DEC-TR-592

Characteristics of Destination Address Locality in Computer Networks: A Comparison of Caching Schemes

Raj Jain

Digital Equipment Corporation
550 King St. (LKG1-2/A19)
Littleton, MA 01460

Network Address: Jain%Erlang.DEC@DECWRL.DEC.COM

Raj Jain is now at
Washington University in Saint Louis
Jain@cse.wustl.edu
http://www.cse.wustl.edu/~jain/

February 1989

This report has been released for external distribution.

Characteristics of Destination Address Locality in Computer Networks: A Comparison of Caching Schemes

Raj Jain
Distributed Systems Architecture & Performance
Digital Equipment Corp.
550 King St. (LKG 1-2/A19)
Littleton, MA 01460
ARPAnet: Jain%Erlang.DEC@DECWRL.DEC.COM

DEC-TR-592

Version: June 25, 1990

Abstract

The size of computer networks, along with their bandwidths, is growing exponentially. To support these large, high-speed networks, it is necessary to be able to forward packets in a few microseconds. One part of the forwarding operation consists of searching through a large address database. This problem is encountered in the design of adapters, bridges, routers, gateways, and name servers.

Caching can reduce the lookup time if there is a locality in the address reference pattern. Using a destination reference trace measured on an extended local area network, we attempt to see if the destination references do have a significant locality.

We compared the performance of MIN, LRU, FIFO, and random cache replacement algorithms. We found that the interactive (terminal) traffic in our sample had quite different locality behavior than that of the noninteractive traffic. The interactive traffic did not follow the LRU stack model while the noninteractive traffic did. Examples are shown of the environments in which caching can help as well as those in which caching can hurt, unless the cache size is large.

1 INTRODUCTION

The fact that page references by computer programs exhibit locality behavior is now well established and designing computer systems without virtual memory and memory caches is practically inconceivable [20,27]. In the 1970s there were a large number of studies of program behavior [16,26] that helped design several good page replacement algorithms and caching strategies. In the 1980s, with the increasing trend towards distributed computing, the caching of files (located remotely) and the study of file reference behavior became an interesting topic [5,6,13,14,17,21,23,28,29].

Recently, we discovered that the frames on computer networks also exhibit locality behavior [9]. The understanding of this behavior will help us design the large networks of the 1990s in an efficient manner.

The trend toward networks becoming larger and faster, and addresses also increasing in size has impelled a need to understand and exploit the locality, if one exists. DECnet Phase IV currently allows up to 64,000 nodes and DEC's internal networki, called EasyNet [18], has more than 30,000 nodes. Such large networks obviously need more efficient address lookups. The size of the addresses themselves is also growing. HDLC, a commonly used datalink protocol standard, was designed with 8-bit addresses. All IEEE 802 LAN protocols support 48-bit addresses and the ISO/OSI network layer requires 160-bit (20 octets) addresses. This increased length of the key has also necessitated a need to find efficient ways to look up addresses. Finally, as networks are becoming faster, network routers, which previously handled a few hundred frames per second, are now expected to handle 8000 to 16,000 frames per second. This fast handling requires squeezing every cycle out of the frame forwarding code.

The realization that the frame destinations exhibit locality behavior makes caching a possible alternative for efficiently supporting large networks. By caching the destinations recently seen, the intermediate nodes can avoid looking through large tables of nodes with a high probability. The address space need not be hierarchical, the caching works with flat as well as hierarchical address. Caching is transparent in that no protocol changes are generally required to accommodate caching and noncaching implementations in the same network.

The cost of memory chips has been falling rapidly, however, their access times have not decreased as fast. As a result, although the cost of the memory to hold these large address databases may not be a significant consideration (as was the case for development of virtual memory)i, but the access time of the address database is the major reason for our need to find efficient ways to look up addresses. Caching allows such decisions to be made correctly within the specified time limit with a high probability. Since incorrect decisions may result in frames being retransmitted, the cache should be designed so that a very low miss probability will result, typically less than 0.1%. This should be contrasted with page replacement algorithms, where miss probability of 10% may be considered acceptable.

In this paper, we are concerned with the problem of address recognition in bridges. However, there are a number of other applications in computer networks where caching can help avoid searching through a number of entries. For example, datalink adapters can use caching to search through the list of multicast addresses. The network adapter board [11] uses caching to help decode the received frame header. Routers and gateways can cache forwarding databases. Also, name servers and their clients can use caching to improve the efficiency of name lookup. Although, the conclusions of our reference trace are not applicable to these other applications, our methodology, when applied to traces of these applications, can be used to find the appropriate caching strategy.

The organization of this paper is as follows. First, we describe the environment in which the address trace was measured. Second, we explain various locality concepts and analyze the applicability of different locality models. We then compare the performance of various cache replacement algorithms.

2 Measured Environment

In order to compare various caching strategies, we used a trace of destination addresses observed on an extended local area network in use at Digital's King Street, Littleton facility. The network consists of several Ethernet LANs interconnected via bridges. The network is a part of Digital's company-wide network called EasyNet [18], which has more than 30,000 nodes. The building itself has approximately 1200 nodes on several Ethernet LANs interconnected via bridges. There are 30 Level-1 routers, six Level-2 routers, and approximately 80 bridges in the building. A promiscuous monitor attached to one of the Ethernet LANs produced a time-stamped reference string of approximately 2 million frames. For some analyses, we subdivided the trace into 11 subtraces of approximately 200, 000 frames each. The characteristics of these subtraces along with that of the complete trace are listed in Table 1.

Table 1: Trace Characteristics

Subtrace		Addresses		
#	Frames	Total	Dest.	Hours
1	200000	460	244	0.12
2	200000	450	208	0.12
3	200000	449	210	0.11
4	200000	437	210	0.11
5	200000	435	203	0.11
6	200000	436	204	0.10
7	200000	444	201	0.11
8	200000	433	205	0.10
9	200000	424	210	0.09
10	200000	431	207	0.10
11	46000	379	186	0.02
Total	2046000	495	296	1.09

The total column includes addresses in destination as well as source fields of the frame. This number is approximately equal to the number of stations on the extended LAN since all stations periodically broadcast a 'hello' message to indicate their presence on the network. Not all addresses appear in the destination address field since only a fraction of individually addressed (unicast) frames pass through the monitored LAN. For example, in subtrace 1, there were 460 distinct addresses; of these, only 244 appeared in the destination address fields. Due to bridge filtering, only those frames whose desinations have a short path through the monitored segment are seen on the segment. The hour column gives the duration of the subtrace in hours. As shown in the table, the complete trace was a result of approximately one hour of monitoring.

