Efficient Packet Classification with Digest Caches

Francis Chang, Wu-chang Feng, Wu-chi Feng
Systems Software Laboratory
OGI School of Science and Engineering at OHSU
Beaverton, Oregon, USA
{francis, wuchang,wuchi}@cse.ogi.edu

Kang Li
Department of Computer Science
University of Georgia
Athens, Georgia, USA
kangli@acm.org

***Abstract*— In this paper, we propose a digest cache-based algorithm for efficient packet classification in network devices. The digest cache-based algorithm classifies packets by using a hash of the flow identifier allowing for smaller sized cache entries at the expense of a small amount of packet misclassification. Experimentation will show that our technique is superior to previous Bloom filter based designs in all areas, including extensibility, computational complexity, and memory efficiency. We also discuss how to extend this technique to improve performance for exact caches.**

Keywords—packet classification; caches; probabilistic algorithms

I. INTRODUCTION

As the number of hosts and network traffic continues to grow, the need to efficiently handle packets at line speed becomes increasingly important. Packet classification is one technique that allows in-network devices such as firewalls, network address translators, and firewalls to provide differentiated service and access to network and host resources by efficiently determining how the packet should be processed. These services require a packet to be classified so that a set of rules can be applied to such network header information as the destination address, flow identifier, port number, or layer-4 protocol type. The development of more efficient classification algorithms has been the focus of many research papers including: [2],[11],[15],[20],[29],[31]. However, the hardware requirements of performing a full classification on each packet at current line rates can be overwhelming [23]. Moreover, there does not appear to be a good algorithmic solution for multiple field classifiers containing more than two fields [3].

A classic approach to managing data streams that exhibit temporal locality is to employ a cache that stores recently referenced items. Packet classification is no different [8]. Such caches have been shown to increase the performance of route lookups significantly [18],[32]. How well a cache design performs is typically measured by its hit rate for a given cache size. Generally, as additional capacity is added to the cache, the hit rates and performance of the packet classification engine should increase. Unlike route caches that only need to store destination address information, packet classification caches require the storage of full packet headers. Unfortunately, due to the increasing size of packet headers (the eventual deployment of IPv6 [16]), storing full header information can be prohibitively expensive given the high-speed memory that would be required to implement such a cache.

Recently, we proposed a third axis for designing packet classification algorithms: accuracy [5]. That is, given a certain amount of error allowed in packet classification, can packet classification speeds be significantly increased? In a previous paper, we proposed the use of a modified Bloom filter [1] for packet classification. In that approach, classified packets satisfying a binary predicate are inserted into the filter that caches the decision. For instance, a network bridge would add flows that it has identified that it should forward to the Bloom filter. Subsequent packets then query the filter to quickly test membership

This work supported by the National Science Foundation under Grant EIA-0130344 and the generous donations of Intel Corporation. Any opinions, findings, or recommendations expressed are those of the author(s) and do not necessarily reflect the views of NSF or Intel.

before being processed further. Packets that hit in the filter are processed immediately, based on the predicate, while packets that miss go through the slower full packet classification lookup process.

There are three primary limitations of this Bloom filter cache design. First, each Bloom filter lookup requires N independent memory accesses, where N is the number of hash levels of the Bloom filter. For a Bloom filter optimized for a 1 in a billion packet misclassification probability $N = 30$. Second, no mechanism exists to recover the current elements in a Bloom filter, preventing it from using efficient cache replacement mechanisms such as LRU. Finally, a Bloom cache is only effective in storing less than 256 binary predicates. Thus, it is not an appropriate data structure to attach an arbitrary amount of data, due to the increasing number of Bloom filters required to support the data.

In this paper, we propose the notion of *digest caches* for efficient packet classification. The goal of digest caches is similar to Bloom-filter caches in that they trade some accuracy in packet classification in exchange for increased performance. Digest caches, however, allow traditional cache management policies such as LRU to be employed to better manage the cache over time. Instead of storing a Bloom filter signature of a flow identifier (source and destination IP addresses & ports and protocol type), it is necessary only to store a hash of the flow identifier, allowing for smaller sized cache entries. We will also discuss how to extend this idea to accelerate exact caching strategies by building multi-level caches with digest caches.

Section II covers related work while Section III outlines the design of our architecture. Section IV evaluates the performance of our design using sample network traces while Section V discusses the performance overhead incurred by our algorithm as measured on the IXP1200 network processor platform.

II. RELATED WORK

Due to the high processing costs of packet classification, network appliance designers have resorted to using caches to speed up packet processing time. Early work in network cache design borrowed concepts from computer architecture (LRU stacks, set-associative multi-level caches) [18]. Some caching strategies rely on CPU L1 and L2 caches [23] while others attempt to map the IP address space to memory address space in order to take advantage of the hardware TLB [6]. Another approach is to add an explicit timeout to an LRU set-associative cache to improve performance by reducing thrashing [32]. More recently, in addition to leveraging the temporal locality of packets observed on networks, approaches to improving cache performance have applied techniques to compress and cache IP ranges to take advantage of the spatial locality in the address space of flow identifiers as well [7],[14]. This effectively allows multiple flows to be cached in a single cache entry, so that the entire cache may be placed into small high-speed memory such as a processor's L1/L2 cache. There has been work using Bloom Filters to accelerate exact prefix-matching schemes [10].

Much of this work is not applicable to higher-level flow identification that is the motivation for our work. Additionally, all of these bodies of work are fundamentally different from the material presented in this paper, because they only consider exact caching strategies. Our approach attempts to maximize performance given constrained resources and an allowable error rate.

III. OUR APPROACH

Network cache designs typically employ simple set associate hash tables, ideas that are borrowed from their traditional memory management counterparts. The goal of the hash tables is to quickly determine the operation or forwarding interface that should be used, given the flow identifier. Hashing the flow identifier allows traditional network processors to determine what operation or forwarding interface should be used while examining only a couple of entries in the cache. We believe one limitation of exact matching caches for flow identifiers is the need to store quite large flow identifiers (e.g. 37 bytes for an IPv6 flow identifier) with each cache entry. This limits the amount of information one can cache or increases the time necessary to find information in the cache.

