IMPROVING GATEWAY PERFORMANCE WITH A ROUTING-TABLE CACHE

David C. Feldmeier
Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Ma. 02139

Abstract

A way to increase gateway throughput is to reduce the routing-table lookup time per packet. A routing-table cache can be used to reduce the average lookup time per packet and the purpose of this paper is to determine the best management policies for this cache as well as its measured performance. The performance results of simulated caches for a gateway at MIT are presented. These results include the probability of reference versus previous access time, cache hit ratios, and the number of packets between cache misses. A simple, conservative analysis using the presented measurements shows that current gateway routing-table lookup time could be reduced by up to 65%.

1. Introduction

A gateway is a device that connects two or more heterogeneous networks and is responsible for transporting packets among them, assuring that packets arriving from one network are retransmitted toward their ultimate destination. Any gateway that is not directly connected to the destination must send the packet to the next gateway in the chain from source to destination; the address of this next gateway is the next-hop address. The gateway examines the destination address of a received packet and determines the next hop address by using the destination address as an index to a *routing table*. A routing table is a table of destination/next-hop address pairs that is maintained by the gateway. A routing-table lookup converts an internet destination address into a local network address. If the packet destination is on the same network as the source, then the next-hop is the local network address of the destination. Otherwise, the next-hop is the local network address of the next gateway that brings the packet closer to its final destination. The gateways used in this study forward TCP/IP packets.

The routing-table lookup is a time-consuming process of packet forwarding. Routing-table caching of destination/next-hop address pairs will decrease the average processing time per packet if locality exists for packet addresses. Therefore, gateway throughput can be increased if internet address locality exists and earlier measurements suggest that this is likely. The purpose of this research is to determine whether internet address caching in gateways is effective and what type of cache performs best. This research simulates the performance of several routing-table caches in a busy gateway, using recorded gateway traffic to drive the simulation. The following sections include a discussion of the cache simulation models, the collection of data that drive the cache simulation, the simulation results and analysis, a cache performance analysis, and conclusions.

2. Data Generation

The method used for generating data to determine the effectiveness of routing-table caching is *trace-driven simulation*. A trace is gathered for an operating gateway by recording every routing table reference during normal gateway operation. This trace is then used to drive a cache simulation model. The cache model can be altered to simulate a cache of any type and size, and the simulation can be repeated on the same data set.

2.1. Cache Modeling

2.1.1. Cache Location

Determining the next-hop address of a packet based on its destination address is more complex than a simple routing-table access on current gateways. Currently, routing is based on a hierarchical addressing scheme, which means that some part of the destination address contains information about what network the destination host is on. The routing table contains information only about the next hop to reach a given network, rather than a given destination. Before the routing table is used, the network address for the destination must be extracted from the destination address. The decoding of the destination address into a network address and a host address is non-trivial because the network address may be encoded in the destination address in many different ways.

Two possible positions for a cache are before the address decoder or between the address decoder and the routing table. The advantage of placing a cache before the address decoder is that destination addresses found in the cache need never be processed by the decoder. The disadvantage is that for a hierarchical addressing scheme, the number of addresses exceeds the number of networks,

0298 CH2534-6/88/0000-0298 $1.00 © 1988 IEEE

so for a cache of fixed size, the cache performance for the full address must be no better than the performance of a cache for the network part of the address. The choice of cache location is determined by the relative cache performance for flat and hierarchical addresses and by the time required for address decoding. In fact, some simple pre-processing of the destination address in such a way that does not affect address decoding may increase the performance of the cache placed before the address decoder at relatively little cost. The cache/pre-processor combination before the address decoder may offer the best overall performance.

2.1.2. Cache Simulation

A routing-table cache differs from a main memory cache in several ways. A main-memory cache is bidirectional - it must be able to handle both read and write commands from the processor. A routing-table cache is unidirectional, because the packet-forwarding process only reads from the table. Read-only operation eliminates the need for routing-table update from the cache and simplifies cache operation. Another difference is that on a network with dynamic routing, the routing table changes over time. If the routing table changes, those entries in the cache may become inconsistent with the routing table. The cache model assumes that the cache is never flushed, which approximates true cache operation if the routing table is quasi-static or there exists an inexpensive method of updating the cache. Current gateway routing tables are relatively static, so never flushing the cache approximates cache performance on current gateways.

Different types of caches will behave differently with the same data set, so a type of cache to simulate must be chosen. A routing-table cache is defined by its associativity, replacement, and prefetch strategies, as well as size. Each of these must be defined for the simulation model.

The associativity of a cache ranges from fully associative to direct mapped. Fully associative caches allow any destination/next-hop pair to be stored in any cache location. A direct-mapped cache allows each destination/next-hop pair to be stored in only a certain cache location, so multiple destination addresses are mapped into a single cache location. In general, the cache is divided into a disjoint set of locations that may contain the contents of a certain subset of destination/next-hop pairs, and this is a set-associative cache. For these measurements, a fully associative cache is used because it provides the best performance for a given number of cache slots. Also, the performance of any set-associative cache depends on the value of those items to be stored in the cache and the cache association mechanism. A fully-associative cache avoids the problem of measurement bias because of specific network addresses in the measurements and makes the results general for any set of network addresses with similar characteristics. In addition, broad associative searches are easily implemented in MOS VLSI, so it is reasonable to expect that highly associative caches will be available on a chip[1].