There are several advantages and disadvantages of using a trace. A trace is more credible than references generated randomly using a distribution. On the other hand, traces taken on one system may not be representative of the workload on another system. We hope that others will find the methodology presented here useful and will apply it to traces taken in environments relevant to their applications.

3 Locality: Concepts

In this section we review some of the well-known concepts about locality. These concepts were developed during studies of page reference patterns, but apply equally well to file reference or destination reference patterns. In the following discussion, the term *address* refers to page, file, or the destination node encountered.

The locality of a reference pattern may be temporal or spatial. Temporal locality implies a high probability of reuse. For example, the reference string $\{3, 3, 3, 3, 3, \ldots\}$ has a high temporal locality, since the address 3 is used repeatedly once it is referenced. Spatial locality implies a high probability of reference to *neighboring* addresses. For example, the string {1, 2, 3, 4, 5, ...} has a high spatial locality since after a reference to address k, the probability of reference to $k+1$ is very high. While the definition of *neighboring* addresses is somewhat clear for page and file addresses, it is not so clear for networks. Spatial locality, if present, is useful in designing prefetching algorithms since the information likely to be used in the near future is fetched before its first reference, thereby, avoiding a *cache miss*. Page reference patterns exhibit both temporal as well as spatial locality.

The terms *persistence* and *concentration* have also been used to characterize locality behavior [2]. Persistence refers to the tendency to repeat the use of a single address. This is, therefore, similar to temporal locality. Concentration, on the other hand, refers to the tendency of the references to be limited (concentrated) to a small subset of the whole address space. A reference string with high concentration is good in that a small cache would produce large performance gains. Persistence can be measured by counting consecutive references to the same address, while concentration can be measured by computing the fraction of address space used for a large fraction of the reference string. For example, in a reference string with high persistence, the probability of the same address being referenced consecutively may be 60%, for instance. Similarly, in a string with high concentration, 99% of the references may be to 1% of the address space. Bunt and Murphy [2] have done extensive studies of persistence and concentration in memory and file reference strings.

Virtual memory is one of the first applications of locality concepts in computer systems design. The pages actively being used are kept in the physical (cache) memory. The key differences between virtual memory, file caching, and destination address caching are summarized in Table 2. In virtual memory systems, a very large cache (physical memory) gives better performance, but is too expensive. In remote file systems, large local caching not only requires large local memory, but also results in a large amount of information being transported over the network. Thus, in this case, there is an optimal cache size over which the caching does not pay. This is true for destination address caching too. If the cache is too big, the search time is large and caching is not useful. Too small caches may result in too many page faults in virtual memory systems or too many network accesses in remote file systems. In either case, the system has to wait while the information is being fetched, causing increased response time. This is also true for destination address caching. A long delay in address look up may result in the source retransmitting the frame. The cache miss rate has to be kept low. Acceptable miss rates range from 0.1% to 10% depending upon the ratio of lookup time with and without the cache. A larger ratio would increase the probability of retransmissions and would need a smaller miss ratio.

4 Models of Reference Behavior

A number of models have been developed for page reference behavior. These well known models are the independent reference model (IRM), the least recently used (LRU) stack model, and the working set (WS) model. In the following subsections, we describe these models and see their applicability to our address reference trace.

Table 2: Locality in Page vs File vs Node References

	Page	File	Node
Year needed	1970	1980	1990
Why needed	Large programs	Remote files	Large networks
Why not infinite cache	Memory cost	Memory cost & comm. overhead	Access time
Cost of a miss	Page fault	Network access	Packet lost or delayed
Effect of a high miss rate	Thrashing		Instability
Good miss rate	10%	1%	0.1% to 10%

4.1 Independent Reference Model

The independent reference model assumes, as the name implies, that the references are independent [19]. Knowing that the last reference was to address k does not give any information about the next address to be referenced. In other words, this model assumes that the reference strings do not have any temporal or spatial locality. The probability of reference to address i is p_i, and all p_i's need not be equal. In a more restricted IRM, called Uniform-IRM, the probability p_i's are assumed to be all equal. This is equivalent to assuming that there is no concentration of references.

Figure 1 show the cumulative frequency of reference as a function of fraction of distinct addresses seen in the trace. Notice that the destination reference probability is nonuniform. For uniform probability, the curve would have been a straight line between (0%, 0%), and (100%, 100%). The median and 90-percentile points on the curves are listed in Table 3.

Notice that 50% of the frames are destined to 4% of the destinations and that 90% of the frames are destined to 17% of the destinations. Thus, destination references exhibit a strong *concentration*. This is a good news since it implies that if we cache highly probable destinations, we may get high hit rates with small caches.

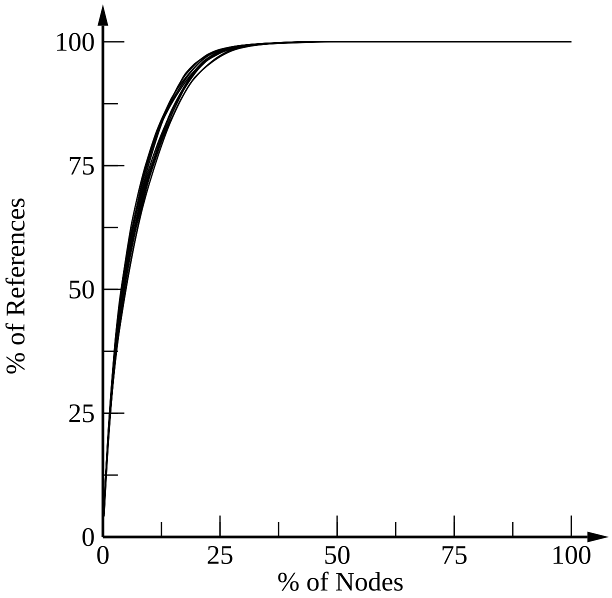


Figure 1: Percentage of frames vs percentage of destinations.

