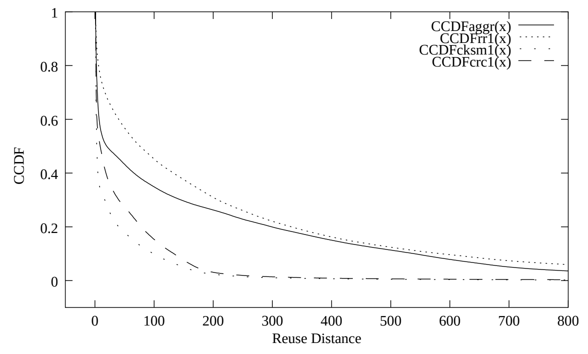

Figure 2: CCDF's in a 4-RE System with the UofA Trace

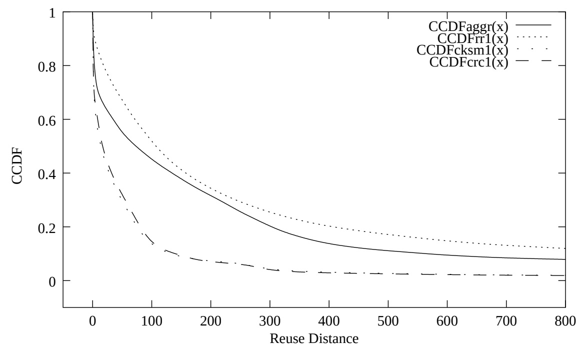

Figure 3: CCDF's in a 4-RE System with the LDestIP Trace

stable, and they can be modeled with a bi-modal distribution function.

A convenient way to characterize the reuse distance as a random variable is the Complementary Cumulative Distribution Function (CCDF):

$$CCDF(x) = 1 - CDF(x) \quad ,$$

where $CDF(x)$ is the (straightforward) Cumulative Distribution Function. This characterization is particularly convenient from the point of view of caching, since $CCDF(x)$ directly gives the miss ratio of an x-entry LRU cache operating on the source trace. Thus calculating CCDF's has the advantage over evaluating miss ratios for fixed cache sizes, because the CCDF's only need to be calculated once to obtain miss ratios for all cache sizes. Given two traces t_1 and t_2, one can say that trace t_1 has better temporal locality than t_2, if $CCDF_{t_1}(x) < CCDF_{t_2}(x)$ for all $x > 0$.

We investigate a forwarding system consisting of N RE's, with each RE using a route cache of M entries. Consequently, the cache miss rate observed by the i-th RE is $CCDF_i(M)$, where $CCDF_i(x)$ $(0 \le i < N)$ is the CCDF of the reuse distance in the address trace submitted to the RE. By $CCDF(x)$, we shall denote the CCDF of the full trace at the entry to the packet scheduler, before the traffic is split into the N streams.

The following two scheduling schemes are considered:

Round-Robin (RR) Packets are scheduled to RE's in a Round-Robin fashion. Note that in real life, strict Round-Robin scheduling is only possible when the per-packet processing time is fixed and the same for all the RE's. With caching, a realistic solution will

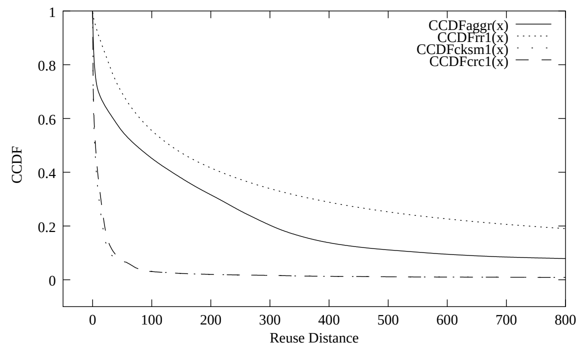

Figure 4: CCDF's in a 16-RE System with the LDestIP Trace

not be strictly Round-Robin since per-packet processing time for a cache hit is different from that for a miss. However, the strict Round-Robin scheme can still be used as a simple (unparameterized) representative of all those scheduling policies that are oblivious to caching.