Whenever there is a cache miss, the missed entry must be entered into the cache and some other item must be discarded. The three most popular strategies are random replacement, first-in first-out (FIFO), and least recent use (LRU). A random replacement strategy simply discards a random location in the cache. The "random" location is most easily chosen based on some counter value (such as a clock) from elsewhere in the system. A FIFO replacement strategy discards the cache entry that is the oldest; a FIFO cache is easily implemented as a circular buffer. The most complex strategy, but also the one that generally provides the best cache performance, is LRU. An LRU cache operates by storing not only the data, but also the time that each datum was last referenced. If a reference is in the cache, its time is updated; otherwise the data that was least recently accessed is discarded and the data for the new reference is cached.

An LRU cache is a good choice for the type of address locality expected and a fully-associative, LRU cache forms an upper-bound on the cache hit ratio for a fixed number of cache slots. Unfortunately, this type of cache is also the most complex and expensive to implement. A different type of cache may perform worse for a given number of slots, but for a fixed price, a less efficient cache can afford to have more slots and perhaps exceed the performance of a smaller fully-associative LRU cache; this is a cost trade-off that only the gateway designer can assess.

A strategy for prefetching is suggested by observing that a packet from host A to host B is often followed by a packet from B to A[2]. If the cache prefetches the next-hop for the packet source, then a packet traveling through the same gateway in the reverse direction no longer causes a cache miss. For the cache simulations, two situations are analyzed. The first is that only the packet destination is used to update the cache and this corresponds to a case where prefetching is not economical. The second assumes that prefetching is free and should be done any time that the packet source is not in the cache. In reality, the truth is somewhere in-between, so these two sets of curves give lower and upper bounds for fully associative LRU caches where prefetching occurs.

2.2. Trace Generation

We would like to estimate the performance of a gateway with a routing-table cache before the system is built. The performance can be determined by trace-driven simulation, which requires a list of all routing-table accesses on a gateway. We are interested in statistics of packet arrivals at various gateways, but building a measurement system directly into a gateway has several disadvantages. The main problem is that each gateway would need to be reprogrammed to include a monitoring system. Reprogramming is difficult because some of the gateways are owned by other research groups, which are reluctant to disturb gateway operation; other gateways are commercial products, in which case the source code is unavailable. In addition, any monitoring system installed in a gateway will use processor time and memory space and this overhead could affect gateway operation during periods of heavy

load, causing packets to be dropped. Since periods of heavy load are of particular interest to determine the efficiency of the caching system, the measurement artifact of such a monitoring program is unacceptable. Also, it is advantageous to be able to try various caching strategies and cache sizes on a single measurement set, so that differences in results are directly attributable to the change in strategy, rather than simply a change in the gateway traffic. Instead of monitoring the packet arrivals at each gateway, it is simpler to measure all of the packets on the network that pass through the gateway. This replaces monitoring programs in each gateway with a single dedicated monitoring system.

Trace-driven simulation computes the exact cache performance as long as the trace-gathering operation is exact and has no processing cost. Measuring packets on the network with a separate measurement system assures that the gateway itself operates as usual. Two types of packets are observed traveling to a gateway: packets to be forwarded to another network or packets destined to the gateway itself. Not all packets addressed to the gateway itself are observed, since only one of the networks attached to the gateway is monitored, but this is a problem only if a packet to a gateway causes a routing table reference. Whether a packet to a gateway causes a routing table reference depends on the gateway implementation. One possibility is to explicitly check each incoming packet against a table of all of the gateway's addresses to see if there is a match. This explicit check means that the routing table is never consulted about packets destined to the gateway. Another possibility is to use the routing table for all packets. The gateway will discover that it is directly connected to the proper network and will try to send the packet. Before the packet is sent, the gateway checks if the packet is to its own address on the appropriate network. If the packet is for the gateway, then it is never transmitted. The explicit check after the routing-table lookup requires checking a smaller number of gateway addresses at the increased cost of a routing-table lookup. For this paper, it is assumed that gateways do an explicit address check and that the unobserved packets do not cause routing-table accesses.

Another disadvantage of measuring all packets is that the packets on the network to (or from) a gateway may not be the same as those to be handled by the gateway software. If the current gateway drops a packet, some protocols will retransmit that packet. Thus our packet trace now has multiple copies of a single packet, only one of which is actually forwarded by the gateway. Some of these extraneous packets can be eliminated by filtering the packet trace to remove identical packets with certain interarrival times. In any case, these duplicate packets should only appear when the gateway is overloaded, and this is relatively rare.

The measurement system on the token ring is a general measurement system for monitoring all traffic and was originally built for the research done by Feldmeier[3]. The monitoring system consists of two computers - a passive monitor and a data analysis machine. The passive monitor receives the packets on the ring and timestamps them upon arrival. The passive monitor then compresses

The measurement system on the token ring is a general measurement system for monitoring all traffic and was originally built for the research done by Feldmeier[3]. The monitoring system consists of two computers - a passive monitor and a data analysis machine. The passive monitor receives the packets on the ring and timestamps them upon arrival. The passive monitor then compresses the information of interest from each packet into a large data packet that is sent to the analysis machine. The monitoring system generates less than 2% of the network traffic, so the monitoring overhead is acceptable; also, none of the packets pass through a gateway, so the arrival order of packets at a gateway remains unchanged. A more complete description of the design of the network monitoring station may be found in a paper by Feldmeier [4]. For these measurements, the monitor compresses information only for packets that are to or from non-local hosts; in other words, all the packets that pass through the gateways. The analysis machine simply stores all of the received data for later analysis.

Only a single trace using complete internet addresses for all packets through the gateway is needed for both flat and hierarchical addressing scheme cache simulation. Flat addressing measurements use the entire internet address for the cache simulation; hierarchical addressing uses only the network field of the internet addresses for the cache simulation.