Table 3: Cumulative Percentage of References

Subtrace	Median	90-Perc
1	4.1	15.7
2	4.2	15.6
3	4.7	16.9
4	4.8	16.9
5	5.1	17.7
6	5.0	17.0
7	4.5	15.5
8	4.4	16.4
9	4.0	16.0
10	4.6	17.2
11	4.7	17.2
Total	4.4	17.8

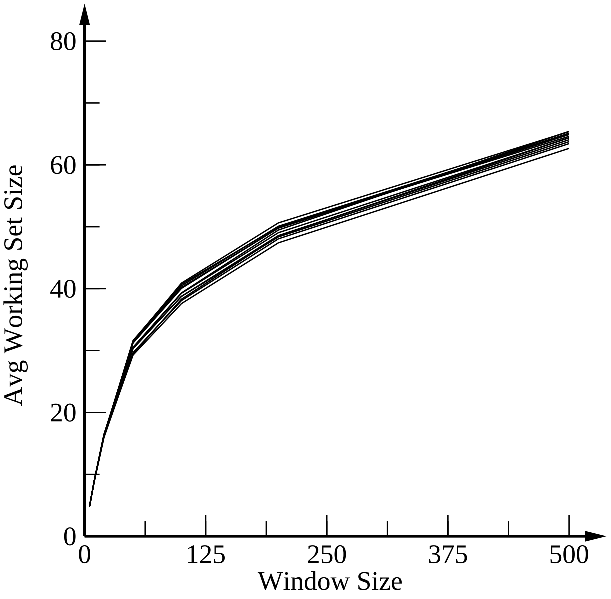


Figure 2: Working set size.

Another distinct feature of Figure 1 is that all subtraces have almost identical behavior. Since these traces consist of traffic during different time intervals on the same network, the observed behavior does not seem to be a reflection of a short-term activity.

5 Working Set Model

The working set model [3] assumes that the addresses referenced in the last W references are highly likely to be rereferenced. The interval W is called the **working set window size**, and the number of distinct references in the interval is called the **working set size**. High temporal locality is reflected by a small working set size.

Figure 2 shows the average working set sizes for several different window sizes. The data shows that the destination reference pattern has a high temporal locality. For example, 65 distinct destinations were referenced on the average in successive working set windows of 500 references. In the absence of temporal locality, this number should have been close to 500.

Also, notice that the temporal locality does not exist for small working set window sizes (of up to 50). For example, the average working set size for a window of 10 references is 9.

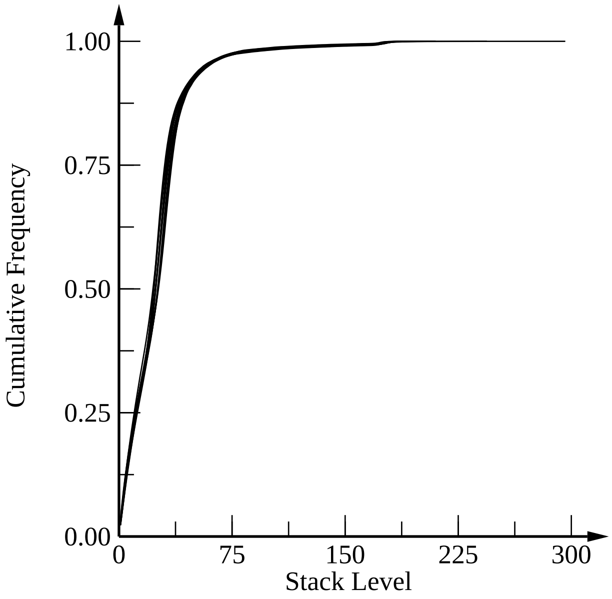


Figure 3: Stack distance cumulative probability distribution function.

6 LRU Stack Model

The LRU stack model assumes that the probability of reference to an address is a decreasing function of time since it was last referenced. If the addresses are arranged in a stack so that the address referenced is always taken out of its current position in the stack and pushed to the top of the stack, the probability p_i of i^{th} stack position (counting from the top toward the bottom of the stack) being referenced is a decreasing function of i. For a reference string with a high temporal locality, the probability p_1 of the stack top being referenced again would be high. This model has been analyzed extensively in literature beginning with [22].

The cumulative frequency of reference up to several different stack levels is shown in Figure 3. Notice that:

1. The stack top (level 1) reference frequency is only 2% to 3%. This is different from the data measured at M.I.T. [4,9] where 30% of the references were found at the stack top and the top two levels had a cumulative reference frequency of 60%.
2. We see that the top 100 stack positions (20% of the total possible stack positions) account for

98% of the frames. This is much lower than corresponding figures seen for page reference and file reference strings [2].

The first observation above is further substantiated by a study of consecutive references. Table 4 shows the observed frequency of a destination being referenced in n successive frames for various values of n. Notice that the frequencies are rather small.

Table 4: Frequency of Consecutive References

Sub-	Number of Consecutive References				
trace	1	2	3	4	Longest
1	0.946	0.024	0.001	0.000	10
2	0.948	0.023	0.001	0.000	8
3	0.955	0.021	0.001	0.000	8
4	0.940	0.026	0.002	0.001	9
5	0.947	0.023	0.001	0.000	14
6	0.955	0.021	0.001	0.000	10
7	0.948	0.023	0.001	0.000	9
8	0.947	0.022	0.002	0.000	8
9	0.936	0.025	0.003	0.000	9
10	0.946	0.024	0.002	0.000	9
11	0.957	0.020	0.001	0.000	5
Total	0.947	0.023	0.001	0.000	14

7 Cache Replacement Algorithms

More important than the theoretical question of which locality model applies best to the destination references is the practical question of which replacement algorithm is best for caching such addresses. To answer this latter question, we compared different cache replacement algorithms. The traditional metric for performance of a cache is the number of *faults* or *misses*. A fault or miss is said to occur when an address is not found in the cache. On a cache miss, one of the entries in the cache must be replaced to bring in the missed entry. Several replacement algorithms can be found in the literature on processor design and virtual memory. We chose four popular algorithms for comparison: least recently used (LRU), first in first out (FIFO), random (RAND), and a theoretically optimal algorithm called MIN [1]. Given a reference trace and a fixed-size cache, it has been proven that the MIN algorithm would cause less faults than any other algorithm. MIN chooses the address that will be referenced farthest in future. It, therefore, requires looking ahead in the reference string. Obviously, it cannot be implemented in a real system. Nonetheless, it provides a measure of how far a particular algorithm is from the theoretical optimal.

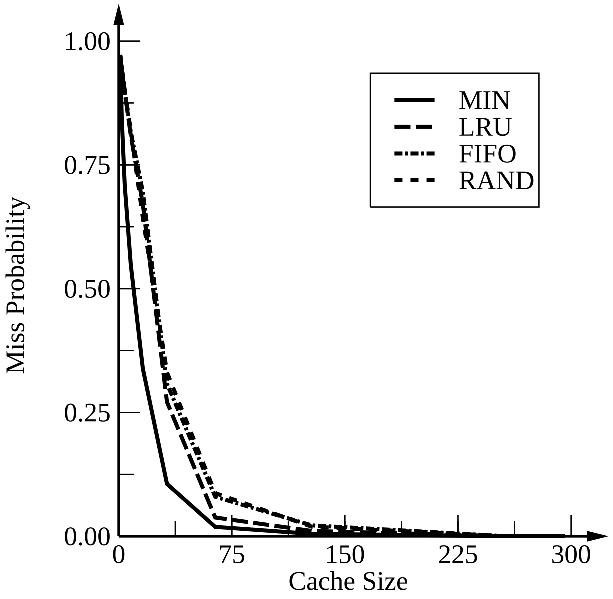


Figure 4: Cache miss probability for various cache replacement algorithms.

