Efficient Cache Structures of IP Routers to Provide Policy-Based Services

Shingo Ata† Masayuki Murata‡ Hideo Miyahara‡

†Department of Information and Communication Engineering,
Graduate School of Engineering, Osaka City University
E-mail ata@info.eng.osaka-cu.ac.jp

‡Department of Infomatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University
E-mail {murata, miyahara}@ics.es.osaka-u.ac.jp

Abstract— **The policy-based service is becoming more important for the Internet shared by various applications. To realize the policy-based service, the router is required to forward packets according to the requirements of the traffic flow that the packet belongs to. However, the packet (and flow) classification easily becomes a bottleneck because the router is necessary to handle multiple fields within the packet. In this paper, we propose a new packet classification algorithm capable of following the packet forwarding rate of the high-speed routers with the commercially available CPU, RAM, and cache memories. Through our experiments, we show the effects of parameters of the cache structure on the processing rate of the packet classification.**

I. Introduction

A rapid growth of the Internet and proliferation of new multimedia applications lead to demands of high-speed networks. In the today's Internet, policy-based service is becoming more important for meeting various requirements of applications. To realize the policy-based service, the router is required to forward the packets according to the requirement of the traffic flow that the packet belongs to. In addition, some routers providing a flow-based operation are required a capability to store the information of each flow to their storage devices. As a result, the processing on the packet (and flow) classification would easily become a bottleneck because the router is necessary to search multiple fields within the packet for the flow classification.

There are some studies for speeding up the address lookup at the router (see, e.g., [1] and references therein). However, it is difficult to apply those proposals directly to the flow classification, because the size of a lookup table increases exponentially with the number of the packet fields. From this reason, it is necessary to investigate the new algorithm suitable for the flow classification. A study on the flow classification can be found in [2], [3]. However, these proposal use parallel processing to look-up multiple fields simultaneously, and the customized hardware (e.g., FPGA or ASIC) is necessary to obtain the expected performance, leading to easily increase the cost of routers. If we can use a general-purpose CPU, on the other hand, the total cost of the flow classification capable routers can be kept low since the cost of commercially available CPU becomes more and more reasonable. Another approaches can be considered by using CAMs (Content Addressable Memory). However, due to the hardware limitation of CAMs, information of millions of flows cannot be stored simultaneously, which is required for the backbone routers. In such case, using DRAMs is reasonable for the cost of the implementation.

When the flow classification mechanism is realized by the general-purpose CPU, the latency of memory accesses would be most dominant in determining the performance. That is, the optimized data structures determines the performance of the flow classification by decreasing the number of accesses to RAM. In this paper, we propose a new flow classification algorithm capable of following the packet forwarding rate of the high-speed routers by using the commercially available CPU, RAM, and cache memories. Then, through our experiments, we show the effects of the cache memory structures on the processing rate of flow classification.

In this paper, we first describe the flow classifier and the memory requirement to store the flow statistics in Section II. We next show the brief review of the cache memory structure and the effect of parameters of the cache memory in Section III. We then examine the proposed flow classification scheme in Section IV. Section V shows our experimental results. Finally, we conclude our work with future research topics in Section VI.

II. Definition of Flow Classifier

In this section, we define our flow classifier based on the statistical analysis of the flow data gathered at the operating network. Requirements on the memory structure to store values associated with each flow are also examined.

A. Flow Classifier

It is reasonable to consider the sequence of packets belonging to the same connection as the same flow. Henceforth, it might be adequate to identify the flow using the following fields contained in the IP packet.

{*Source IP, Destination IP, Source Port Number, Destination Port Number, Protocol*}

However, the total length of the above fields reaches 104 bits, which is too large for the size of lookup table. Fortunately, we can disregard the *Protocol* field for flow classification since it shows the kind of the IP packets such as TCP, UDP, ICMP, and RAW IP, and few applications have two or more *Protocol* values. Furthermore, HTTP, which is used by a currently dominant application WWW, supports multiple connections for retrieving text and/or image files. Those connections can be treated as an aggregated single flow. It is meaningful when we consider the fairness among users in flow classification. We therefore use the following fields as a flow classifier.