2.3. Data Analysis

The measurements used in this paper are from the MIT ARPANET gateway. This is the main ARPANET connection to MIT and it serves the entire campus and several local companies involved in research as well. The ARPANET gateway processed 2,147,956 packets during the 24 hour measurement period. About 91% of all packets passing through the ARPANET gateway are part of virtual circuit connections (TCP protocol). The other interface of the ARPANET gateway is attached to a 10 Mbps token ring at the MIT LCS. In addition to 8 gateways, 20 computers also reside on the ring, including 10 VAXs, 7 PCs, a microVAX, an IBM-4341, and a PDP-11/45. The total number of packets processed by all gateways during a 24 hour period was 4,546,766. The ring itself carried 4,819,189 packets.

The results in this paper are based on network measurements made during the 24 hour period from 1:00 AM Monday, November 30th to 1:00 AM Tuesday, December 1st 1987. It is for this data that cache performance is evaluated; however any cache must start with no data and each cache reference that initializes the cache causes a cache miss. Since the reason for installing a cache in a gateway is to improve long-term average performance of the system, we are interested in the steady-state cache performance. To avoid cache misses caused by cache initialization, the cache is pre-loaded so that once the measurement period begins, the cache has many of its most referenced entries. Measurements during the 61 minute period from 11:59 PM Sunday, November 29th to 1:00 AM on Monday, November 30th 1987 were used to pre-load the cache to avoid measurements during cache initialization. During cache initialization, 230 distinct destination addresses and 254

distinct destination and source addresses arrived, out of 44,284 packets. Although a longer cache initialization would increase the cache performance, there is a limit to the amount of data storable by the network measurement system.

As mentioned above, the monitor cannot receive packets addressed to the gateway or broadcast packets from other networks, and it is assumed that these packets do not cause routing table accesses. Any packets of these types received on the monitored network are ignored in the data analysis.

As with any measurement, the data collected is not as precise as we would like. Because a machine could not be dedicated to the receiving of data from the network monitor, a time-sharing machine was used instead. The monitor transmits each packet to the analyzer once to minimize network loading, and if the receiving machine is occupied with other network functions at that time, the monitoring data are lost. During the course of monitoring, data were lost for 0.33% of packets passing through gateways on the ring.

Another source of measurement inaccuracies is caused by low-level retransmissions. As mentioned above, not all packets on the network are seen by the intended receiver. To assure more reliable packet transmission, the ring has a low-level acknowledgment system built into the network hardware; if an interface receives a packet, an acknowledgment bit in the packet trailer is marked. Since the transmitter must remove any packets that it transmits on the ring, the transmitter always can determine whether the packet it has sent was received by checking for the receiver's mark. The device drivers for hosts on the ring include a low-level retransmit mechanism that automatically retransmits a packet up to eight times without higher-level intervention if the receiver has not acknowledged the packet by marking it. This means that several closely-spaced identical packets may be transmitted, but only a single packet is received by the intended host. The monitoring station is fast enough to receive all of the packets present in a burst, but the receiver itself receives only one of them. The monitoring station cannot check the acknowledgment bit, so somehow these retransmissions must be removed from the data so that the time locality of the data will not be exaggerated. Since a low-level packet burst is at high speed, any packets with the same source and destination address that arrive within a short time of each other should be discarded; however, if this threshold is set too high, multiple packets between a host-destination pair may be incorrectly eliminated. The specific value that determines whether a packet is caused by a low-level retransmission depends on parameters of the network and the network hosts. Also, the threshold should exceed the transmission time of most packets. Most packets on the ring are 576 bytes or less long, and at 10 MBPS the transmission time is 461 microseconds.

The graph in figure 2-1 is a histogram of the interpacket arrival times for packets to all gateways with identical source and destination addresses, from 0 to 5 milliseconds with a clock granularity of 25 microseconds. There is a peak around 725 microseconds,

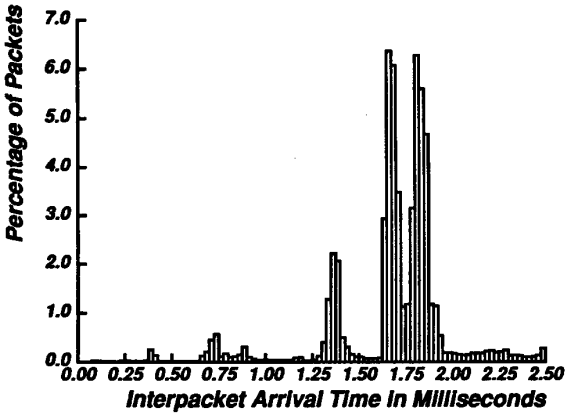

Figure 2-1: Interpacket Arrival Time of Identically Addressed Packets

and this is where most of the low-level retransmission must occur. The higher the low-level retransmission detection threshold is set, the more low-level retransmissions that are eliminated. However, the upper bound of this threshold is set by the interpacket time for the fastest host on the network. Since a gateway can transmit over 700 packets per second, the interpacket time of 1.35 milliseconds (the peak around 2% of all packets) is probably caused by a gateway transmission. For this data, any packets with the same source and destination that arrive within a millisecond or less of each other are discarded as low-level retransmissions.

3. Measurement Results

Although hierarchical addressing is relevant to existing gateways, the behavior of the cache under a flat addressing scheme is more easily understood because the performance of the cache can be understood in terms of various packet flows among hosts. It is also of interest to learn whether flat addressing can be done efficiently with a cache. Hierarchical addressing requires that many flat addresses are mapped to the same network address. Hierarchical addressing requires an understanding of a conglomerate of host flows seen as network flows.