Hashing A hash function is applied to the destination address to yield the index of the RE to deliver the packet. For the particular hash function, we examined the simple Fletcher checksum bits [20] used in the Internet protocols and the 16-bit CRC (Cyclic Redundant Check), as they both have been shown to be good hash functions for address lookup as well as load balancing.

Figures 2 and 3 show the CCDF's for a 4-RE system driven by a 1-million-entry trace gathered at University of Alberta and a 31-million-entry LDestIP trace from NLANR (National Laboratory of Applied Network Research) [21]. Figure 4 shows the results for the LDestIP trace in a 16-RE system. The following notations are used in these figures:

$CCDF_{aggr}(x)$ is the CCDF of the unscheduled, aggregate traffic.

$CCDF_{rr1}(x)$ is the CCDF of the traffic processed at the first RE, under Round-Robin.

$CCDF_{cksm1}(x)$ is the CCDF of the traffic processed at the first RE, under checksum hashing.

$CCDF_{crc1}(x)$ is the CCDF of the traffic processed at the first RE, under 16-bit CRC hashing.

The overall patterns of the CCDF's at other RE's under a given scheduling scheme are very similar to that observed at the first RE. There are differences, though, that will be discussed later in Section . Since the results are similar for the checksum and CRC hashing functions we will only discuss the results for the checksum function.

The three figures show the impact of the two scheduling schemes on the locality of scheduled traffic. Recall that the CCDF's can also be seen as the curves of miss ratio versus cache size [7] with simple destination route caches [3]. The workload at each RE under hashing has a much better temporal locality, and thus better cache performance, than for both RR and the original aggregate traffic. For example, for the UofA trace, with 50 cache lines, the miss ratio for *cksm1* is less than one third of that for *rr1*. With larger caches or more processors, the difference between the two disciplines is even larger.

It is evident that the RR-scheduled traffic has less temporal locality and hashing-scheduled traffic has more temporal locality than the original unscheduled traffic. Intuitively, RR disperses the original traffic over RE's but hashing groups packets that belong to the same flow and sends them to the same RE, thus improving temporal locality. These observations are abstracted and proved in [10] as the "Partitioning Non-Harmful" theorem which says that the expected hit rate in a partitioned mapping (e.g., hashing) is greater than or equal to that in a non-partitioned mapping (e.g., Round-Robin).

LOAD BALANCING

Another interesting performance aspect of our parallel forwarding system is the degree of load balancing achieved by a given scheduling policy. As shown in Table 1, under either hashing or Round-Robin, each RE sees roughly the same number of flows, although the number of flows seen by an RE under hashing is significantly smaller than that under RR. The total number of flows in the UofA trace is 5861. Under RR, the number of packets seen on each RE is trivially the same. However, under hashing, the number of packets seen at RE1 is more than twice that seen by RE2. In other words, the load under RR is perfectly balanced but skewed under hashing.

Hashing

RE	No. of Flows	No. of Packets
1	1473	385801
2	1436	174544
3	1525	213375
4	1427	226273

Round-Robin

RE	No. of Flows	No. of Packets
1	4282	249998
2	4268	249999
3	4303	249998
4	4258	249998

Table 1: No. of Flows vs No. of Packets Seen at Each RE (UofA Trace, 4 RE's)

The problem is that although hashing divides the number of flows almost evenly among RE's, due to the difference in flow volumes, the numbers of packets processed by different RE's can differ from each other. That is, hashing balances the number of flows while RR balances the number of packets.

In [22], Internet traffic flows are classified into *alpha* and *beta* traffic, where alpha traffic "is caused by large file transmissions over high-bandwidth links and is extremely bursty" and beta traffic is caused by file transmission over low-bandwidth links. Under hashing, when an alpha flow shows up in the workload, all packets of that flow will be dispatched to the same RE. There are relatively few alpha flows compared to the number of beta flows, but when one alpha flow is scheduled to an RE, this RE receives more packets to process than another RE handling only short-lived and low-volume beta flows.

