A Study of Cache-based IP Flow Switching*

Osman N. Ertugay
Jonathan M. Smith
Department of Computer and Information Science
University of Pennsylvania

16th November 2000

Abstract

Meeting the service demands from QoS-based network applications is a very challenging task performed in many high-end routers and switches. This task involves management of resources like bandwidth and memory in network devices. The memory in the form of a very fast cache that instruments wire-speed classification, discrimination, and forwarding of network packets needs to be managed very effectively. We examine the management of a specific IP flow-cache architecture through simulations based on traffic traces collected from a campus intranet. A probabilistic cache install policy is examined over a range of cache sizes and install probabilities. This policy succesfully identifies the flows that warrant caching and slightly improves the performance they receive. We argue that dynamically deployable policies based on site-specific traffic information can increase the switching performance even higher.

1 Introduction

Most high-end routers and layer-3 switches provide wire-speed forwarding of IP flows combined with a range of QoS guarantees. At the heart of such a high-end system is usually a very fast cache mechanism that allows several million IP flow lookups per second. An IP flow is defined as a sequence of one or more IP packets with the same protocol, destination address, source address, destination port, and source port fields. This, in practice, restricts our interest only to TCP and UDP packets, which make up most of the IP traffic. In such a system, an important decision made for each arriving packet is whether to perform a flow-based lookup or a regular lookup based on the IP destination address only. This decision has to be made at the wire-speed, hence usually by a hardware filter mechanism. Hardware filter mechanisms are usually much coarser compared to the number of all possible flows simply because the time per packet to

*This research was supported by the NSF grant ANI99-06855 and by the 3Com Corporation.

make a decision at the wire-speed is very small. A coarse filtering mechanism, consequently, leads to many unintended packets being looked up by flow-id. When a flow-based lookup fails, forwarding of the corresponding packet falls back to the software. The software, apart from forwarding the packet, makes a decision on whether or not to install a new cache entry for the flow that the packet belongs to. This decision also has a major effect on the forwarding performance.

In this paper, we provide an analysis of forwarding performance for a probabilistic cache installation policy. We simulate a specific cache-based forwarding architecture with IP traffic traces collected from an intranet. Section 2 describes the traffic traces we collected. Section 3 outlines the flow-switching architecture. Section 4 describes the simulations we performed. The simulation results are discussed in Section 5.

2 Traffic Traces

We collected traffic traces from three subnets of the SEASNet at the University of Pennsylvania. A Linux PC (dual Pentium-pro 200MHZ, 192MB RAM) with three Etnernet NICs was used. Each NIC was attached to one of the subnets. We ran a tcpdump process for each subnet simultaneously. The output file from the tcpdump process for each subnet was then merged into a single file. Order of the packets in the merged file was determined by the packet timestamps. Since some packets (those travelling between the three subnets that we traced) appeared twice in the merged file, we removed the duplicates by checking if a packet was seen before (to be more correct, if a packet was seen in the last 3 seconds; packets were identified by the IP protocol, source address, destination address, source port, destination port, ident, and length fields.) Consequently, the resulting file contained TCP and UDP packets only. Traffic other than TCP/UDP makes up around 2% of the overall traffic in the traces. (1.56% and 2.82% in the two traces we collected)

Before the merging process, traces were also analyzed to see if the capturing process had dropped any packets. We identified the TCP flows and found out the gaps in TCP sequence numbers. Based on our analysis, we concluded that at most 1% of the traffic might have been drooped by our capturing process.

Two sets of traces were collected on two different days during the same time period. The IP addresses appearing in the traces were scrambled due to privacy issues. The scrambling process was designed such that the flow information (end-to-end addressing associations) in the traces were preserved.

Table 1 provides information about the collected traces.

3 Flow-switching Architecture

We used a 4-way set-associative flow-cache with LRU replacement policy for our simulations. Cache sizes ranging from 64 to 4096 rows were used throughout the simulations. All lookups are based on the IP flow-id. A

	Trace1	Trace2
Date	Tue Dec 21 1999 14:05	Tue Dec 22 1999 14:05
Duration	3.5 hrs	3.5 hrs
Number of packets	12,614,772	11,142,250
Number of flows	124,476	81,437
Packet rate	1001 pkts/sec	884 pkts/sec
Bit rate	2.91 Mbits/sec	2.99 Mbits/sec
Percentage of TCP traffic	97.9%	98.2%
Percentage of external traffic	16.6%	6.9%

Table 1: Traffic Information

successful lookup leads to hardware forwarding for the packet. Lookup failures are handled by the software. The software forwards the packet, and makes a decision for whether to install a new flow-cache entry or not. Ideally, one would like to install a new entry if sufficiently large number of packets for that flow will arrive in the near future and/or an existing entry which is taking lots of hits is not kicked out by the new entry. Most importantly, time and space complexity of the decision algorithm should not degrade the system performance.

We came up with a probabilistic installation policy: whenever a lookup failure occurs, we install a new cache entry only with probability p. The motivation behind this policy is:

- flows with small number of packets and/or small packet rates (hence not warranting a flow-cache entry) will have less chance of being installed, thereby not disturbing already installed flows,
- thrashing due to a set of flows continually kicking each other out of the cache will decrease,
- time and space complexity of the install policy is constant, no need to keep per flow state.