In this paper, we propose the notion of *digest caches*. The most important property of a digest cache is that it stores only a hash of the flow identifier instead of the entire flow identifier. The goal of the digest is to significantly reduce the amount of information stored in the cache, in exchange, for a small amount of error in cache lookups. As will be described later in this section, digest caches can be used in two ways. First, they can be used as the only cache for the packet classifier, allowing the packet classifier caches to be small. Second, they can be used as an initial lookup in an exact classification scenario. This allows a system to quickly partition the incoming packets into those that are in the exact cache and those that are not.

In the rest of this section, we will motivate approximate algorithms for packet classification caches. We will then focus on properties of the digest cache, comparing it to previously proposed Bloom-filter-based packet classifiers, and using them to speed up exact packet classifiers.

Digest caches are superior to Bloom caches in two ways. Cache lookups can be performed in a single memory access, and they allow direct addressing of elements, which can be used to implement efficient cache eviction algorithms, such as LRU.

A. The Case for an Approximate Algorithm

For the purposes of this study, we use a misclassification probability of one in a billion. Typically, TCP checksums will fail for approximately 1 in 1100 to 1 in 32000 packets, even when link-level CRCs should only admit error rates of 1 in 4 billion errors. On average, between 1 in 16 million to 1 in 10 billion TCP packets will contain an undetectable error [30]. We contend that a misclassification probability of this magnitude will not meaningfully degrade network reliability. It is the responsibility of the end system to detect and compensate for errors that may occur in the network [26].

Errors in the network are typically self-healing in the sense that misdirected flows will be evicted from the cache as they age. Moreover, the network already guards against mis-configurations and mistakes made by the hardware. For example, the IP TTL fields are used to protect against routing loops in the network.

Another argument underscoring the unreliability of the network is that TCP flows that are in retransmission timeout (RTO) mode are of no use. Consider a web browser. Flows that are stalled in RTO mode often result in the user re-establishing a web-connection. In the case that a reload is necessary, a new ephemeral port will be chosen by the client, and thus a new flow identifier is constructed. If an approximate cache has misclassified a previous flow, it will have no impact on the classification of the new flow.

In some cases, such as firewalls, it is undesirable for the cache systems to have errors. To "harden" approximate caching hardware against misclassifications, layer-4 hints, such as TCP SYN flags can be used to force a full packet classification pass to ensure that new flows are not misclassified.

B. Dimensioning a Digest Cache

The idea of our work is simply the direct comparison of hashed flow identifiers to match cached flows. In this sense, we will trade the accuracy of a cache for a reduced storage requirement. We will partition memory into a traditional, set-associative cache.

When constructing our cache, we need to decide how to structure our usage of memory. Previous work has demonstrated that higher cache associativity yields better cache hit-rates [18][21]. However, in the case of the digest cache, an increase in the degree of associativity must be accompanied by an increase in the size of the flow identifier's hash, to compensate for the additional probability of collision.

If the digest is a c-bit hash, and we have a d-way set associative cache, then the probability of cache misidentification is

$$p \approx \frac{d}{2^c} \tag{1}$$

The equation can be described as follows: Each cache line has d entries, each entry of which can take 2^c values. A misclassification occurs whenever a new entry has coincidentally the same hash value as any of the existing d entries. We must employ a stronger hash to compensate for increasing collision opportunities (associativity).

Figure 1 graphs the number of flows that a 4-way set associative can store, assuming different misclassification probability tolerances. The maximum number of addressable flows increases linearly with the amount of memory and decreases logarithmically with the packet misclassification rate.

C. Theoretical Comparison

To achieve a misclassification probability of one in a billion, a Bloom filter cache must use 30 independent hash functions to optimally use memory. This allows us to store a maximum of k flows in our cache [5],

$$k_{Bloomcache} = \frac{\ln(1 - p^{1/L})}{\ln(1 - L/M)} \tag{2}$$

where $L = 30$, the number of hash functions, M, the amount of memory, in bits, and p, the misidentification probability. To compare directly with a digest cache, the maximum number of flows that our scheme can store, independent of the associativity, is given by

$$k_{digest} = \frac{M}{c} \tag{3}$$

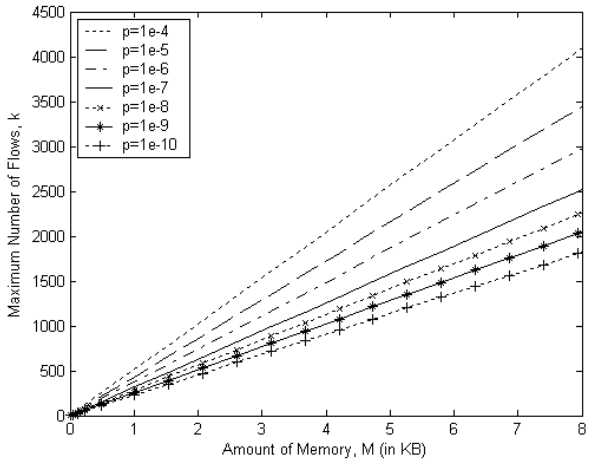

Figure 1: Maximum number of flows that can be addressed in a 4-way set associative digest cache, with different misclassification probabilities, p

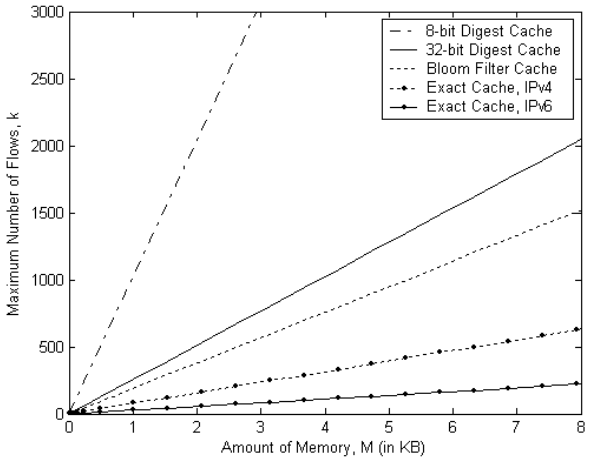

Figure 2: Comparison of storage capacity of various caching schemes. The Bloom filter cache assumes a misidentification probability of one in a billion, which under optimal conditions is modeled by a Bloom filter with 30 hash functions.

where the required number of bits in the digest function is given by

$$c = \lceil \log_2(d / p) \rceil \quad (4)$$

This relation is dependent on p, the misidentification probability and d, the desired level of cache set associativity. The derivation of this formula follows from Equation 1.