{*Source IP, Destination IP, Port Number*}

Note that *Port Number* above is the lower value of two port numbers (*Source Port Number* and *Destination Port Number*). It is because most applications use the administrative port number smaller than 1024 that the application user cannot use.

B. Requirements for Storing the Flow Values

The flow information to be stored into the router memory depends on service policy. In this subsection, we examine some existing important applications of the policy-based service, and esti-

mate the required memory size to store the value associated with the flow for given service (which we will call a *flow value* for short). We consider three types of applications; **Random Early Detection Based on Flow Classifications** [4], [5], **Application to Core-Stateless Routers** [6], and **Fairness with the Deficit Round Robin** [7]. We conclude from these examples that 2 Bytes is sufficient to store the flow value. Of course, there may be other policy services requiring more bytes. However we continue to discuss the memory structure by assuming 2 Bytes flow value in this paper. Note that our method can easily be extended to handle more bytes of the flow value, but in that case, the required memory capacity is increased.

III. Characteristics of the Cache Memory

Before describing our flow classification algorithm and memory structure for the lookup table, we show the brief description on the cache structure to investigate the effects of parameters of the cache memory.

A. The Structure of the Cache Memory

A two-level cache structure is common in the recent CPU technology. A memory access from CPU is first reached at L1 (Level 1) cache to examine whether the address is cached or not. If it is cached, CPU reads from the L1 cache. Otherwise, the L2 (Level 2) cache is checked. If the address is not cached in the L2 cache, the address content is read from RAM. We often call a *cache hit* for the case where the address is cached, and *cache miss* for the case where the address is not cached. Note that a more detailed caching mechanism (such as write-through or write-once) may affect the performance of flow classification, but it is out of scope in this paper.

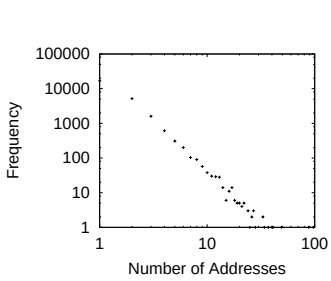
B. Effect of Parameters

Here, we examine three parameters of the cache memory; the cache block size, the degree of set associative mapping, and the cache size.

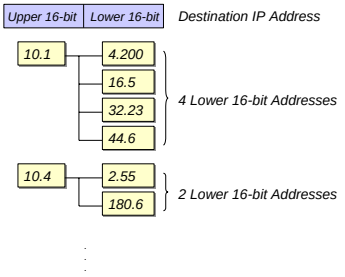
Effect of the Cache Size — It is necessary to increase the capacity of the cache memory to improve the cache hit rate. However, as the cache size becomes large, the cost of the cache memory increases exponentially. Furthermore, too large cache memory does not necessarily improve the cache hit ratio. It is therefore important to investigate the appropriate cache size for classifying flows efficiently in our case.

Effect of the Set Associative Mapping — Set associative mapping is used to increase the utilization of the cache memory. When the associativity is set to be large, the cache memory is highly and impartially utilized. However, the large associativity is too expensive.

Effect of the Cache Block Size — The cache block is a unit of data transfer between L1 and L2, and between L2 and RAM. When the cache miss occurs, the cache memory reads the block of the memory. If the block size is large, the continuous address can be cached. It is therefore expected to improve the cache hit rate. However, the larger block size leads to larger delay of a memory read.



(a) The Frequency Distribution



(b) Examples; the upper 16-bit IP address '10.1' was followed by 4 lower 16-bit addresses. ('4.200', '16.5', 32.23' or '44.6')

Fig. 1. Th Neumber of Lower 16-bit Addresses

IV. Flow-Value Lookup Procedures

A. Partitioning the Flow Classifier

The delay for memory accesses is a dominant factor for determining performances of flow classification and its lookup. Of course, the delay of the table lookup would be minimized by one memory access if we can prepare a large lookup table. The size of such a table, however, reaches 2×2^{80} Bytes for our 80 bits flow classifier and 2 Bytes flow value. Because there is a trade-off between the number of memory references and the required table size, we need to decide the structure of the flow lookup table by carefully examining both the memory size and the access speed.