The implication seems to be that the biased RE's processing alpha flows need more buffer space because they have more packets to process during bursts. Otherwise there has to be some flow-reassignment mechanism [12, 13] to keep the load balanced. However, this may be mitigated, or even completely counter-balanced by the fact that during a burst, most packets are from the same flow, so that locality and thus performance at the particular RE is improved. The most heavily loaded processor is often operating in a very efficient manner because it has a much lower cache miss ratio than the lightly loaded ones.

UofA with 4 RE's

Cache Size (entries)	Most Loaded RE (385801 packets)	Least Loaded RE (174544 packets)
1	0.655392	0.781786
10	0.330909	0.566196
100	0.0972235	0.187798
1000	0.00145907	0.00292227

LDestIP with 16 RE's

Cache Size (entries)	Most Loaded RE (3180088 packets)	Least Loaded RE (1586432 packets)
1	0.599992	0.739821
10	0.259524	0.415481
100	0.0214719	0.0392854
1000	0.00535446	0.0101868

Table 2: Route Cache Miss Rate vs No. of Packets under Hashing

Table 2 compares the cache performance of the most loaded and least loaded RE's in two cases. All RE's were configured identically and had the same amount of buffer space. Generally, the cache miss ratio is much lower for the heavily loaded RE's. With around 100 cache entries, the miss ratios for the heavily loaded RE's are approximately half those of the lightly loaded RE's. Note that Table 2 shows a relatively long term average result; in a traffic burst, the differences in performance can be more dramatic.

One can thus conclude that, depending on the speed gap between cache and memory, caching is helpful in reducing the input queue length of more heavily loaded processors. For example, if we use the following notation:

- t_m = the lookup time when there is a cache miss,
- t_h = the lookup time when there is a cache hit,
- R_1 = the miss ratio for the most loaded RE,
- R_2 = the miss ratio for the least loaded RE,
- $n = R_2/R_1$,
- t_i = the average address lookup time for RE_i,

then:

$$\begin{aligned}\frac{t_2}{t_1} &= \frac{t_m R_2 + t_h(1-R_2)}{t_m R_1 + t_h(1-R_1)} \\ &= \frac{R_2(\frac{t_m}{t_h}-1)+1}{\frac{R_2}{n}(\frac{t_m}{t_h}-1)+1} \qquad (2)\end{aligned}$$

Assuming $\frac{t_m}{t_h} = 11$ (a reasonable value for current memories and lookup algorithms), using the values in Table 2, for the UofA trace, $\frac{t_2}{t_1}$ is about 1.5 when each RE has a 10-entry route cache. That is, the least loaded RE is actually 50% slower than the most heavily loaded one.

This effect of caching is useful because it improves the performance of the processor that is processing the heaviest load. This, in turn, is beneficial because it reduces the chance that packets in a flow have to be assigned to different processors under load balancing mechanisms, such as the one described in [12].

PERFORMANCE METRICS

The throughput T of our parallel forwarding system is measured in terms of the number of packets forwarded during some unit period of time:

$$T = \frac{n}{t_d(p_n) - t_e(p_1)}$$

where

- n is the number of packets,
- $t_d(p)$ is the time when packet p is dequeued,
- $t_e(p)$ is the time when packet p is scheduled for lookup,
- $p_1, p_2, \ldots, p_n$ are the packets in their arrival order.