We used the following three metrics to compare the replacement algorithms:

1. Miss probability
2. Interfault distance
3. Normalized search time

We have defined these metrics and the results are presented in the following subsections.

7.1 Miss Probability

The miss probability is defined as the probability of not finding an address in the cache. For a given trace, it is simply the ratio of the number of faults to the total number of references in the trace. The lower the miss probability, the better the replacement algorithm.

The miss probabilities for various cache sizes for the four replacement algorithms are presented in Figure 4. From the figure we see that for small caches, LRU,

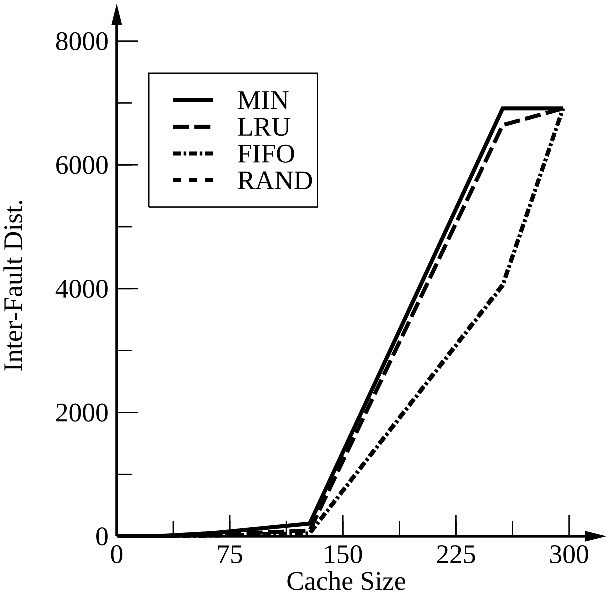


Figure 5: Interfault distances for various cache replacement algorithms.

FIFO, and RAND are not very different for this trace. The miss probability for MIN is better by approximately a factor of two. Thus, there is sufficient room for improvement by designing another replacement algorithm.

For large cache sizes, the miss probability curves are too close to make any inferences. The interfault distance curves provide better discrimination for such sizes.

7.2 Interfault Distance

The interfault distance is defined as the number of references between successive cache misses. For a given trace, the average interfault distance can be computed by dividing the total number of references by the number of faults. Thus, average interfault distance is the reciprocal of the miss probability.

Average interfault distances for our trace using the four replacement algorithms are shown in Figure 5.

From the figure we see that for large caches, LRU is close to optimal. FIFO and RAND are equally bad for this trace. Thus, unless one discovers a better replacement algorithm, we can use large caches with the LRU replacement algorithm.

This leads us to wonder what is the optimal cache size. If a cache is too small, we have a high miss rate. If the cache is too large, we do not gain much even if the miss rate is small since we have to search through a large table. The question of optimal cache size is answered by our third metric, normalized search time, discussed below.

7.3 Normalized Search Time

Caches are useful for several reasons. First, they may have a faster access time then the main database. This is particularly true if the main database is remotely located and the cache is local. Second, they may have a faster access method. For example, caches may be implemented using associated memories (CAMs). Third, the references have a locality property so that entries in the cache are more likely to be referenced than other entries.

We need to separate the effect of locality and find out if there is sufficient locality in the address reference patterns to warrant the use of caches. If there is enough locality, one would want to use a cache even if the access time to cache was same as that of the main database, and if the cache used the same access method (say binary search) that would be used for the main database.

Assuming that the access time and the access method for the cache are the same, we can compute the average access time with and without cache and use the ratio of the two as the metric of contribution to performance due to locality alone.

Assuming that a full database of n entries would generally require a search time proportional to $1 + \log_2(n)$, we have:

$$\text{Time to search without cache} = 1 + \log_2(n)$$

With a cache, if p is the miss probability, we need to search through both the cache and and the full table with probability p, and the normalized search time is defined as the ratio:

$$\begin{aligned}\text{Normalized Search Time} &= \frac{\text{Search time with cache}}{\text{Search time without cache}}\\ &= \frac{(1-p)[1+\log_2(c)] + p[1+\log_2(c)+1+\log_2(n)]}{1+\log_2(n)}\end{aligned}$$

The normalized search time for the four replacement algorithms considered is shown in Figure 6. From the figure, we see that with a cache using the MIN

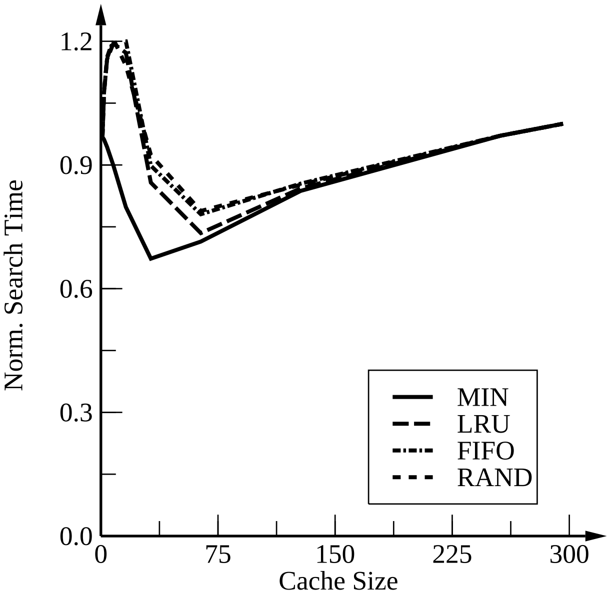


Figure 6: Normalized search time for various cache replacement algorithms.

replacement algorithm, we could achieve up to 33% less search time than that without caching. The payoff with other replacement algorithms is much less. It is more important to observe, however, that with LRU, FIFO, and RAND, the total search time may be *more* with a small caches than that without a cache. For example, with a cache size of 8, these three algorithms would require 20% more search time than without a cache. This trace, therefore, shows a reference pattern in which *caching can be harmful.*

With a very large cache, the cache does reduce the search time, but the gain decreases as the cache size increases. The optimal cache size for this trace is approximately 64, which produces 20 to 25% reduction in search time.