Figure 2 compares the storage capacity of both caching schemes. Both schemes linearly relate storage capacity to available memory, but it is interesting to note that simply storing a hash is more than 35% more efficient in terms of memory use than a Bloom filter, for this application. One property that makes a Bloom filter a useful algorithm is its ability to insert an unlimited number of signatures into the data structure, at the cost of increased misidentification. However, since we prefer a bounded misclassification rate, this property is of no use to the solution to our problem.

D. A Specific Example of a Digest Cache

To illustrate the operation of a digest cache, we will construct an example application of a digest cache. Suppose we have a router with 16 interfaces and a set of classification rules, R.

We begin by assuming that we have 64KB of memory to devote to the cache and wish to have a 4-way associative cache that has a misclassification probability of one in a billion.

These parameters can be fulfilled by a 32-bit digest function, with 4 bits used to store per-flow routing information. Each cache entry is then 36 bits, making each cache line 144 bits (18 bytes).

64KB of cache memory partitioned into 18-byte cache lines, gives a total of 3640 cache lines, which allows our cache to store 10920 distinct entries. A visual depiction of this cache is given in Figure 3.

Now, let us consider a sample trace of the cache, which is initially empty. Suppose 2 distinct flows, A and B.

1. Packet 1 arrives from flow A.
   a. The flow identifier of A is hashed to $H_1(A)$ to determine the cache line to look up. That is, H_1 is a map from flow identifier to cache line
   b. A is hashed again to $H_2(A)$, and compared to all 4 elements of the cache line. There is no match. The result $H_2(A)$ is the digest of the flow identifier that is stored.
   c. A is classified by a standard flow classifier, and is found to route to interface 3.
   d. The signature $H_2(A)$, is placed in cache line $H_1(A)$, along with its routing information. (Interface 3)
   e. The packet is forwarded through interface 3.
2. Packet 2 arrives from flow A.

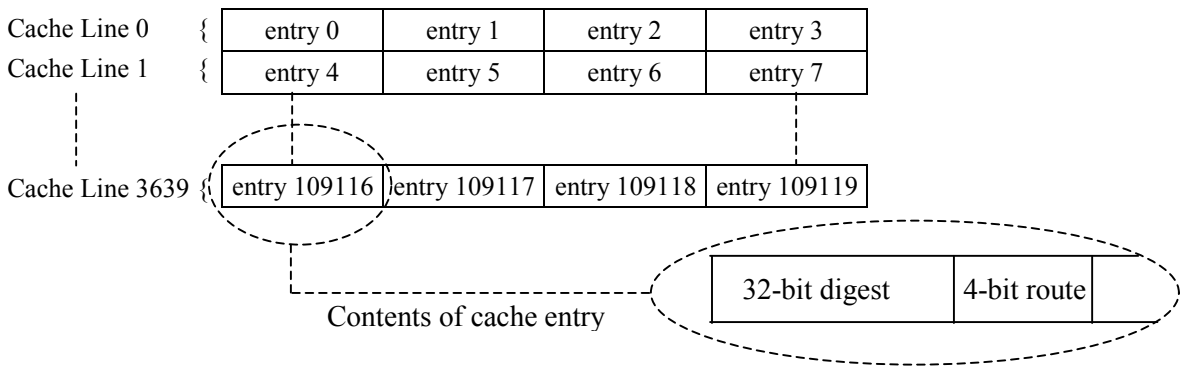

Figure 3: An overview of 64KB 4-way set associative digest cache, with a misclassification probability of 1 in a billion. This cache services a router with 16 interfaces.

a. The flow identifier of A is hashed to $H_1(A)$ to determine the cache line to look up.

b. A is hashed again to $H_2(A)$, and compared to all 4 elements of the cache line. There is a match, and the packet is forwarded to interface 3.

3. Packet 3 arrives from flow B.

a. The flow identifier of B is hashed to $H_1(B)$ to determine the cache line to look up. Coincidentally, $H_1(A)= H_1(B)$

b. B is hashed again to $H_2(B)$, and compared to all 4 elements of the cache line. Coincidentally, $H_2(A)= H_2(B)$. There is a match, and the packet is forwarded to interface 3. The probability that this sort of misclassification occurs has a probability of $4/2^{32} \approx 10^{-9}$.

In the absence of misclassifications, this scheme behaves exactly as a 4-way set associative cache with 10920 entries (3640 cache lines).

Using an equivalent amount of memory (64 KB) a cache storing IPv4 flow identifiers will be able to store 4852 entries, and a cache storing IPv6 flow identifiers will be able to store 1744 entries.

The benefit of using a digest cache is two-fold. First, it increases the effective storage capacity of cache memory, allowing the use of smaller, faster memory. Second, it reduces the memory bandwidth required to support a cache by reducing the amount of data required to match a single packet.

As intuition and previous studies would indicate, a larger cache will improve cache performance [18][21][24]. To that end, in this example, the deployment of a digest cache would have an effect of increasing the effective cache size by a factor of 2-6.

E. Exact Classification with Digest Caches

Digest caches can also be used to accelerate exact caching systems, by employing a multi-level cache (Figure 4). A digest cache is constructed, in conjunction with an exact cache that shares the same dimensions. While the digest cache only stores a hash of flow identifiers, the exact cache stores the full flow identifier. Thus, the two hierarchies can be thought of as "mirrors" of each other.

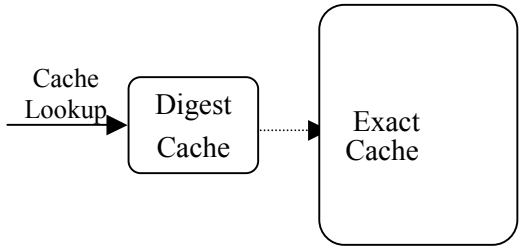

Figure 4: A multi-level digest-accelerated exact cache. The Digest cache allows you to filter potential hits quickly, using a small amount of faster memory.