The cost of a route lookup depends on the cache state. When the route is in the cache, it takes t_h to finish the lookup. When the route is not in the cache, the time is t_m which includes t_h plus the cache miss penalty. All the variables measuring time can be expressed in terms of t_m and t_h, with t_m/t_h being usually much larger than 1. For example, for the BBN MGR ([23]), it is at least 5. As the speed gap

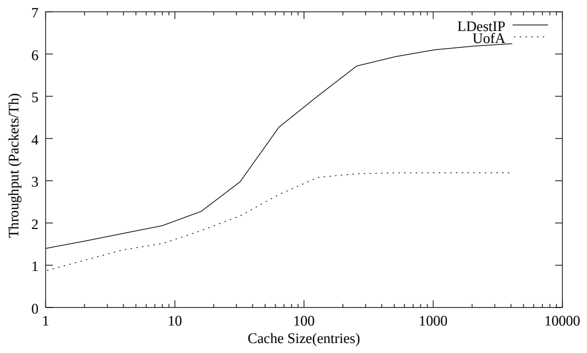

Figure 5: Effect of Cache on Throughput

between off-chip memory and CPU widens, this ratio can be much larger. For example, in [14], it takes the μEngine 30 cycles to transfer a word both to and from memory. Even with hardware assistance, it takes 30 cycles to complete an IP lookup.

Note that in a forwarding system with N RE's, the throughput T is bounded by $T_{max} = \frac{N}{t_h}$. This maximum is achieved if all N RE's are constantly busy and they all operate without misses.

Based on the discussion in section , we would also like to see how caching in general and t_m/t_h in particular affect the ability of network processors to balance the load. Load balancing is intended to achieve full utilization of the system. In other words, the goal is to distribute the workload over the RE's to keep them all busy.

We use B_i to represent the total busy time of the ith RE. The Coefficent of Variation (CV) of busy time is a measure of the effectiveness of load balancing:

$$CV = \frac{\sigma_B}{\mu_B}$$

where σ_B is the standard deviation and μ_B is the mean.

CV is chosen for its independence from the units of data. It is a measure of the combined effect of the traffic being skewed toward a few flows and the system's ability to balance the load. With the simulation input fixed, CV measures the latter. In the ideal case where each RE has exactly the same load, σ_B and CV both become zero.

SIMULATION

The system is assumed to have an infinite buffer that stores the incoming packets. The hash scheduler dispatches the packets in the order of their arrival. The scheduling overhead is ignored. We consider LRU route caches in an 8-RE system.

Figure 5 shows the throughput gain that can be achieved by using a cache. The maximum system throughput, eight packets per t_h, can be achieved only when there are no cache misses and the load is perfectly balanced. Without explicit load balancing mechanisms, the throughput reaches a maximum of 6.25 packets per t_h for the LDestIP trace and 3.19 packets per t_h for the UofA trace. Approximately 100 lines of cache are sufficient to triple or quadruple the throughput, compared to no caching.

Without explicit load balancing, the time that an RE spends on forwarding depends on both the characteristics of flows in the traffic and the forwarding capability of the processor. We have noted that the flows differ in rate, and the presence of alpha flows is the major factor causing load imbalance. Each trace has its specific flow composition and the results for different traces are not meant to be comparable with each other. Rather, we are interested in common trends that show the effects of caching.

To give an appreciation for the dominance of alpha flows, Table 3 lists the percentages of the ten largest flows in each trace. One may argue that there should be no distinguished dominant flows during an extended period of time in Internet traffic. Indeed, in simulations with longer traces, the effect of alpha flows is more likely to be averaged out. However, during the lifetime of an alpha flow, or more importantly, during a burst, the single flow does dominate and cause load imbalance at its selected RE. Table 3 shows that this is true even for the relatively long LDestIP trace.