Earlier measurements at the Massachusetts Institute of Technology [4,9] on a token ring had shown that even a small cache sizewould provide a big payoff. Therefore, we need to understand what behavior in our environment leads to this different conclusion. We suspect several possibilities. First, the traffic level at M.I.T. is only one tenth of that in our environment. At M.I.T., the traffic level was two million frames per day while in our environment we have that much traffic in one hour. The M.I.T. ring uses an 8-bit address field leading to a maximum of 256 possible addresses on the ring. Actually, there are less than 40 stations on the ring. Our environment uses a 48-bit address field and there are 1200 stations on the extended LAN. M.I.T. frames are much shorter too. The maximum frame size seen on the ring is 576 octets (although the ring allows 2048-octet frames), while the maximum frame size on Ethernet is 1518 octets. A user message is broken into more successive frames resulting in higher persistence in the M.I.T. data. Increased traffic level, more stations, and larger packets could certainly make small caches less effective. However, looking at the stack distance probability density function provided another clue, which we discuss next.

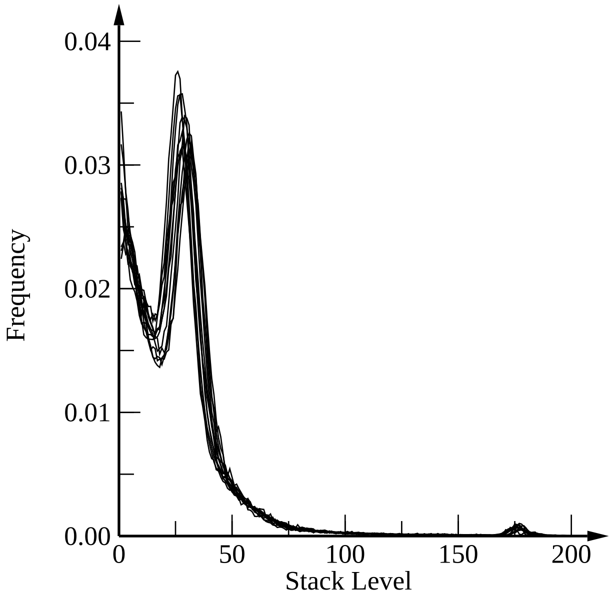
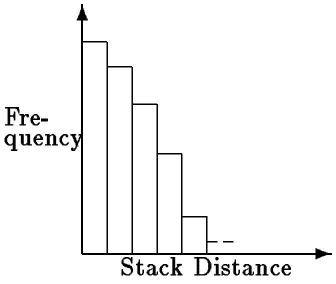


Figure 7: Stack reference frequency.

8 Stack Reference Frequency

Earlier in Section 6, we showed the cumulative probability distribution function using a stack model. If, instead of adding the probability for successive stack positions, we plot the probability for individual stack positions, we get the probability density function **(pdf)** curve as shown in Figure 7. In this figure, we have plotted the stack pdf for the complete trace as well as for the 11 subtraces. In all cases, we see that the pdf is not a continuously decreasing function. Instead, there is a *hump* around stack position 30. *For this environment, the most likely stack position to be referenced is the 30th position and not the stack top.*

(a) Ideal
$\{\ldots, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, \ldots\}$



(b) Round-robin
$\{\ldots, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, \ldots\}$

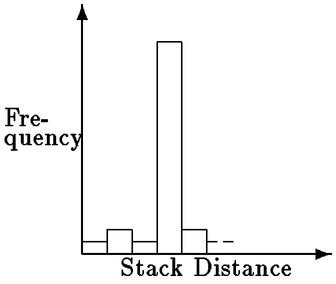


Figure 8: A round-robin reference pattern results in a hump in the stack reference frequency.

LRU is not the best replacement strategy for such a reference string. In general, it is better to replace the address least likely to be referenced again, i.e., the address with minimum probability. For the stack reference probabilities shown in Figure 7, the minimum probability does not always occur at the highest possible stack distance. For example, if the cache size is 30, the address at stack position 15 has a lower probability of reference than that at position 30 and is, therefore, a better candidate for replacement.

One possible cause of the hump could be a round-robin behavior in our reference pattern. To understand this consider two hypothetical reference patterns shown in Figure 8. The first pattern shows a high persistence. Once an address is referenced, it is referenced again several times. Such a reference string would result in a continuously decreasing stack pdf of the type shown in Figure 8a. The second pattern shows a round-robin reference string consisting of k addresses, for instance, repeated over and over again $\{1, 2, 3, ..., k, 1, 2, 3, ..., k, 1, ...\}$. The stack pdf for this string would be an impulse (or Dirac delta) function at k, that is, all references would be to stack position k.

A mixture of round-robin and persistent traffic would result in a curve with a hump similar to the one observed in Figure 7. This round-robin behavior could be caused by the periodic nature of some of the protocols used on our network. In particular, the interactive terminal traffic, which constitutes 77% of the frames in our trace, uses a protocol called the Local Area Terminals (LAT) [15]. Each LAT server is connected to a number of terminals and provides a virtual connection to several hosts on the extended LAN. To avoid sending several small frames, the terminal input is accumulated for 80 milliseconds and all traffic going to one host is sent as a single frame. This considerably reduces the number of frames and improves the performance of the terminal communication. A large number of LAT servers transmitting at regular intervals of 80 milliseconds could very well be responsible for the round-robin behavior observed in the reference pattern.

To verify the above hypothesis, we divided our trace into two subtraces: one consisting entirely of interactive (LAT) frames, and the other remaining noninteractive traffic. The stack pdf for these two subtraces are shown in Figures 9 and 10. Notice that the interactive traffic exhibits a hump, while the noninteractive traffic does not. Thus, the interactive traffic does seem to be responsible for the hump leading to the conclusion that, for environments dominated by LAT and similar protocols, one would need either a cache size equal to the number of LAT servers or to develop a cache prefetch policy that would bring the right address into the cache just before it is referenced.

The observation that the noninteractive traffic has a continuously decreasing stack pdf is an interesting one. Since the LAT traffic is limited to a single extended LAN, it does not go through routers, which are used to connect several extended LANs to wide area networks. The reference pattern seen at routers is expected to be similar to that of the noninteractive traffic, though we have not yet verified this observation. If this is so, it would be interesting to see if caching would pay off for noninteractive traffic alone. We, therefore, analyzed the noninteractive traffic in the next section.