We now demonstrate the sparse nature of the IP packet addresses by using the traced data. We gathered the traced data by the traffic monitor OC3MON [8], [9] located at the gateway of Osaka University. The total number of collected packet headers is over 27 millions. In this traced data, we need to store about 9 millon of flow values.

Figure 1(a) plots a frequency distribution for the number of lower 16-bit destination IP addresses associated with the upper 16-bit destination IP address. More specifically, we plot the figure as follows. For each of 32-bit destination IP address, we divide it into two parts; upper and lower 16-bit addresses. Then, we count the number of different lower 16-bit addresses having the same upper 16-bit address in the traced data. For example, there were 10 IP addresses, each of which shows 21 different lower 16-bit addresses in the traced data. From the figure, it is apparent that it follows the Zipf's Law [10] with parameter $\alpha = 1$. The lower 16-bit of addresses which did not appear in the traced data was counted up over 40,000. The ratio of such addresses is about 60%. Moreover, the maximum number of lower 16-bit addresses was 97, which is less than 1% of the 16 bit address space.

We also consider the number of varieties of (Source IP address, Port Number) fields categorized by the destination IP address. We count the number of (Source IP Address, Port Number) pairs for each destination IP address. The result also follows the Zipf's Law. It means that there are very few destination addresses having a large number of pairs of (Source IP Address and Port Number), and the most of destination addresses have a very small number of (Source IP Address, Port Number) pairs.

To check the generality of our results, we also examined the

public trace archive [11], which is not shown in this paper due to the space limitation.

In summary, the flow classifier should be divided into several parts to improve memory utilization even though the number of memory references increases to some degree. In our scheme, we divide the flow classifier into the following three parts; the upper 16-bit of the destination IP address, the lower 16-bit value of the destination IP address, and the source IP address + port number.

B. Table Compression with the Hash Function

As shown in Figure 1(a), the maximum number of different upper 16-bit addresses for the lower 16-bit addresses is less than 1% of the lower 16-bit address space. In such case, the hash table is useful to reduce the size of the lookup table to obtain higher utilization of the lookup table. It becomes helpful especially when the number of active entries is quite smaller than the number of total entries in the table. In the above case, the hash table with 256 entries is enough to store all lower 16-bit destination addresses. Because the number of entries is reduced from 65,536 to 256, different lower addresses may have the same hash index. When the entry has already been used by another lower-address, the algorithm checks the next entry of the hash table. A step-by-step check is meaningful to reduce the number of memory accesses because the CPU caching is performed for each block. On the other hand, the result in the trace archive shows that maximum number of lower 16-bit addresses is 3598. It was much larger than our traced data (97 entries), but it is still smaller than the 16-bit address space. We can easily expand our table entries (i.e., from 256 to 4096).

We can also reduce the size of the (Source IP Address, Port Number) table by using the hashing technique. From our experiments, the maximum number of (Source IP Address, Port Number) pairs is 766. The table with 1,120 entries is sufficient. It is required for 40 blocks in the 32 Bytes block size.

Note that we always assign the maximum number of entries for each destination address to obtain the maximum speed of the flow classification (i.e., avoiding the overhead of the table expansion), which still leads to the under-utilization of memory structures. The memory size requirement can be reduced if we can change the number of entries of the lookup table dependent on the destination address. However, it is out of scope in the current paper.

C. Algorithm of Flow Classification and Lookup

We now describe our flow classification algorithm and lookup procedunre. Figure 2 shows the data structure of our classification algorithm. The structure of the lookup table consists of three major parts.