3.1. Results for Flat Addresses

From a previous study, it is known that a packet from host A to host B has a high probability of being followed by another packet from A to B^2, which implies time locality of packet addresses. Consider a time-ordered list of previously seen addresses; for a current address that has been previously seen, how far down the list is this address? Figure 3-1 shows the percentage of references versus time-ordered position of previous reference for the 400 most recently referenced packet destination addresses. The graph is averaged over all packets received by the gateway and it is plotted on a logarithmic scale so that the relative popularity of the slots is more easily seen. The graph for destination/source caching is similar.

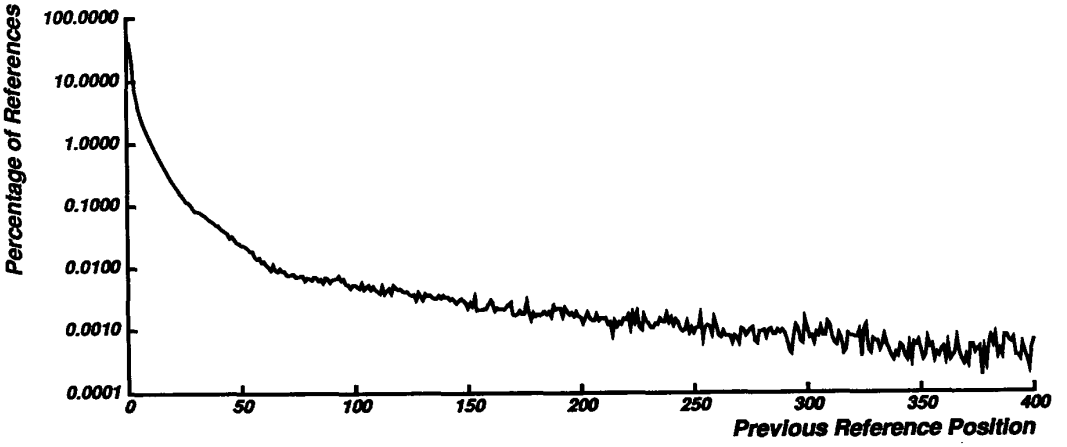

Figure 3-1: Percentage of Reference versus Time of Last Reference (Destination)

The decrease in probability for both graphs is nearly monotonic until about 50 most recently referenced addresses. This monotonically decreasing function implies that the LRU cache management strategy is optimal for caches of less than 50 slots. After reference 50, the probability continues to decrease generally, but there is more variance. One reason for the increased variance is that the probability of reference decreases with increasing age, so that the amount of data that falls into the farther positions is too small to properly approximate the probability distribution curve at these points. At these outer parts of the curve, the average number of packets per slot is about 15, as compared to between 500 and 876,000 in the first 50 positions. Another explanation is that the relationship between probability of reference and time of last reference changes at some point on the curve. This change in relationship might be caused by two separate mechanisms that determine whether the current packet address is in the cache.

The first mechanism that produces cache hits is a current packet that is part of a train of packets. The first packet of the train initializes the cache and as long as the cache is not too small, all following packets are cache hits. The second mechanism is that the first packet of a train is to a destination already in the cache. If N is the number of packets in a train*, the number of packets that arrive due to the first mechanism must be greater than N-1 times the number of packets that arrive due to the second mechanism**. This means that the destination addresses of packets in an active train are most likely to be near the first slot of an LRU cache and that addresses of inactive but popular train destinations are likely to be between slot *i*, where *i* is the number of simultaneously active trains, and the last slot of the cache. The relationship among train arrivals at a destination is unknown, but undoubtedly it is weaker than the relationship among packets in a train, and this could explain the higher variance in the later part of the probability distribution. Additional information is necessary to decide whether an insufficient number of samples or two cache hit mechanisms more correctly explains the increased variance in the later part of the probability distribution above.

If the probability distribution does change with age after a certain point, the most cost-effective strategy may be to have an LRU cache with 50 slots, followed by a FIFO or random management cache with a few hundred slots. If the probability of reference does not decrease monotonically with increasing last-reference age, then an LRU cache decreases in efficiency relative to other cache types and its higher cost may not be justified because other cache types allow a larger cache for the same price and perhaps a higher cache hit ratio for a given cost.

Given the data above, an LRU cache should be effective for reducing the number of routing-table references. The probability of reference versus time of previous reference data allows simple calculation of an LRU cache hit ratio for a given number of cache slots. The relationship between cache hit ratio and probability of access is:

$$f_h(i) = \sum_{j=1}^{i} p_j$$

Where $f_h(i)$ is the cache hit ratio as a function of *i*, the number of cache slots; and p_j is the probability of a packet address being the j^{th} previous reference. Figure 3-2 shows the percentage of cache hits versus cache size for both destination and destination/source caches. Notice that even a relatively small cache has a high cache hit ratio. Table 3-1 shows just how quickly the probability of a cache hit climbs even for small cache sizes. With as few as 9 slots, the hit ratio is already above 0.9.

*Jain and Routhier claim the average number of packets in a train is 17.4[2].