9 Analysis of the Noninteractive Traffic

In this section, we present the graphs for miss probability, interfault distance, and normalized search time

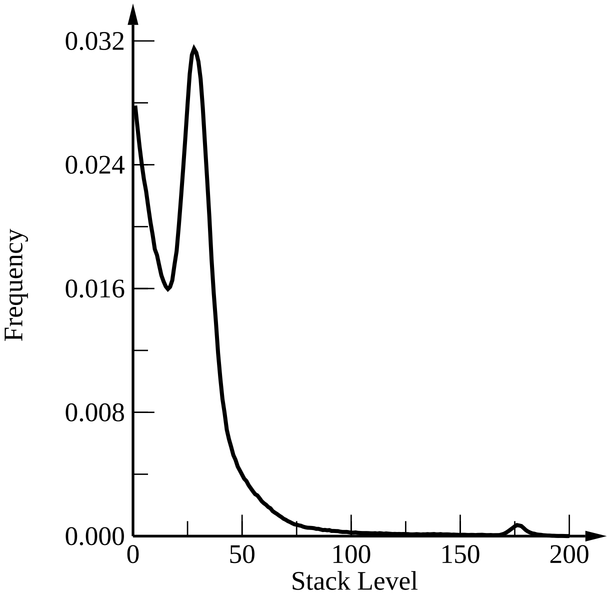


Figure 9: Stack distance density function for LAT traffic.

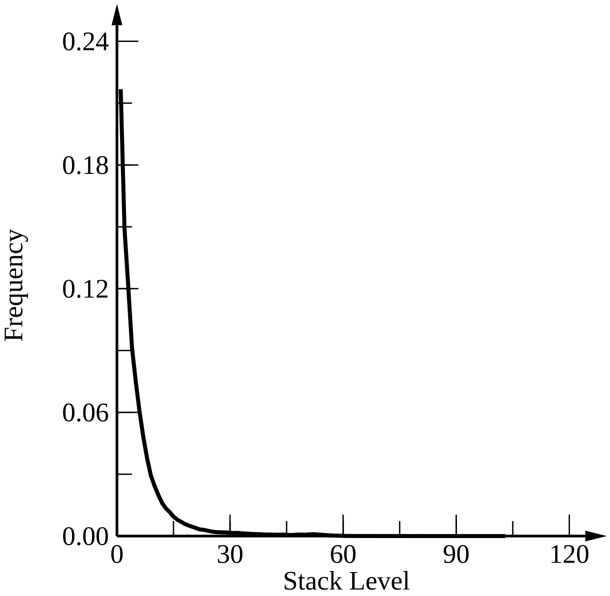


Figure 10: Stack distance density function for noninteractive traffic.

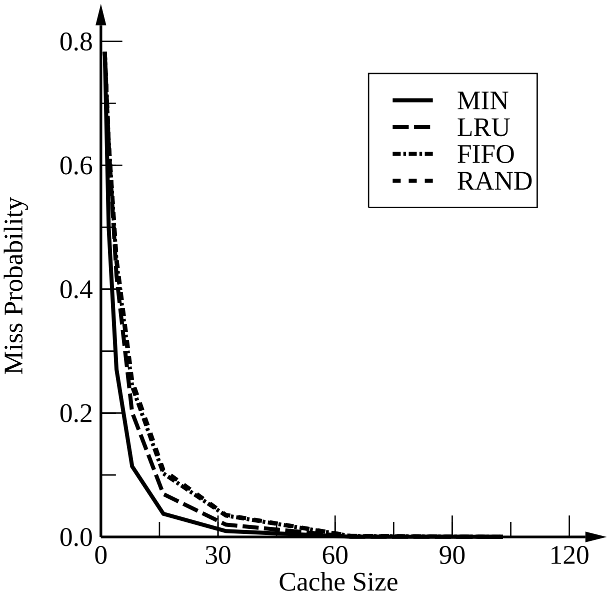


Figure 11: Cache miss probability for noninteractive frames.

for noninteractive traffic alone. There are two reasons for repeating the analysis for noninteractive traffic alone. First, as we said earlier, it may give us some indication of behavior of references in routers. Second, it helps us illustrate how some of the conclusions reached earlier would be different in a different environment.

Figure 11 shows the miss probability for the four replacement algorithms. Notice that even for small caches, LRU is significantly better than FIFO and RAND. This is not surprizing considering the fact that for any reference trace with nondecreasing stack pdf, LRU is the optimal cache replacement algorithm [26]. LRU is optimal in the sense that no other practical algorithm can give a lower number of faults for any given cache size. MIN does give a lower number of faults and, hence, a lower miss probability, but that is due to its knowledge of future references. For reference patterns similar to noninteractive traffic, therefore, we do not need to look for other replacement algorithms. Of course, if LRU is too complex to implement, which is often the case, one would go for simpler algorithms, but that would always come at a cost of increased faults.

Figure 12 shows the interfault distances for the four replacement algorithms. We see that for large cache sizes also, LRU is far superior to FIFO and RAND

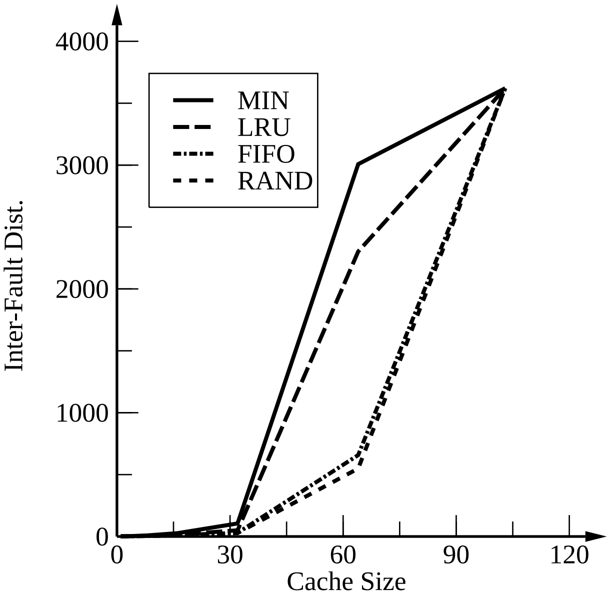


Figure 12: Interfault distances for noninteractive frames.