1. Upper 16-bit destination IP address lookup table:
   It is a directly accessible table (65,536 entries), and each entry contains a pointer to the second table.
2. Lower 16-bit destination IP address lookup table:
   It is a 256-entry hash table, and each entry contains a pointer to the third table.
3. Source address lookup table:
   The source address consists of the combination of (Source IP address, Port Number), and it is a hash table having 1,120 entries. Each entry has
   - Destination IP address for verification
   - A hash value of (Source IP address, Port Number)

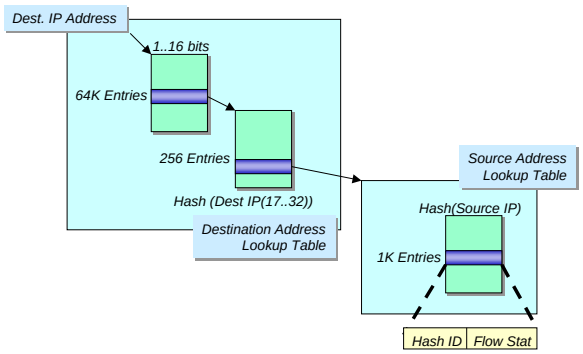


Fig. 2. Proposed Data Structure

   - The value associated with the flow

We describe the data structure of the third lookup table for the case where when the block size of the cache memory is 32 byte. The first 4 byte is used to store the destination IP address. It is used for checking whether the destination address in the packet is equal to the stored value or not. If the verification succeeds, no memory reference is necessary because the flow value is cached. In this structure, $(32-4)/4 = 7$ entries can be stored into one cache block. The required number of blocks becomes 160 to create the hash table with 1,120 entries.

We finally show the address calculation algorithm for the flow lookup.

```
1. Ptr1 = HeadTable[Dest Addr(1..16)]
2. Ptr2 = Ptr1[Hash(Dest Addr (17..32))]
3. statePtr = Ptr2 × 32 × 160 + 32 ×
   (Hash(Src, Port) / 7) + (Hash(Src, Port)
   mod 7) + 4
```

V. Numerical Experiments and Discussions

In this section, we show our experimental results on the flow lookup delay and the effect of parameters of the cache structure.

A. Experimental Setup

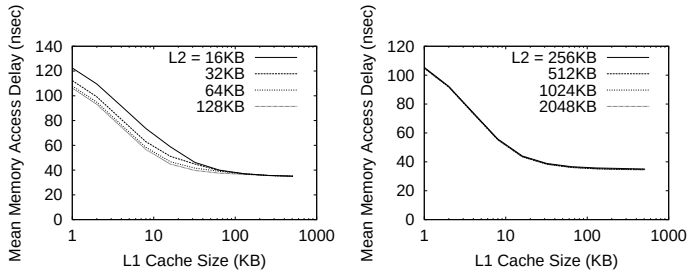
We implemented our proposed scheme on the following architecture.

- Intel Pentium II 450 MHz
  - 16 KB 4 way set associative L1 cache
  - 512 KB 4 way set associative L2 cache
  - 256 MB main memory
- Linux 2.0.36
- GNU gcc version 2.95.2 19991024 (release)

We used the packet traced data that we have described in Section 2.

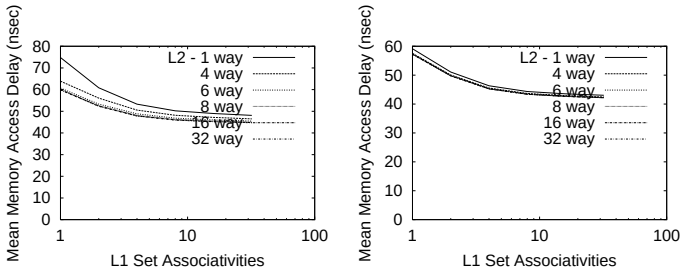
To investigate the effect of parameters of the cache memory, we used trace driven cache simulator, Dinero IV version 7 [12]. Dinero IV reports the total number of memory references N, and the cache miss ratios of L1 and L2 (denoted by M_{L1} and M_{L2}, respectively). If we have the reference delays of memory, L1, and L2 cache (N_{Mem}, N_{L1}, and N_{L2}, respectively), we can determine the mean access delay D for the flow classification as

$$\begin{aligned} D &= N/P \times ((1-M_{L1})N_{L1} \\ &\quad +M_{L1}(1-M_{L2})N_{L2} \\ &\quad +M_{L1}M_{L2}N_{Mem}) + Calc \qquad [\text{cycles}], \end{aligned} \tag{1}$$