A c-bit, d-way set associative digest cache implemented in a sequential memory access model will be able to reduce the amount of exact cache memory accessed (due to cache misses) by a factor of

$$p_{miss\_savings} = \frac{1}{2^c} \quad (5)$$

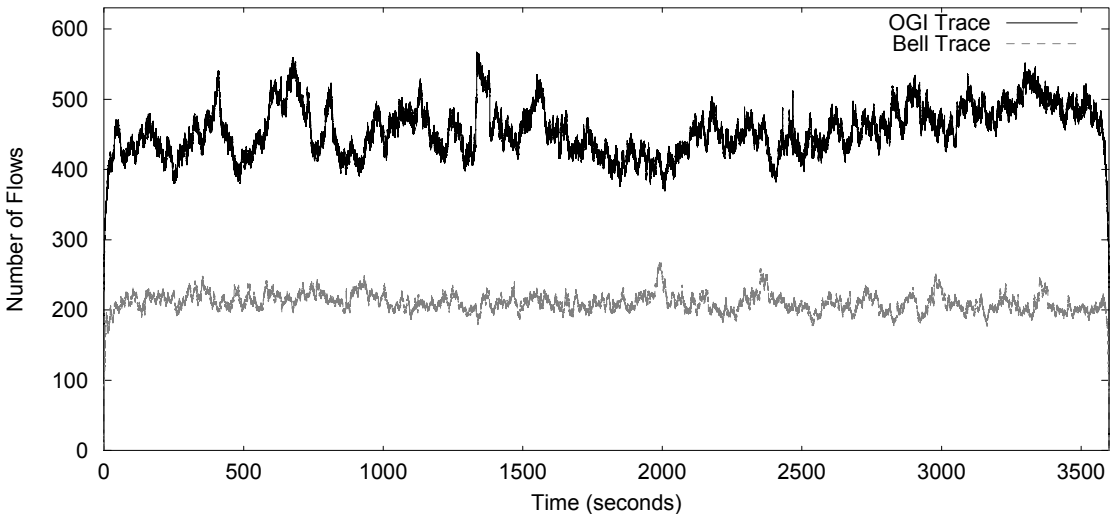

Figure 5: Number of concurrent flows in test data sets

while the amount of exact cache memory accessed by a cache hit is reduced by a factor of

$$p_{hit\_savings} = \frac{1}{d} + \frac{1}{2^c} \times \frac{d-1}{d} \quad (6)$$

The intuition behind Equation 6 is that each cache hit must access the exact flow identifier, while each associative cache entry has an access probability of 2^{-c}.

Note that the digest cache allows for multiple entries in a cache line to share the same value because the exact cache can resolve collisions of this type.

Since this application relies on hashing strength only for performance and not for correctness, it is not necessary to have as strong a misclassification rate.

A multi-level 8-bit 4-way set associative digest-accelerated cache will incur a 4-byte 1st level lookup overhead. However, it will reduce 2nd level memory access cost of an IPv6-bit cache miss lookup from 148 bytes to 37.4 bytes, and a cache miss lookup from 148 bytes to .6 bytes. Assuming a 95% hit rate, the average cost of cache lookups is reduced to 4 bytes of 1st level cache and 35.6 bytes of 2nd level cache.

IV. EVALUATION

For evaluation purposes, we used two datasets, each of one hour in length. The first of the datasets was collected by Bell Labs research, Murray Hill, NJ, at the end of May 2002. This dataset was made available through a joint project between NLANR PMA and Internet Traffic Research Group [25].

The trace was of a 9 Mb/s Internet link, serving a staff of 400 people.

The second trace was a non-anonymized trace collected at our university OC-3c link. Our link connects with Internet2 in partnership with the Portland Research and Education Network (PREN). This trace was collected on the afternoon of July 26th, 2002.

	Bell Trace	**OGI Trace**
Trace Length (seconds)	3600	3600
Number of Packets	974613	15607297
Avg. Packet Rate (Packets Per Second)	270.7	4335.4
TCP Packets	303142	5034332
UDP Packets	671471	10572965
Number of Flows	32507	160087
Number of TCP Flows	30337	82673
Number of UDP Flows	2170	77414
Avg. Flow Length (seconds)	3.27	10.21
Longest Flow (seconds)	3599.95	3600
Avg. Packets/Flow	29.98	97.49
Avg. Packets/TCP Flow	9.99	60.89
Avg. Packets/UDP Flow	309.43	136.58
Max # of Concurrent Flows	268	567

Table 1: Summary statistics for the sample traces

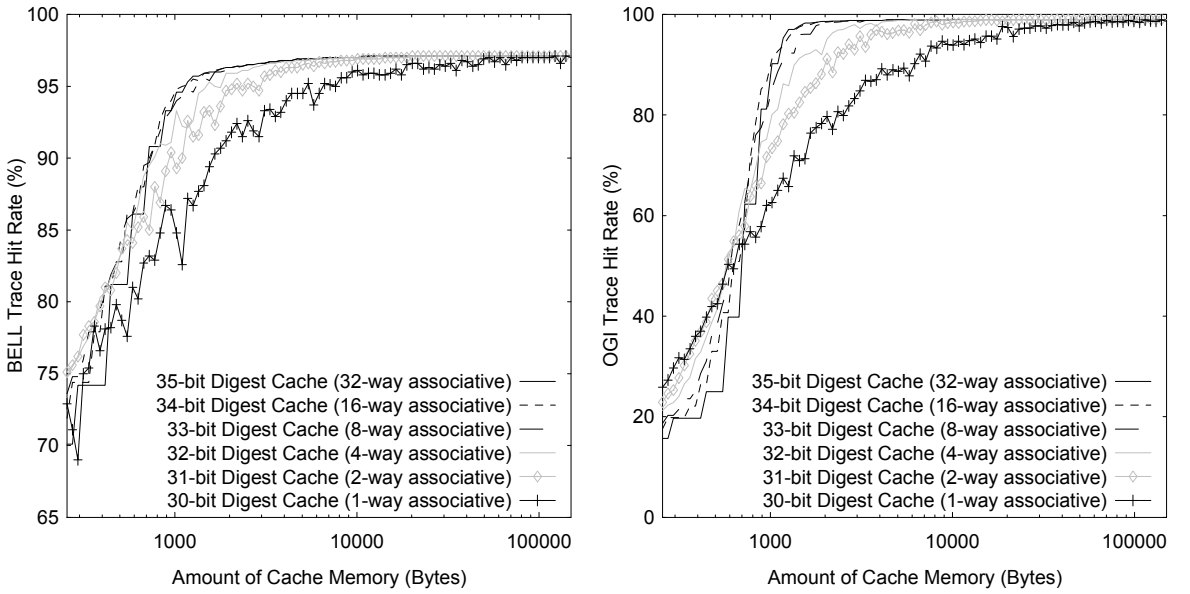

Figure 6: Hit Rates for digest caches, as a function of memory for various set associativity, assuming a misclassification rate of 1 in a billion