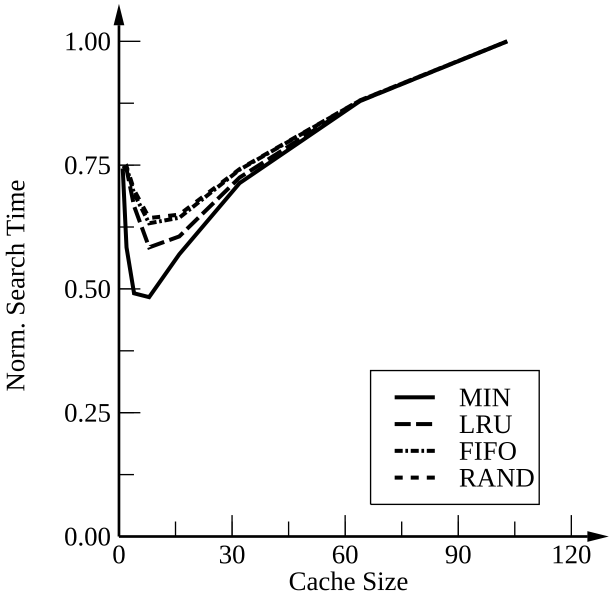


Figure 13: Normalized search time for noninteractive frames.

for this subtrace.

The normalized search time for noninteractive traffic is shown in Figure 13. Notice that for small caches, we now have a valley where we had a peak in Figure 6. Thus, not only are the small caches helpful they are also optimal. The optimal cache size with LRU is about 8 entries. This reduces the search time by about 40%.

10 Other Cache Design Issues

There are many cache design issues that remain to be addressed before caching of network addresses can become a reality. The issues can be classified as cache management, cache structuring, and multicache issues.

Cache management issues relate to algorithms for replacement, fetching, lookup, and deletion. Several replacement algorithms have been compared in this paper. We assumed demand fetching where the address is brought into the cache when it is actually referenced. Prefetching, such as that of source addresses, needs to be analyzed. Address matching strategies, such as the most significant octet first or the least significant octet first may produce different performances. Finally, the issue of deleting addresses periodically needs to be studied.

Processor caches are generally structured as sets [24]. Each set consists of several entries. A given address is first mapping to a set and the replacement, lookup etc is then confined to that set. Two extreme cache structures are: direct mapped in which each set consists of only one entry, and fully associative in which all entries are part of the same set and there is no mapping.

Another issue related to cache structuring is that of organizing separate caches for different types of addresses. For example, in many computer systems, instruction and data caches are organized separately since their reference patterns are so different [25]. In computer networks, one may want to study the effect of organizing separate caches for group and individual addresses, separate caches for interactive and noninteractive traffic, or a separate cache for each protocol type.

Multicache consistency [12] is also an interesting issue, particularly in multiport intermediate systems in which each port has a separate cache of addresses.

Finally, in many networks such as token ring systems, it is important for an intermediate system to immediately decide whether to set the 'address recognized' and 'frame copied' flags in the frame. In such a system, cache lookup time is bounded. It remains to be seen what impact this time bound has on cache management and structuring strategies.

11 SUMMARY

As sizes of computer networks grow, we need to find ways to efficiently and quickly recognize destination addresses. Caching is one one such alternative that helps if there is locality in the reference pattern. *Concentration* of references to a small fraction of addresses as well as the *persistence* of the references to recently used addresses help achieve a low miss probability even with small caches.

We reviewed the concepts of spatial and temporal locality along with well-known models such as IRM, working set, and LRU and tried to apply them to destination reference strings.

We compared four different cache replacement algorithms: MIN, FIFO, LRU, and random and discovered that although address traces do have both concentration and persistence, the periodic nature of certain protocols may make the use of small caches ineffective. For those environments where a similar round-robin reference pattern is observed, either we need to develop new cache replacement and fetch algorithms, or to use larger caches.

Some of the observations presented in this paper are limited to our environment and application (bridge caching). However, the methodology is general and can be applied to other environments and problems as well. In particular, it would be interesting to apply it to the study of the reference pattern of the 20-octet addresses used in ISO network layers and the name reference patterns in various name servers and distributed systems.

12 Acknowledgments

We would like to thank Ruei-Hsin Hsiao of DEC for helping to gather the trace data. Shawn Routhier, currently of Prime Computers, Inc., also helped in collecting a trace when he was with M.I.T. Although the M.I.T. trace results have not been presented here, the data did help us in getting started on this project. Thanks are also owed to Bill Hawe, Tony Lauck, and other members of Digital's Networking Architecture group for their valuable feedback during the project.

References

[1] L. A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer," IBM Systems Journal, Vol. 5, No. 2, 1966, pp. 78-101.

[2] R. B. Bunt and J. M. Murphy, "The Measurement of Locality and the Behavior of Programs," The Computer Journal, Vol. 27, No. 3, 1984, pp. 238-245.

[3] P. J. Denning, "The Working Set Model for Program Behavior," Comm. of ACM, Vol. 11, No. 5, May 1968, pp 323-33.

[4] D. C. Feldmeier, "Improving Gateway Performance with a Routing-Table Cache," Proc. IEEE INFOCOM '88, March 1988.

[5] R. Floyd, "Short-Term File Reference Patterns in a UNIX Environment," University of Rochester, Comp. Sc. Dept., TR-177, March 1986.

[6] R. Floyd, "Directory Reference Patterns in a UNIX Environment," University of Rochester, Comp. Sc. Dept., TR-179, August 1986.

[7] V. K. Garg and C. V. Ramamoorthy, "Effect of Locality in Large Networks," Proc. IEEE 7th Internat. Conf. on Distributed Computing Systems, Berlin, West Germany, September 1987, pp. 544-550.

[8] W. Hawe, A. Kirby, and B. Stewart, "Transparent Interconnection of Local Networks with Bridges," Journal of Telecommunications Networks, Vol. 2, No. 2, September 1984, pp. 117-130.

[9] R. Jain and S. Routhier, "Packet Trains: Measurements and New Model for Computer Network Traffic," IEEE Journal on Special Areas in Communications, Vol. SAC-4, No. 6, September 1986, pp. 986-994.

[10] R. Jain, "A Comparison of Hashing Schemes for Address Lookup in Computer Networks," DEC Technical Report, DEC-TR-593, January 1989, (Available from the author).

[11] H. Kanakia and D. R. Cheriton, "The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors," Proc. SIGCOMM'88, Stanford, CA, August 1988, pp. 175-197.

[12] A. R. Karlin et al, "Competitive Snoopy Caching," Carnegie-Mellon University, Report CMU-CS-86-164, 1986.

[13] C. A. Kent, "Cache Coherence in Distributed Systems," DEC Western Research Lab, Report 87/4, December 1987.

[14] O. Kure, "Optimization of File Migration in Distributed Systems," University of California, Report UCB/CSD 88/413.