(a) L2=16 KB – 128 KB (b) L2=256 KB – 2048 KB

Fig. 3. The Effect of the L1 Cache Size



(a) L1=16 KB, L2=64 KB (b) L1=16 KB, L2=512 KB

Fig. 4. Effect of the Set Associative Mapping

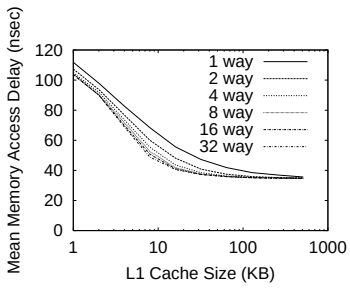


Fig. 5. Relation between Cache Size and Associativity (L2 = 512 KB, 4-way)

where P is the total number of packets and $Calc$ is the delay of address calculations which is independent from the memory access.

To obtain the delays of memory references (N_{Mem}, N_{L1}, and N_{L2} in (2)), we run a simple benchmark program on our experimental architecture, which collects the number of CPU cycles for reading from L1 cache, L2 cache, and RAM memories. From results of our benckmark program, we get reference delays of memory, L1, and L2 as about 6.7 nsec, 44.4 nsec, and 155.6 nsec, respectively.

B. Effects of Cache Memory Sizes

We first show the effect of the L1 cache size on the average access delay. Note that the average memory access delay is affected not only by the L1 cache size, but also by the size of L2 cache. We thus evaluate the effect of the L1 cache size with various L2 cache sizes from 16 KB to 2048 KB.

Figures 3(a) and 3(b) show the mean access delay against the L1 cache size by utilizing the cache simulator Dinero IV. In Figure 3(a), four different values of the L2 cache size are used; 16 KB, 32 KB, 64 KB, and 128KB. On the other hand, 256 KB, 512 KB, 1024 KB and 2048 KB are used in Figure 3(b). The mean access delay can be decreased by the increased size of the cache memory. The effect of the L1 cache size is significant when the L1 cache size is between 1 KB and 64 KB. However, we cannot observe the improvement as the L1 cache size goes beyond 64 KB. Namely, the currently available L1 cache size of Intel Pentium II (and III) is slightly smaller than the expected one. In Figure 3(a), the mean delay can be improved by the larger size of L2 cache, but such an improvement cannot be seen in Figure 3(b). We can observe from these figures that the 256 KB L2 cache is sufficient in our experiments.

We next evaluate the effect of the L2 cache size when the size of L1 cache is varied from 1 KB to 256 KB. We can see that the effect of the L2 cache size is significant when the size of L1 cache is small. However, the increase of L2 cache does not much affect the mean access delay if the L1 cache size is larger than 128 KB.

C. Effects of Set Associative Mapping

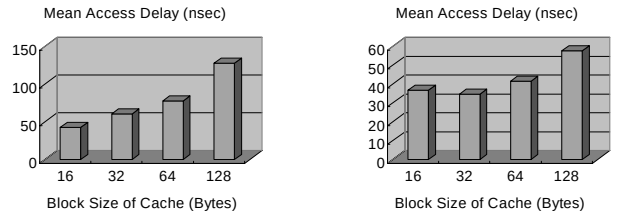
Figures 4(a) and 4(b) show the mean access delays against the set associative mapping method. The size of the L1 cache is fixed at 16 KB. The L2 cache sizes are 64 KB in Figure 4(a), and 512 KB in Figure 4(b), respectively. The effect of set associative mapping is significant when the cache size is small as shown in Figure 4(a). However, even if we change L1 set associativities from 8-way to 16-way, the mean access delay are not changed. From these figures, we can observe that the expected associativities of L1 cache is 8-way, but a currently available CPU (4-way) also gives a good performance.