Table 1 presents a summary of the statistics of these two datasets. A graph of the number of concurrent flows is shown in Figure 5. For the purposes of our graph, a flow is defined to be active between the time of its first and last packet, with a 60 second maximum inter-packet spacing. This number is chosen in accordance with other measurement studies [13][22].

A reference "perfect cache" was simulated. We define a perfect cache to be a fully associative cache with an infinite amount of memory. Thus, a perfect cache only takes compulsory cache misses. The results are presented in Table 2. The OGI trace captured a portion of an active Half-life game server, whose activity is characterized by a moderate number (~20) of long-lived UDP flows.

	Bell Trace	OGI Trace
Hit Rate	0.971	0.988
Intrinsic Miss Rate	0.029	0.012
Maximum misses (over 100ms intervals)	6	189
Variance of misses (over 100 ms intervals)	1.3540	17.438
Average misses (over 100 ms intervals)	0.775	5.843

Table 2: The results of simulating a perfect cache

A. Reference Cache Implementations

A Bloom filter cache[5] was simulated, using optimal dimensioning. Both cold caching and double-buffered aging strategies were run on the benchmark datasets. Optimal dimensioning for a misclassification probability of one in a billion requires 30 independent hash functions, meaning that each cache look-up and insertion operation requires 30 independent 1-bit memory accesses.

The digest cache presented in this paper was chosen to be a four-way set associative hash table, using 32-bit flow identifier digests. Each lookup and insertion operation requires a single 16-byte memory request. An LRU cache replacement algorithm was chosen, due to its low cost complexity and near-optimal behaviour [18].

A 4-way set associative cache was chosen, because it performs almost as well as a fully associative cache [21]**.** Figure 6 graphs the behaviour of digest caches with different set associativities.

We also compare our cache against a traditional four-way set associative layer-4 IPv4 and IPv6 based hash tables. Each lookup and insertion operation requires a single 52-byte or 148-byte memory request, respectively.

Hashing for all results presented in this paper was accomplished with a SHA-1 [12] hash. It is important to note that the cryptographic strength of the SHA-1 hash is not an important property of an effective hashing function in this domain. It is sufficient that it is a member of the class of universal hash functions [4].

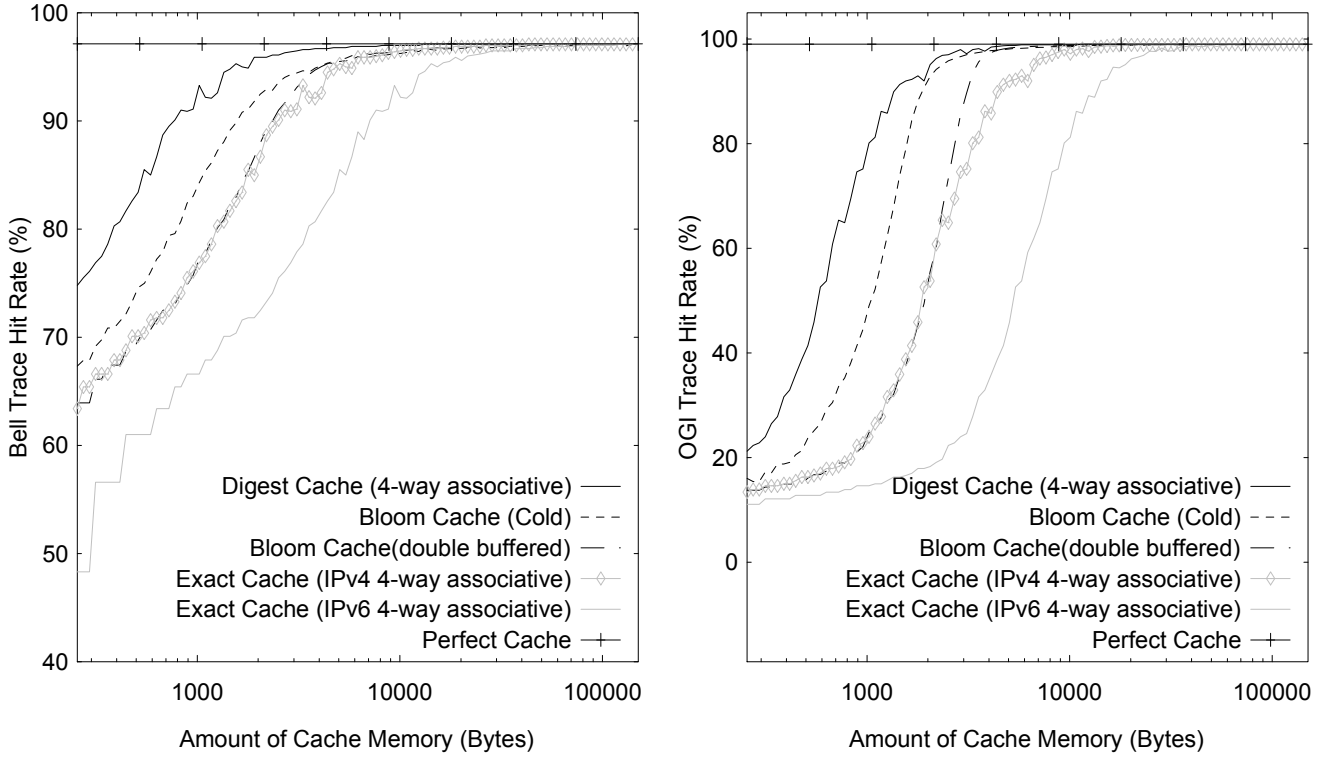

Figure 7: Cache hit rates as a function of memory, M . The Bell trace is on the left, the OGI trace is on the right