**For a cache large enough to hold entries for all simultaneous trains.

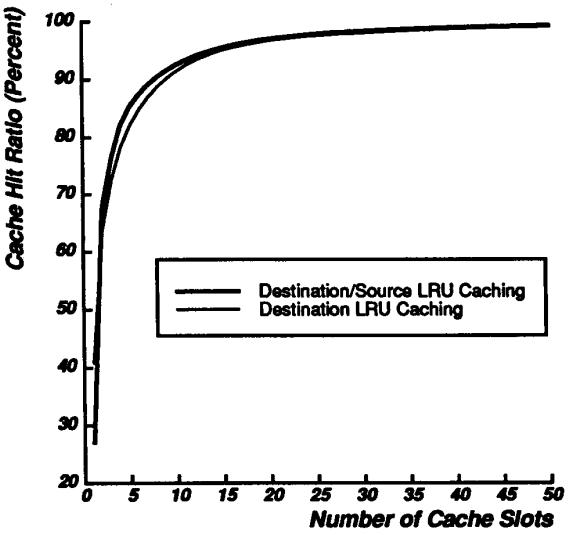

Figure 3-2: Percentage of Cache Hits versus Cache Size

Table 3-1: Cache Hit Ratio Percentage versus Cache Size

Number of Cache Slots	Destination Only	Destination Source
1	40.78	27.05
2	63.53	67.83
3	72.34	76.30
4	78.19	82.12
5	81.96	85.34
6	84.78	87.44
7	86.95	89.15
8	88.72	90.48
9	90.17	91.57
10	91.37	92.52
11	92.38	93.30
12	93.22	93.97
13	93.92	94.54
14	94.52	95.05
15	95.02	95.48
16	95.45	95.87
17	95.82	96.19
18	96.13	96.47
19	96.40	96.73
20	96.64	96.96

Another way of looking at the hit ratio data is to plot the average number of packets through the gateway between cache misses. The relation between cache hit ratio and packets between cache misses is:

$$f_m(i) = \frac{1}{1 - f_h(i)}$$

Where $f_m(i)$ is the number of packets between cache misses as a function of *i*, the number of cache slots; and $f_h(i)$ is the probability of a cache hit as a function of the number of cache slots.

The effectiveness of a cache depends on the ratio of the packets-between-misses values before and after the cache, not an absolute number, so the graph in figure 3-3 is plotted on a logarithmic scale so that equal ratios appear as equal vertical intervals. This graph shows the incremental efficiency of each additional slot in the cache. In addition to curves for destination LRU, destination/source LRU, destination FIFO, and destination/source FIFO, also shown are the destination and destination/source curves for a two cache system consisting of a 64 slot LRU cache followed by a FIFO cache.

As expected, the destination/source LRU cache performed best, followed closely by the destination LRU cache. The destination/source cache outperforms the destination only cache, which confirms that much of the traffic through the gateway is bidirectional. Performance of both FIFO caches is relatively poor until the destination FIFO cache size exceeds 1250. It turns out that 1035 is the number of distinct destination addresses handled by the gateway in a 24 hour period, in addition to 230 destinations in the preload, for a total of 1265. For destination/source caching, the FIFO buffer size does not help until 1370 slots, which is about the number of distinct destination and source addresses processed by the gateway in a 24 hour period, 1130, plus 254 during preload for a total of 1384.

The incremental efficiency of each slot in the cache is the slope of the curve at that slot number. A sharp decrease in the slope indicate a point of diminishing returns for larger cache sizes. Particularly notice the drop in LRU cache efficiency around cache slot 50.

The dual cache systems perform moderately well. For example, consider a 128 slot destination LRU cache (this can be thought of as two 64 slot LRU caches back to back). The same performance can be obtained with a 64 slot destination LRU cache followed by a 241 slot destination FIFO cache. A FIFO cache is simpler to implement than an LRU cache and if an 241 slot FIFO cache is cheaper than a 64 slot LRU cache, then the dual cache combination provides a better hit ratio for a given cost than a single LRU cache.

If the interest is in determining how many cache slots are necessary for a given number of packets between cache misses, the above graph is better plotted on a linear scale. The graph in figure 3-4 is interesting because each of the curves seems to be composed of two or three linear regions. At least for the LRU curves, the number of packet between misses seems proportional to the number of cache slots, until the curve approaches its maximum, where the incremental slot efficiency drops.

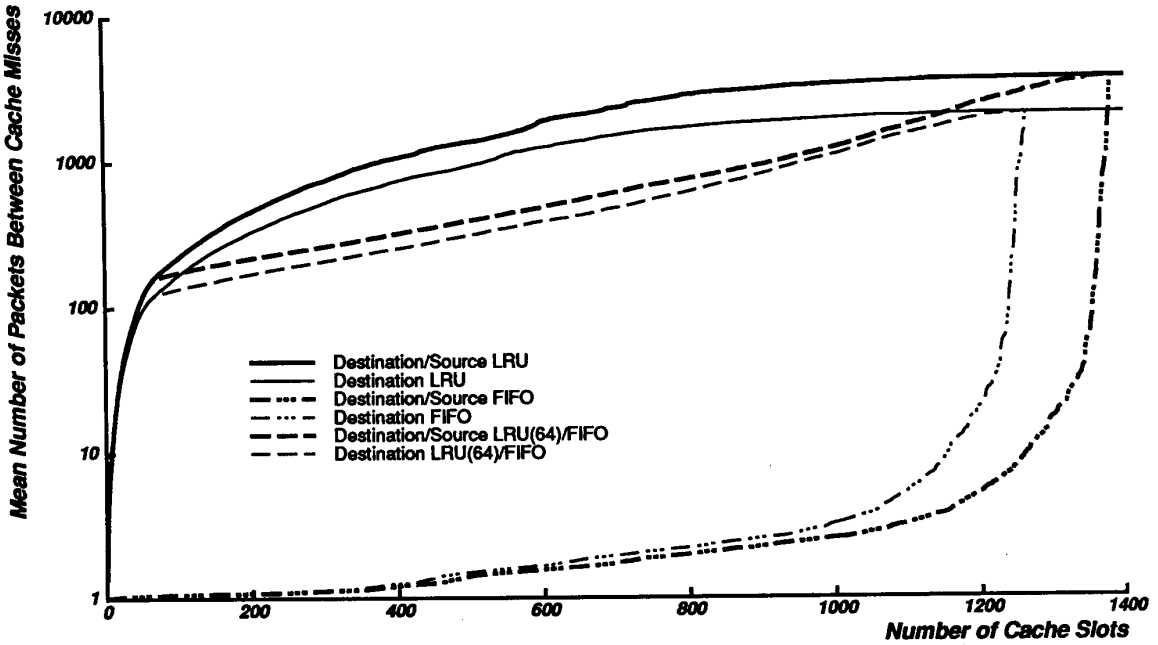

Figure 3-3: Number of Packets Between Cache Misses versus Cache Size (log-linear)