We also examine the effect of L2 set associative mappings. By contrast with Figure 4, the effect of L2 set associative mapping is limited. In particular, the L2 set associativity does not affect the mean access delay when the L2 cache size is 512 KB. It is because as the cache size becomes large, the cache miss by the address conflict is rare.

Figure 5 plots the relation between the cache size and the associativity of set mappings. The figure shows that the effect of set associative mapping is remarkable if the L1 cache size is between 8 KB and 32 KB. For example, when the L1 cache size is 16 KB, changing associativities from 2-way to 4-way gives good results; i.e., a similar effect can be obtained as doubling the size of L1 cache.

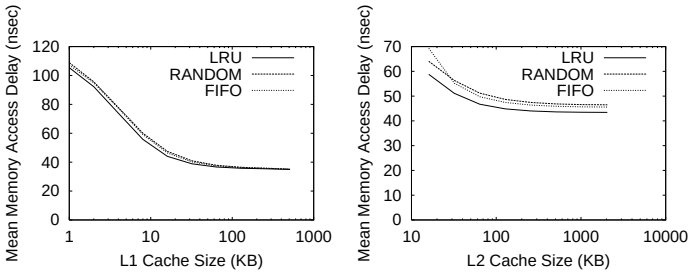
D. Effects of Cache Block Size

Figures 6(a) and 6(b) plot the mean access delay dependent on the cache block size. The L1 cache size is set to be 8 KB in Figure 6(a), and 64 KB in Figure 6(b). The advantage of the larger cache block size is that it can be cached at the continuous addresses, and be expected to increase the cache hit rate. However, the cache miss penalty becomes large in the case of cache miss. As a result, one may expect that there exists an optimal block size. However, Figure 6(a) shows that the mean access delay is exponentially increased as the block size becomes large. In Figure 6(a), the L1 cache size is small (8 KB) and the cache miss ratio increases significantly by the larger block size. Because many packets of various connections are mixed at the IP router, accesses to the same address does not occur continuously, but periodically. It causes frequent cache replacements when the cache size is small. On the



(a) L1=8 KB, L2=512 KB (b) L1=64 KB, L2=512 KB

Fig. 6. Effect of the Cache Block Size



(a) L1=1–512 KB, L2=256 KB (b) L1=16 KB, L2=16–1024 KB

Fig. 7. Comparisons among Cache Replacement Policies

other hand, if the cache size is large, the effect of the large block size overcomes above disadvantages as shown in Figure 6(b).

E. Effect of the Cache Replacement Policy

In this subsection, we examine the effect of the cache replacement policy in the set associative cache memory. In our experiments, we use three replacement policies; LRU (least recently used), FIFO (first in first out), and RANDOM. Figures 7(a) and 7(b) compare the mean memory access delay among three cache replacement policies by changing the size of L1 or L2 cache. From these figures, LRU gives the best performance, and FIFO is better than RANDOM. Especially, the effect of cache replacement policy is remarkable when the cache size is small. However, the implementation of LRU policy is very difficult even in 4-way set associative mapping. From this reason, FIFO is still reasonable if a slight performance degradation is acceptable.

F. CPU Cycles and Required Memory Size

We finally show the average CPU cycles necessary for the packet classification by using RDTSC instruction in Pentium II [13]. The sizes of L1 and L2 caches and associativities are determined based on 450 MHz Intel Pentium II. That is, we used 16 KB 4-way set associative L1 cache and 512 KB 4-way set associative L2 cache. Our result shows that the average CPU cycles is 228.8 (e.g., 506 nsec). This result indecates that our implementation can classify about 2 million packets per second if we neglect other operations. The required memory size was 49.1 MB. It is rather large since lookup table size is fixed for each flow in the current implementation. If we can change the size of lookup table dependent on the flow, the required memory size could be reduced, which is our future research topic.