Trace	UofA	LDestIP
No. of Pkts	999,993	31,518,464
No. of Flows	5,861	130,163
No. of Pkts in 10 Largest Flows	158707 (15.9%)	1183834(3.7%)
	24,245 (2.4%)	581,495 (1.8%)
	20,769 (2.1%)	524,542 (1.7%)
	17,482 (1.7%)	235,363 (0.7%)
	15,146 (1.5%)	212,150 (0.7%)
	14,305 (1.4%)	168,384 (0.5%)
	13,308 (1.3%)	160,798 (0.5%)
	12,348 (1.2%)	138,657 (0.5%)
	12,028 (1.2%)	125,531 (0.5%)
	11,824 (1.1%)	125,389 (0.5%)

Table 3: Alpha Flows in the Traces

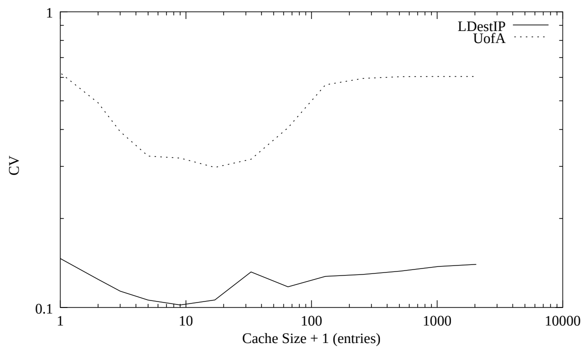

Figure 6: Effect of Cache Size on Load Balancing

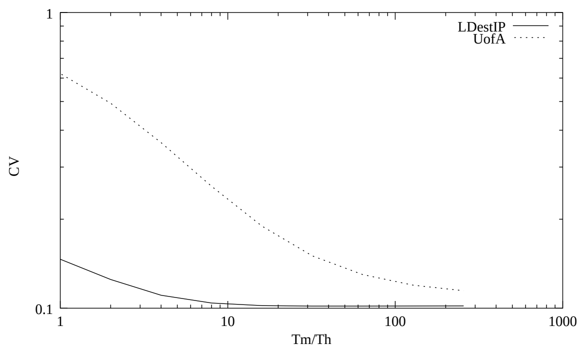

Figure 7: Effect of t_m/t_h on Load Balancing

Figure 6 illustrates the perceived load imbalance of different RE's under varying cache size, with $t_m/t_h = 6$. To show the value of CV when there is no cache (cache size equals to zero) in a log-scale plot, the x axis is set to represent "Cache Size + 1". The differences in position and scale of the curves for the two traces indicate that the more skewed the traffic is, i.e., the more dominant the few alpha flows are, the poorer the load balancing.

Initially, before the cache size reaches about 10, CV decreases with the increasing cache size. After that point, increasing cache size means higher CV. In the extreme, when the hit rate approaches 100 percent, the forwarding capabilities of all RE's equalize again—as if there were no caches. The shallow minimum in the CV for the UofA trace shows that it is not always beneficial (in terms of load balancing) to make caches as large as practically possible. This is even clearer for the LDestIP trace, where a lower percentage of packets in alpha flows leads to a sharper minimum in Figure 6.

Figure 7 shows the load imbalance as a function of t_m/t_h. Here, each RE has a cache of 16 entries. Again, due to different compositions of the two traces, the two curves differ. But generally, CV decreases as t_m/t_h increases, indicating that increasing miss time, or decreasing hit time, is beneficial to load balancing. Larger t_m/t_h ratios help to hide the fact that some RE's get more packets to forward than others because then there is less benefit from caching.

The simulation results in Figure 5 indicate that besides improving throughput, caching helps to balance RE utilization under certain circumstances, although caching alone is not a sufficient measure to balance the workload (as shown in Figures 6 and 7).

CONCLUSIONS

We have shown that significant temporal locality exists in hash-scheduled traffic in a parallel forwarding system. As a result, system performance can be improved by using caches. Packet dispatching using a hash function does not usually create a load-balanced system because of differences in flow volumes: the largest flows contribute to the imbalance of the system. However, the largest flows also improve temporal locality in the traffic that passes through an RE. Thus a side-effect of caching is to improve the load balance and thus system utilization, especially during bursts.