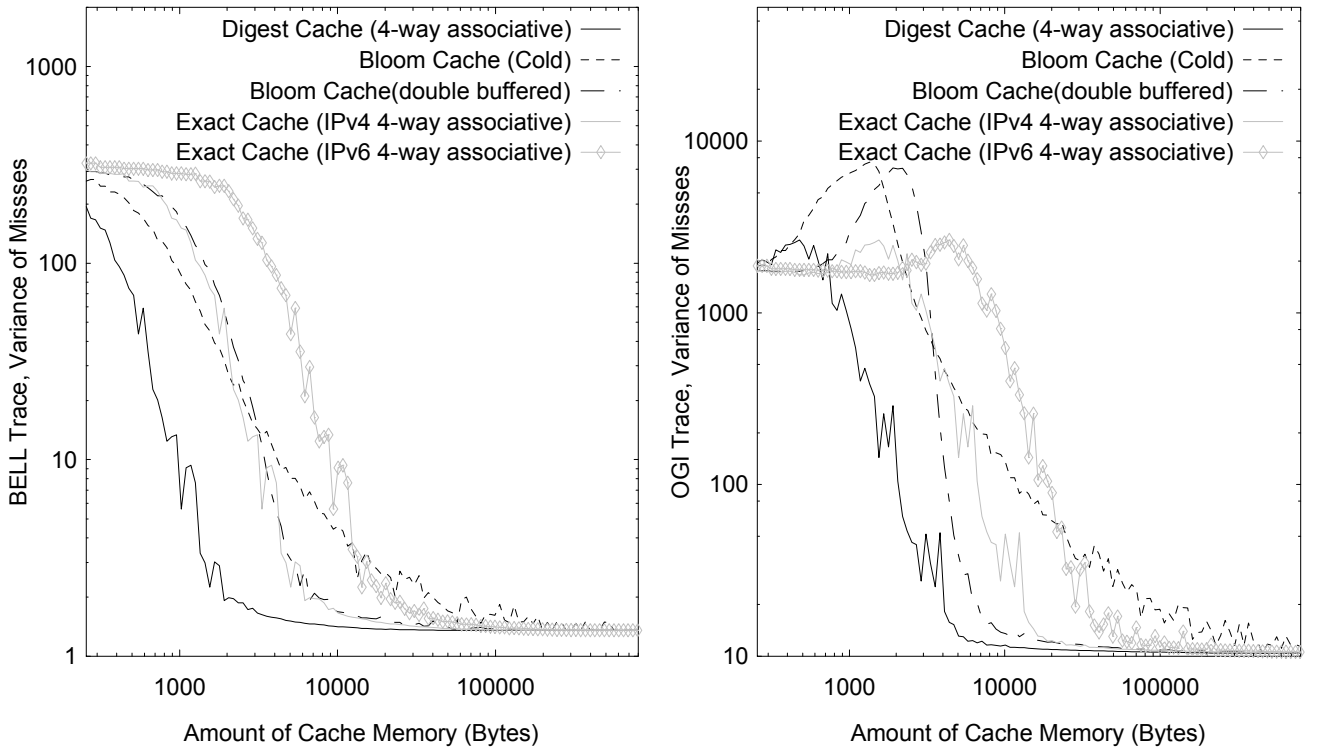

Figure 8: Variance of cache misses as a function of memory, M (aggregate over 100ms time scales). The Bell trace is on the left, the OGI trace is on the right

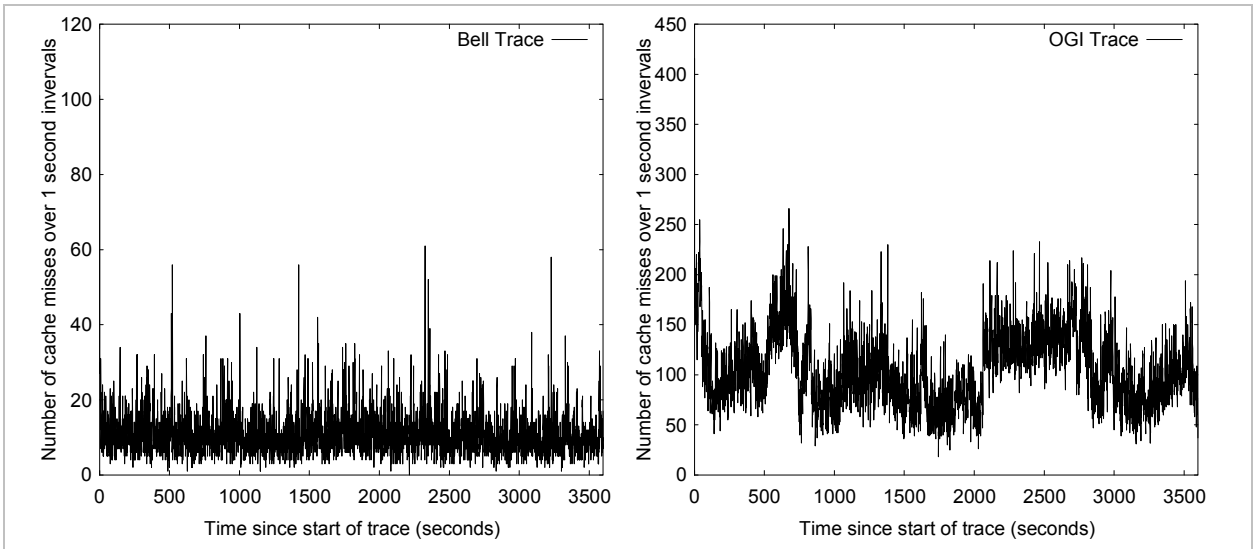

Figure 9: Cache miss rates aggregate over 1 second intervals, using a 2600 byte 4-way set associative digest cache. The Bell trace gave a 95.9% hit rate, while the OGI trace achieved a 97.6% hit rate.