VI. Concluding Remarks

In this paper, we have proposed a new packet classification algorithm capable of following the packet forwarding rate of the high-speed routers with the commercially available CPU, RAM, and cache memories. Through our experiments, we have shown that the L1 cache size gives a great impact to the performance of the flow classification, but the effect cannot be found if the sufficient L1 cache exists. We have also shown that the set associative mapping is effective when the cache size is small, and sometimes gives the same performance improvement to double the cache size. The effect of cache block size has also been examined. Our results have shown that the effect of the large block size would be appeared if the cache size is large. Finally, we have checked the effect of cache replacement policies. From numerical results, we have shown that LRU gives a best performance, but FIFO is still reasonable if we consider the implementation cost. In this paper, we always assign the maximum number of entries for flow lookup tables to avoid the overhead of the table expansion. However, it leads to the under-utilization of memory structures. The memory size requirement can be reduced if we can change the number of entries of the lookup table dependent on the destination address. It is our future research topic.

Acknowledgements

This work was supported in part by Research for the Future Program of Japan Society for the Promotion of Science under the Project “Integrated Network Architecture for Advanced Multimedia Application Systems”(JSPS-RFTF97R16301).

References

[1] Tzi-cker Chiueh and P. Pradhan, “High–performance IP routing table lookup using CPU caching,” in *Proceedings of IEEE INFOCOM ’99*, vol. 3, pp. 1421–1428, April 1999.

[2] T. V. Lakshman and D. Stiliadis, “High–speed policy–based packet forwarding using efficient multi–dimensional range matching,” in *Proceedings of ACM SIGCOMM ’98*, pp. 203–214, September 1998.

[3] P. Gupta and N. McKeown, “Packet classification on multiple fields,” in *Proceedings of ACM SIGCOMM ’99*, pp. 147–160, September 1999.

[4] D. Lin and R. Morris, “Dynamics of random early detection,” in *Proceedings of ACM SIGCOMM ’97*, pp. 127–137, September 1997.

[5] T. J. Ott, T. V. Lakshman, and L. Wong, “SRED: Stabilized RED,” in *Proceedings of IEEE INFOCOM’99*, pp. 1346–1355, March 1999.

[6] I. Stoica, S. Schenker, and H. Zhang, “Core–stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks,” in *Proceedings of ACM SIGCOMM ’98*, pp. 118–130, September 1998.

[7] M. Shreedhar and G. Varghese, “Efficient fair queueing using deficit round robin,” *IEEE/ACM Transactions on Networking*, vol. 4, pp. 375–385, June 1995.

[8] K. Thompson, G. J. Miller, and R. Wilder, “Wide–area Internet traffic patterns and characteristics,” *IEEE Network*, vol. 11, pp. 10–23, November 1997.

[9] S. Ata, M. Murata, and H. Miyahara, “Analysis of network traffic and its application to design of high-speed routers,” *IEICE Transactions on Information and Systems*, vol. E83-D, pp. 988–995, May 2000.

[10] M. Aida, N. Takahashi, and T. Abe, “A proposal of dual Zipfian model for describing HTTP access trends and its application to address cache design,” *IEICE Transactions on Communications*, vol. E81–B, pp. 1486–1496, July 1998.

[11] National Laboratory for Applied Network Research, “NLANR network traffic packet header traces.” `http://moat.nlanr.net/Traces/`.

[12] J. Edler and M. D. Hill, “Dinero IV : Trace-driven uniprocessor cache simulator,” `http://www.cs.wisc.edu/~markhill/DineroIV`, 1994.

[13] Intel Corporation, “Using the RDTSC instruction for performance monitoring,” *Pentium II Application Note*, `http://developer.intel.com/drg/pentiumII/appnotes/RDTSCPM1.HTM`, 1998.