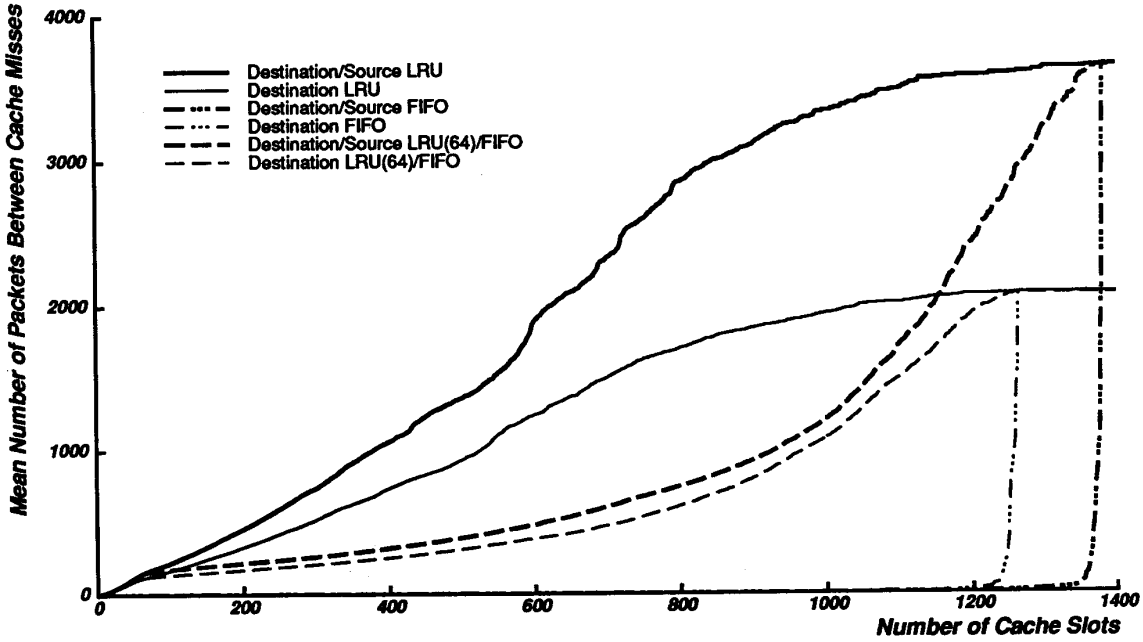

Figure 3-4: Number of Packets Between Cache Misses versus Cache Size (linear-linear)

3.2. Results for Hierarchical Addressing

Although in the previous section packet addresses were treated as flat addresses, in reality they are hierarchical addresses, which means that an address can be separated into network and host pieces. Currently, routing is done based on network addresses and since the number of networks is smaller than the number of hosts, the cache hit ratio for a network cache is higher than the hit ratio of a flat address cache. In order to cache network addresses, the complete address must pass through an

address decoder that extracts the network address. The tradeoff is that the hit ratio of the cache is higher than for flat addresses, but each address must be decoded.

Hierarchical addressing, as done by current gateways, is complex because the separation of the network and host fields of the internet address is determined by the type of address. In order to avoid this complication, the separation of fields for the purposes of this research is that the first three bytes of the full address refer to the network and that the last byte refers to the host. This procedure never removes part of the network address as long as the address is not a class C address with subnetting (which is not known to be used by anyone). In some cases, some of the host's address will be left with the network address, which means that the cache hit ratio will be lower than if the address were completely separated.

Since the purpose of a cache is to speed up routing table access, it is probably advantageous to place the cache before the address separation, rather than after it, so that each cache hit saves not only a table lookup, but also the trouble of dissecting the full address. It may be advantageous to use a simple address separation procedure before the cache, such as the suggested one of removing the last byte of the address. This preprocessing step increases the cache hit ratio by eliminating most, if not all, of the host address from the full address, and yet avoids the complexity of complete address decoding.

Figure 3-5 shows the percentage of references versus time-ordered position of previous reference for the 80 most recently referenced packet destination addresses. The graph is averaged over all packets received by the gateway and it is plotted on a logarithmic scale so that the relative popularity of the slots is more easily seen. The graph for destination/source caching is similar.

Figure 3-6 shows the percentage of cache hits versus cache size for both destination and destination/source caches. Notice that even a relatively small cache has a high cache hit ratio. Table 3-2 shows just how quickly the probability of a cache hit climbs even for small cache sizes. With as few as 7 slots, the hit ratio is already above 0.9.