B. Results

In evaluating the performance of the caching systems, we must consider two criteria – we must examine the overall hit-rate as well as the smoothness of the cache miss rate. A cache that has large bursts of cache misses is of no use, because it places strain on the packet classification engine.

Figure 7 graphs the resulting hit rate of various caching strategies, using the sample traces. As expected, the digest cache scores hit-rates equivalent to an IPv6 based cache 10 times its size. More importantly, the digest cache still manages to out-perform a Bloom filter cache. The digest cache yields an equivalent hit rate of a cold-caching Bloom filter 50-80% its size, and out-performs a double-buffered Bloom filter cache 2-3 times its size

Figure 8 graphs the variance of cache miss rates of the different caching approaches, aggregate over 100ms intervals. As can be observed from the two traces, a digest cache gives superior performance, minimizing the variance in aggregate cache misses.

It is interesting to note that for extremely small cache sizes, the digest cache exhibits a greater variance in hit rate than almost all other schemes. This can be attributed to the fact that the other algorithms, in this interval, behave uniformly poor by comparison.

As the cache size increases, this hit rate performance improves, and the variance of cache miss rates decreases to a very small number. This is an important observation because it implies that cache misses, in these traces, are not dominated by bursty access patterns.

To consider a more specific example, we have constructed a 2600 byte 4-way set associative digest cache. This number was chosen to be coincidental with the amount of local memory available to a single IXP2000 family micro-engine.

Figure 9 presents a trace of the resulting cache miss rate, aggregate over one second intervals. This graph represents the number of packets a packet classification engine must process within one second to keep pace with the traffic load. As can be observed from the plot, a packet classification engine must be able to classify roughly 60 packets per second (pps) in the worst case for the Bell trace, and 260 pps in the worst case for the OGI trace. Average packet load during the entire trace is 270.7 and 4335.4 pps for the Bell and OGI traces respectively. Peak packet rate for the Bell trace approached 1400 pps, while the peak rate for the OGI trace exceeds 8000 pps.

By employing a 2600 byte digest cache, the peak stress level on the packet classification engine has been reduced by a factor of between 20 and 30 for the observed traces.

V. HARDWARE OVERHEAD

A preliminary implementation on Intel's IXP1200 Network Processor [17] was constructed to estimate the amount of processing overhead a cache would add. The hardware tested was an IXP1200 board, with a 200 MHz StrongARM, 6

packet-processing microengines and 16 ethernet ports.

A simple micro-engine level layer-3 forwarder was implemented as a baseline measurement. A cache implementation was then grafted onto the layer-3 forwarder code base. A null-classifier was used, so that we could isolate the overhead associated with the cache access routines. The cache was placed into SRAM, because scratchpad memory does not have a pipelined memory access queue, and the SDRAM interface does not support atomic bit-set operations.

The simulation was written entirely in microengine C and performance tests were run in a simulated virtual machine. A trie-based longest prefix match on the destination address is always performed, regardless of the outcome of the cache operation.

A. IXP Overhead

The performance of our implementation was evaluated on a simulated IXP1200 system, with 16 virtual ports. The implementation's input buffers were kept constantly filled, and we monitored the average throughput of the system.

The IXP1200 has a 3-level memory hierarchy, scratchpad, SRAM and SDRAM, each having 4KB, 16MB and 256MB respectively. Scratchpad memory is the fastest of the three, but does not support queued memory access – subsequent scratchpad memory accesses block until the first access is complete. The IXP micro-code allows for asynchronous memory access to SRAM and SDRAM. The typical register allocation schema allows for a maximum of 32 bytes to be read per memory access.

Number of Hash Levels	All-Miss Cache Throughput
0	990 Mb/s
1	868 Mb/s
2	729 Mb/s
3	679 Mb/s
4	652 Mb/s
5	498 Mb/s

Table 3: Performance of Bloom Filter caches in worst case data flows, on a simulated IXP1200

The cache implementation we constructed was designed in a way to ensure that no flow identifier was successfully matched, and each packet required an insertion of its flow ID into the cache. This was done so that the worst possible performance of a Bloom filter cache could be ascertained. The code was structured in a way to disallow any shortcutting or early negative membership confirmation. The performance results of the IXP implementation are presented in Table 3, using a trace composed entirely of small, 64-byte packets. By comparison, a four-way set associative digest cache was able to maintain a sustained average throughput of 803 Mb/s.

The IXP is far from an ideal architecture to implement a Bloom filter, in large part due to its lack of small, high-speed bit-addressable on-chip memory. Ideally, a Bloom filter would be implemented in hardware that supports parallel access on bit-addressable memory [27]. Nevertheless, the performance results presented here serve to underscore the flexibility of our new cache design – specialized hardware is not required.

B. Future Designs

The next generation IXP2000 hardware will feature 2560 bytes of on-chip memory per micro-engine, improving access latencies by a factor of fifteen [9][19].

Let us consider implementing a packet classification cache on this architecture. If we usea this memory for an exact IPv4 cache, we would be able to store a maximum 196 flow identifiers. An equivalent IPv6 cache would be able to store only 69 flows. Using this memory in a 32-bit 4-way set associative digest cache will allow each micro-engine to cache 640 flows.

If we use an 8-bit 4-way set associative exact digest cache, we can use just 1 KB of on-chip memory, and 38KB of SRAM, to store over 1000 flows per micro-engine.

The ability for this algorithm to reduce the amount of memory required to store a flow identifier is especially important in this architecture, because of the limited nature of memory transfer registers. Each micro-engine thread has access to 16 32-bit memory transfer registers, which means that fetching more than one IPv6 flow identifier requires multiple, independent memory accesses, which must be serialized. Since independent memory accesses are significantly more expensive than single, longer memory accesses, this significantly penalizes the performance of a traditional set-associative cache. Coupled with the fact that these memory accesses must be serialized (the first access must complete before the second one can be initiated) the performance benefit of avoiding SRAM memory accesses becomes overwhelmingly important.

For comparison, a modern TCAM implementation can perform 100 million lookups

per second [28]. The IXP2000 can perform 233 million local memory accesses per second [19]. Without even considering the cost or power required to maintain a TCAM, a digest cache becomes a promising alternative.