System utilization can be improved further by using a load-balancing scheme that re-assigns flows from busy RE's to idling ones. Under hash-based scheduling, alpha flows in Internet traffic are the main contributors to workload imbalance. Moving high-volume flows instead of re-assigning flows indiscriminately will result in less disturbance to the cache of the target RE, which is desirable. We are currently developing effective load-balancing mechanisms based on this observation.

References

[1] R. Jain, S. Routhier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal of Selected Areas in Communications*, SAC-4(6):986–995, September 1986.

[2] R. Jain. Characteristics of destination address locality in computer networks: a comparison of caching schemes. *Computer Networks and ISDN Systems*, 18(4):243–254, May 1990.

[3] D. Feldmeier. Improving gateway performance with a routing table cache. In *IEEE INFOCOM' 88*, pages 298–307, 1988.

[4] J. Mogul. Network locality at the scale of processes. In *Proc. ACM SIGCOMM '91*, pages 273–285, 1991.

[5] V. Almeida, A. Bestavros, M. Crovella, A. Oliveira. Characterizing reference locality in the WWW. In *The IEEE Conference on Parallel and Distributed Information Systems (PDIS'96)*, pages 92–107, Miami Beach, FL USA, 1996.

[6] S. Jin, A. Bestavros. Sources and characteristics of Web temporal locality. In *Proceedings of Mascots'2000: The IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 28–35, San Fransisco, CA, USA, August 2000.

[7] W. Shi, M. H. MacGregor, P. Gburzynski. Synthetic IP address trace generation, 2003. Submitted to ACM SigMetrics 2003.

[8] R. Jain. A comparison of hashing schemes for address lookup in computer networks. *IEEE Transactions on Communications*, 40(3):1570–1573, October 1992.

[9] Z. Cao, Z. Wang, E. Zegura. Performance of hashing-based schemes for Internet load balancing. In *IEEE Infocom 2000*, pages 332–341, March 2000.

[10] D. G. Thaler, C. V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ ACM Transactions on Networking*, 6(1):1–14, February 1998.

[11] IBM Corporation. IBM PowerNP network processor, 2002.

[12] G. Dittmann, A. Herkersdorf. Network processor load balancing for high-speed links. In *2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS2002)*, San Diego, USA, July 2002.

[13] L. Kencl, J. Le Boudec. Adaptive load sharing for network processors. In *IEEE Infocom 2002*, New York, USA, June 2002.

[14] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb. Building a robust software-based router using network processors. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 216–229, Banff, AB, Canada., 2001.

[15] Intel Corporation. Intel IXP1200 network processor, 2000.

[16] M. Degermark, A. Brodnik, S. Carlsson, S. Pink. Small forwarding tables for fast routing lookups. In *Proc. ACM SIGCOMM '97*, pages 3–14, Cannes, France., 1997.

[17] W. Shi, M. H. MacGregor. Cache reference behavior of three IP routing table lookup algorithms. In *The 5th Multi-Conference on Systemics, Cyberntics and Informatics (SCI-2001)*, Orlando, Florida USA., 2001.

[18] T. Chiueh, P. Pradhan. High performance IP routing table lookup using CPU caching. In *Proceedings of IEEE INFOCOM'99*, pages 1421–1428, 1999.

[19] J. Spirn. *Program Behavior: Models and Measurements*. Elsevier-North Holland, N.Y., 1977.

[20] J. G. Fletcher. An arithmetic checksum for serial transmissioins. *IEEE Transactions on Communications*, COM-30(1):247–252, January 1982.

[21] NLANR (National Laboratory for Applied Network Research) Measurement and Operations Analysis Team (MOAT). Packet header traces. http://moat.nlanr.net.

[22] S. Sarvotham, R. Riedi, R. Baraniuk. Connection-level analysis and modeling of network traffic. In *Proc. ACM SIGCOMM Internet Measurement Workshop IMW2001*, pages 99–103, San Francisco, California, 2001.

[23] C. Partridge, et al. A fifty gigabit per second IP router. *IEEE/ACM Trans. on Networking*, 6(3):237–248, 1998.