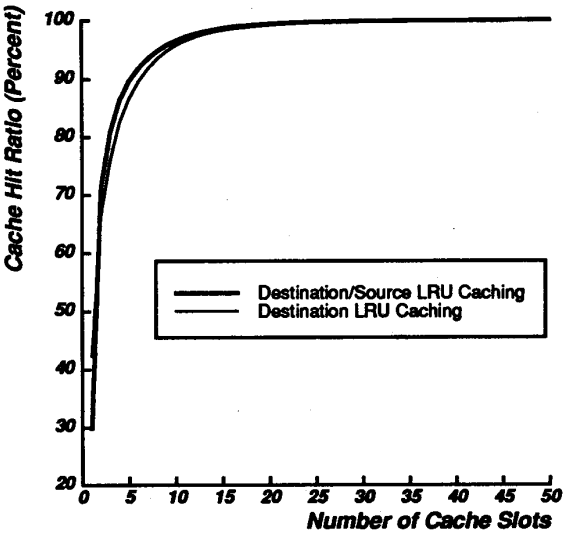

Figure 3-6: Percentage of Cache Hits versus Cache Size

Another way of looking at this data is to plot the average number of packets through the gateway between cache misses. The graph in figure 3-7 shows the incremental efficiency of each additional slot in the cache. Curves are shown for destination LRU, destination/source LRU, destination FIFO, and destination/source FIFO.

As expected, the destination/source LRU cache performed best, followed closely by the destination LRU cache. Performance of both FIFO caches is relatively poor until the destination FIFO cache size exceeds 102. It turns out that 52 is the number of distinct destination addresses handled by the gateway in a 24 hour period, in addition to 52 destinations in the preload, for a total of 104. For destination/source caching, the FIFO buffer

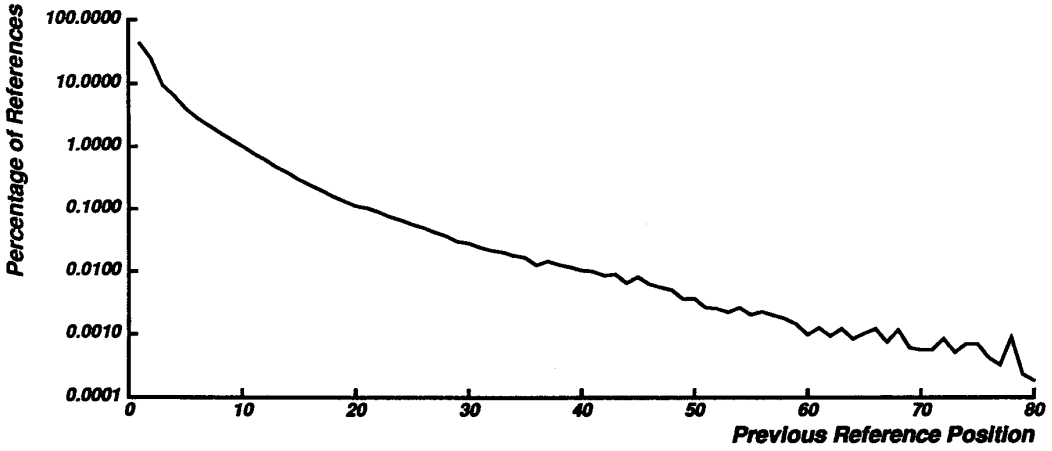

Figure 3-5: Percentage of Reference versus Time of Last Reference (Destination)

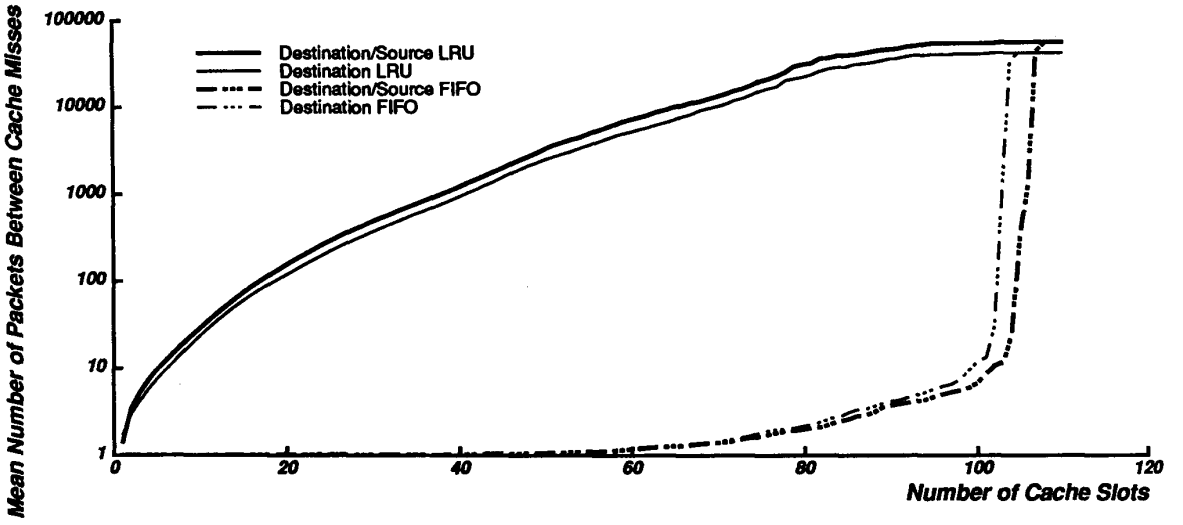

Figure 3-7: Number of Packets Between Cache Misses versus Cache Size (log-linear)

size does not help until 104 slots, which is about the number of distinct destination and source addresses processed by the gateway in a 24 hour period, 54 plus 53 during preload for a total of 107.