These arguments make our proposed techniques a prime candidate for creating efficient caches for use on upcoming network processors.

CONCLUSION

Typical packet classification caches trade-off size and performance. In this paper, we have proposed a novel cache architecture that efficiently and effectively uses memory, given a slightly relaxed accuracy requirement. Performance of any existing flow caching solution that employed exact caching can be improved dramatically by employing our technique, at the sacrifice of a small amount of accuracy.

Our new technique is superior to previous Bloom filter approximate caching algorithms, in both theoretical and practical performance while also addressing the shortcomings in the previous Bloom Filter cache design without introducing any additional drawbacks.

This technique can be applied to the design of a novel 2-level exact cache, which can take advantage of hierarchical memory to accelerate exact caching algorithms, with strong results.

ACKNOWLEDGMENT

We would like to thank Ed Kaiser and Chris Chambers for their comments regarding draft versions of this paper. We would also like to thank our anonymous reviewers for their feedback.

REFERENCES

[1] Bloom, B. H. Space/time tradeoffs in hash coding with allowable errors. Communications of ACM 13, 7 (July 1970), 422-426

[2] Baboescu, F. and Varghese, G., Scalable Packet Classification. In Proceedings of ACM SIGCOMM 2001, pages 199-210, August 2001.

[3] Baboescu, F., Singh, S. and Varghese, G., Packet Classification for Core Routers: Is There an Alternative to CAMs?, Proceedings of IEEE Infocom 2003.

[4] Carter, L., and Wegman, M. Universal classes of hash functions. Journal of Computer and System Sciences (1979), 143-154.

[5] Chang, F., Li, K. and Feng, W. Approximate Packet Classificiation, In Proceedings of IEEE INFOCOM'04, Hong Kong, Mach 2004.

[6] Chiueh, T. and Pradhan, P. High Performance IP Routing Table Lookup using CPU Caching In Proc. of IEEE INFOCOMM'99, New York, NY, March 1999

[7] Chiueh, T. and Pradhan, P. Cache Memory Design for Network Processors, Sixth International Symposium on High-Performance Computer Architecture (HPCA 2000)

[8] claffy, k. Internet Traffic Characterization, Ph.D. thesis, University of California, San Diego, 1994

[9] Comer, D. Network Systems design Using Network Processors. Prentice Hall, 2003

[10] Dharmapurikar, S., Krishnamurthy, P., and David E. Taylor Longest Prefix Matching using Bloom Filters, In Proceedings of ACM SIGCOMM'03, August 25-29, 2003, Karlsruhe, Germany.

[11] Feldmann, A., and S. Muthukrishnan, Tradeoffs for Packet Classification, IEEE INFOCOM, 2000

[12] FIPS 180-1. Secure Hash Standard. U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, VA, April 1995

[13] Fraleigh, C., Moon, S., Diot, C., Lyles, B., and Tobagi, F. Packet-Level Traffic Measurements from a Tier-1 IP Backbone. Sprint ATL Technical Report TR01-ATL-110101, November 2001, Burlingame, CA

[14] Gopalan, K. and Chiueh, T. Improving Route Lookup Performance Using Network Processor Cache. In Proceedings of the IEEE/ACM SC2002 Conference

[15] Gupta, P., and McKeown, N. Algorithms for packet classification, IEEE Network Special Issue, March/April 2001, vol. 15, no. 2, pages 24-32

[16] Huitima, C. IPv6: The New Internet Protocol (2nd Edition). Prentice Hall, 1998.

[17] Intel IXP1200 Network Processor, http://www.intel.com/design/network/products/npfamily/ixp1200.htm

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

[19] Johnson, E. and Kunze, A. IXP1200 Programming. Intel Press, 2002.

[20] Lakshman, T. V., and Stiliadis, D., High-speed policy-based packet forwarding using efficient multi-dimensional range matching, In Proceedings of the ACM SIGCOMM 1998, pages 203-214, August, 1998

[21] Li, K., Chang, F., and Feng W., Architecture for Packet Classification, In Proceedings of the 11th IEEE International Conference on Networks (ICON 2003).

[22] McCreary, S., and claffy, k. Trends in wide area IP traffic patterns a view from Ames Internet exchange. In ITC Specialist Seminar, Monterey, California, May 2000

[23] Partridge, C., Carvey, P., et al. A 50 GB/s IP Router. IEEE/ACM Transactions on Networking

[24] Partridge, C. Locality and route caches. NSF Workshop on Internet Statistics Measurement and Analysis (http://www.caida.org/outreach/isma/9602/positions/partridge.html), 1996.

[25] Passive Measurement and Analysis Project, National Laboratory for Applied Network Research (NLANR), available at http://pma.nlanr.net/Traces/Traces/

[26] Saltzer, J., Reed, D., and Clark, D. End-To-End Arguments In System Design, ACM Transactions on Computer Systems, vol 2., no. 4, pages 277-288, 1984.

[27] Sanchez, L., W. Milliken, A., Snoeren, F. Tchakountio, C. Jones, S. Kent, C. Partridge, and W. Strayer. Hardware support for a hash-based IP traceback. In Proceedings of the 2nd DARPA Information Survivability Conference and Exposition, June 2001.

[28] SiberCore Technologies, SiberCAM Ultra-4.5M SCT4502 Product Brief. 2003

[29] Srinivasan, V., Varghese, G., Suri, S. and Waldvogel, M. "Fast and Scalable Layer Four Switching" Proceedings of ACM SIGCOMM 1998, pages 191-202, September, 1998

[30] Stone, J., Partridge, C. When the CRC and TCP checksum disagree, In Proceedings of the ACM SIGCOMM 2000 Conference (SIGCOMM-00), pages 309-319, August 2000

[31] Qiu, L., Varghese, G., Suri, S. Fast firewall implementations for software and hardware-based routers. In Proceedings of ACM SIGMETRICS 2001, Cambridge, Mass, USA, June 2001.

[32] Xu, J., Singhal, M., and Degroat, J. A novel cache architecture to support layer-four packet classification at memory access speeds, In Proceeding of INFOCOM 2000, pages 1445-1454, March 2000.