[15] B. Mann, C. Stutt, and M. Kempf, "Terminal Servers on Ethernet Local Area Networks," Digital Technical Journal, No. 3, September 1986, pp. 73-87.

[16] R. L. Mattson et al, "Evaluation Techniques for Storage Hierarchies," IBM Systems Journal, Vol. 9, No. 2, 1970, pp. 78-117.

[17] K. S. Natarajan, "Performance Analysis of Prefetch Strategies for Database Access," Proc. Intl. Conf. on Modeling Techniques and Tools for Performance Analysis, Sophia Antipolis, France, June 1985, North Holland, New York, pp. 207-223.

[18] J. S. Quarterman and J. C. Hoskins, "Notable Computer Networks," Communications of the ACM, Vol. 29, No. 10, October 1986, pp. 932-971.

[19] G. S. Rao, "Performance Analysis of Cache Memories," JACM, Vol. 25, No. 3, July 1978, pp. 378-395.

[20] M. Satyanarayanan and D. Bhandarkar, "Design Trade-Offs in VAX-11 Translation Buffer Organization," IEEE Computer, December 1981, pp. 103-111.

[21] M. D. Schroeder, D. K. Gifford, and R. M. Needham, "A Caching File System for a Programmer's Workstation," Operating Systems Review (ACM), December 1985.

[22] J. E. Shemer and G. A. Shippey, "Statistical Analysis of Paged and Segmented Computer Systems," IEEE Trans. Electronic Computers, Vol. EC-15, 1966, pp. 855-863.

[23] A. J. Smith, "Optimization of I/O Systems by Cache Disks and File Migration: A Summary," Performance Evaluation, Vol. 1, 1981, pp. 249-262.

[24] A. J. Smith, "Cache Memories," Computing Surveys, Vol. 14, No. 8, September 1982, pp. 473-530.

[25] A. J. Smith, "Problems, Directions and Issues in Memory Hierarchies," Proc. 18th Annual Hawaii Intl. Conf. on System Sciences, 1985, pp. 468-476.

[26] J. R. Spirn, *Program Behavior: Models and Measurements*, New York: Elsevier, 1977.

[27] W. D. Strecker, "Cache Memories for PDP-11 Family Computers," Proc. 8th Symposium on Computer Architecture, 1976, pp. 155-158.

[28] J. G. Thompson, "Efficient Analysis of Caching Systems," University of California, Berkeley, Report No. UCB/CSD 87/374.

[29] C. L. Williamson and R. B. Bunt, "Characterizing Short-Term File Referencing Behavior," Proc. 5th Annual International Phoenix Conference on Computers and Communications (PCCC'86), Scottsdale, Arizona, March 1986, pp. 651-660.

13 Appendix: Numerical Results

In this paper, we have presented results graphically wherever possible. To allow easy reading of the values plotted, the same results are now presented in tabular form in this appendix.

Table 5: Average Working Set Size

Sub-	Window Size						
trace	5	10	20	50	100	200	500
1	4.7	8.9	16.2	29.3	37.6	47.4	62.7
2	4.7	9.0	16.3	29.7	38.1	48.0	63.4
3	4.8	9.0	16.2	30.3	38.8	48.6	64.1
4	4.7	8.9	16.0	29.5	38.3	48.5	64.4
5	4.7	9.0	16.3	30.4	39.3	49.5	65.4
6	4.8	9.0	16.4	31.2	40.1	49.8	65.1
7	4.8	9.0	16.5	30.4	38.7	48.4	63.7
8	4.7	8.9	16.2	31.2	40.2	49.8	64.9
9	4.7	8.8	16.1	31.6	40.7	50.0	64.6
10	4.7	8.9	16.2	31.5	40.5	50.1	65.1
11	4.8	9.0	16.3	31.5	40.9	50.6	65.4
Total	4.7	9.0	16.3	30.5	39.3	49.0	64.4

Table 6: Cumulative Frequency for Different Stack Levels

Sub-	Stack Distance						
trace	1	2	5	10	20	50	100
1	0.03	0.06	0.13	0.23	0.41	0.93	0.98
2	0.03	0.05	0.12	0.22	0.41	0.93	0.98
3	0.02	0.05	0.12	0.23	0.41	0.93	0.98
4	0.03	0.06	0.14	0.25	0.44	0.93	0.98
5	0.03	0.05	0.12	0.22	0.40	0.92	0.98
6	0.02	0.05	0.11	0.21	0.38	0.92	0.99
7	0.03	0.05	0.12	0.21	0.38	0.93	0.98
8	0.03	0.05	0.13	0.23	0.39	0.92	0.99
9	0.03	0.07	0.14	0.24	0.39	0.93	0.99
10	0.03	0.06	0.13	0.23	0.38	0.92	0.99
11	0.02	0.05	0.12	0.22	0.39	0.92	0.99
Total	0.03	0.05	0.13	0.23	0.40	0.93	0.98

Table 7: Miss Probability

Cache Size	MIN	LRU	FIFO	RAND
1	0.972	0.972	0.972	0.972
2	0.846	0.946	0.946	0.947
4	0.708	0.896	0.898	0.901
8	0.548	0.810	0.816	0.819
16	0.339	0.670	0.695	0.643
32	0.106	0.271	0.308	0.331
64	0.019	0.038	0.080	0.087
128	0.005	0.011	0.022	0.019
256	0.000	0.000	0.000	0.000
296	0.000	0.000	0.000	0.000

Table 8: Average Interfault Distance

Cache Size	MIN	LRU	FIFO	RAND
1	1.0	1.0	1.0	1.0
2	1.2	1.1	1.1	1.1
4	1.4	1.1	1.1	1.1
8	1.8	1.2	1.2	1.2
16	3.0	1.5	1.4	1.6
32	9.5	3.7	3.2	3.0
64	52.8	26.4	12.5	11.5
128	205.3	92.8	45.4	51.6
256	6912.2	6642.9	4051.5	4067.6
296	6912.2	6912.2	6912.2	6912.2

Table 9: Normalized Search Time

Cache Size	MIN	LRU	FIFO	RAND
1	0.968	0.968	0.968	0.968
2	0.962	1.076	1.076	1.077
4	0.943	1.157	1.159	1.162
8	0.899	1.195	1.202	1.205
16	0.798	1.172	1.200	1.141
32	0.673	0.857	0.898	0.924
64	0.714	0.735	0.781	0.788
128	0.837	0.843	0.855	0.852
256	0.971	0.971	0.971	0.971
296	1.000	1.000	1.000	1.000