Table 3-2: Cache Hit Ratio Percentage versus Cache Size

Number of Cache Slots	Destination Only	Destination Source
1	42.31	29.94
2	66.54	71.28
3	76.03	80.81
4	82.52	86.32
5	86.61	89.61
6	89.53	91.81
7	91.74	93.48
8	93.44	94.76
9	94.75	95.77
10	95.77	96.56

4. Caching Cost Analysis

Although the hit ratios have been given for various gateway caches, the important question is what is the performance improvement that is gained for the gateway. The following analysis is meant to be a simple worst-case analysis. The cache used for this analysis is a fully associative cache with a LRU replacement policy. The cache is implemented completely in software as a doubly-linked linear list. The list is searched linearly from most recently used to least recently used and when the list is rearranged to preserve the LRU ordering, this is done by swapping pointers of the doubly-linked list. Only destination addresses are cached.

Estimates of the number of instructions are based on a load-store machine architecture. Loading the pointer to the cache requires 1 instruction. A matching entry is recognized in 3 instructions; in addition, 7 instructions are needed for each previously checked cache slot. After each cache search, the cache must be reordered to preserve the LRU ordering (except if the entry was found in the first slot) and this takes 21 instructions (except for the last cache slot, which needs 11 instructions).

An estimate of the minimum-time address decode and routing table lookup for a packet is 80 instructions, a figure obtained by estimating the number of load-store instructions generated by the C gateway code at MIT. The current gateway code would take even longer; this estimate is conservative because it does not include function call overhead, error handling, or the additional routing table lookup necessary for subnet routing and assumes that there are no hashing collisions. This estimate also does not include the time necessary to map a next-hop IP address into a local network address, which is at least 2 instructions, but varies depending on the local network.

The average lookup time with the cache is now:

$$1 + 3p_1 + (7+3+21)p_2 + (14+3+21)p_3 + \cdots + (7[k-1]+3+21)p_k + \cdots + (7[n-1]+3+11)p_n + \left(1 - \sum_{i=1}^{n} p_i\right)(7n+21+80)$$

If the cache stores flat addresses and destinations only, then the optimum number of cache slots is 16. The average lookup time with the cache becomes 37.9 instructions, only 47% as long as a table lookup. One problem with LRU caches is that it is expensive to maintain LRU ordering. A way to reduce this expense is to use only two entries and a single bit to determine

which is first in order. In this case, the average lookup time is 33.9 instructions, 42% as long as table lookup. For a source-destination two-element cache, the average lookup time is 29.6 instructions or 37% of table lookup.

With hierarchical addressing, the performance is even better. To obtain a network address from a flat address, the last byte is dropped from the address. Although this does not necessarily result in a network address, the number of distinct addresses that the cache must handle is reduced and the overhead is only a single AND instruction. The optimal cache size is 51 slots and the average lookup time is 31.5 instructions, 39% as long as table lookup. A two element simple cache as described above averages 33.3 instruction, 42% of a table lookup. A source-destination cache requires 27.7 instructions, or 35% of a table lookup.

Although the figures above minimize average lookup time, it may be that a lower maximum lookup time is desired at the cost of a slightly higher average lookup time.

Obviously, a better cache implementation would increase performance. Cache lookup could use a hash table rather than a linear search, the cache slots need not be fully associative, a replacement strategy approximating LRU may be implemented less expensively, the cache fetch strategy could use both destination and source addresses, or a hardware cache could be built into the system. But even with this simple cache implementation and optimistic assumptions about routing-table lookup time, the best cache above reduces lookup time by 65%.

5. Conclusion

Processing time per packet is reduced if average routing-table access time is reduced, and an economical way to reduce access time is to use a cache. Recordings were made of routing-table accesses for several operating gateways and these records are used to drive cache simulations to determine the hit ratio for various types of caches. The caches simulated are fully-associative, LRU or FIFO, and cached destination addresses or destination and source addresses.

The probability of reference to a destination address versus time of previous reference to that address is monotonically decreasing for up to 50 previous references, implying that an LRU cache management procedure is optimal for caches of 50 slots or less.

Locality of packet flat addresses causes an LRU cache of 9 slots to have a hit ratio of over 90%. Hierarchical addressing measurements place hit ratios at over 90% for 7 slots.

In addition to caching the packet destination/next-hop pair, it is worth caching the packet source/next-hop pair if it can be done inexpensively.

The data indicate that back-to-back caches may be the most effective implementation of large caches. This first cache should have an LRU management policy to cache packets arriving in a train. Cache misses from this cache should then check a FIFO cache to check for trains from previously seen destinations.

Hierarchical addressing recognition allows a higher cache hit ratio at the expense of address decoding each packet destination address. Simple preprocessing before a cache for flat addresses provides a hit ratio not as high as that for hierarchical addressing, but higher than the hit ratio for flat addressing at little cost.

A simple conservative cost analysis shows that current gateway routing-table lookup time can be reduced to 35% of its current time. This is a conservative estimate and a good cache design or a hardware cache could further reduce the average lookup time.

The hit ratio for a flat address cache is high enough that flat addressing may be practical to use in a gateway with a cache.

6. Acknowledgments

I would like to thank Dave Clark for suggesting the idea of a routing-table cache in a gateway. My thanks also to Thu Nguyen and an anonymous reviewer for their comments.

References

1. Alan Jay Smith, "Cache Memories", *ACM Computing Surveys*, Vol. 14, No. 3, September 1982, pp. 473-530.

2. R. Jain and S. Routhier, "Packet Trains - Measurements and a New Model for Computer Network Traffic", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 6, September 1986, pp. 986-995.

3. David C. Feldmeier, "Traffic Measurements on a Token Ring Network", *Proceedings of the 1986 Computer Networking Symposium*, (Washington, DC, November 1986), pp. 236-243.

4. David C. Feldmeier, "An Empirical Analysis of a Token Ring Network", Technical Memo TM-254, MIT Laboratory for Computer Science, January 1984.