4 Simulations

We simulated the flow-switching behaviour of the above described architecture with the collected traffic traces for install probability values ranging from 0.1 to 1.0 with step size 0.1 and cache row numbers 64, 128, 256, 512, 1024, 2048, and 4096. In each simulation run, we kept the following information for each flow:

- number of packets
- number of hits
- number of misses (slow-path taken)
- installs
- installs that were kicked out without getting any hits

- time first packet was seen
- time of first install (if any)
- number of packets seen until eventually installed
- time last packet was seen

After each simulation run, the resulting flow information file was analyzed to produce the charts and tables seen in the next section.

5 Results

We classify the flows into two categories: *uncached flows* (flows that were never installed into the cache), and *cached flows* (flows that were at sometime installed into the cache.) For all the charts, Trace1 is shown on the left, and Trace2 on the right.

Figure 1 shows the average hit percentage for all flows. This figure suggests that even very small cache sizes is able to provide acceptable performance for the traffic traces used.

Figure 2 shows the average hit percentage for the *cached flows*. This figure discounts the initial misses up to the first install for a cached flow and the misses for uncached flows. The hitrate for cached flows increases in general as the install probability decreases. This comes at the expense of increased number of slow-path packets per second, which is graphed in Figure 5. However, the slow-path packet rate is way less than the software can handle, and also the number of cache installs per second (Figure 6) is significantly decreased. Furthermore, Table 2 shows that uncached flows make up a significant portion of all flows while the total number of packets belonging to uncached flows make up an insignificant portion of all packets. This means that our probabilistic install policy prevents the right set of flows from being installed into the cache, and hence, the increased slow-path rate (or decreased overall hitrate) is mostly due to flows not warranting a cache install.

Figure 7 shows the percentage of cache installs that got no hits, i.e., potentially harmful installs. This percentage decreases linearly as the install probability decreases.

Table 3 shows the average number of packets that it takes for a flow to get installed in the cache. This value as well as the values in Table 2 have been observed to be very close for all cache sizes, i.e., they are independent of the cache size.

6 Conclusions

We observed that a router connecting three typical subnets to each other and to the rest of an intranet can perform flow-based switching effectively even with very small cache sizes. We analyzed a probabilistic cache install policy, alternative to an "always install" policy. We saw that a significant portion of the flows that does not warrant caching can be identified and routed in the software without degrading system performance. This leads to a higher hit percentage for cached flows, less cache installations, and

smaller proportion of cache entries that get no hits. We believe that augmenting this policy with live statistical information (e.g., per destination port or per cache row traffic statistics) may lead to further performance improvements. We actually analyzed such a policy which applies probabilistic installs only to cache rows that get significantly higher number of installs than the average, and observed an increase (very small though) in the *overall* hit percentage. For some cache rows, we observed a significant increase in the hit percentage. An install policy that can be customized and deployed dynamically based on site-specific traffic information provided by network managers may also be desirable. The installation policy module may be programmed dynamically by managers and injected into the routers which are active-networking enabled.

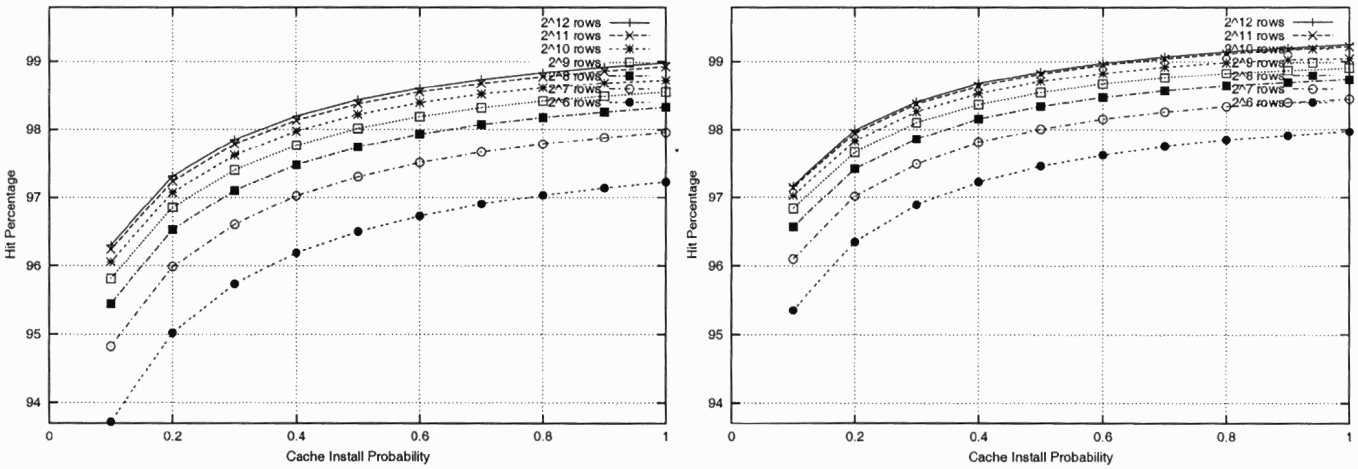

Figure 1: Hit percentage

	Trace1				Trace2			
cache install probability	0.7	0.5	0.3	0.1	0.7	0.5	0.3	0.1
avg packets/flow	1.05	1.20	1.71	3.14	1.04	1.18	1.64	3.14
max packets/flow	7	14	23	69	7	14	21	76
% uncached flows	12.6	22.1	35.5	62.6	12.9	22.3	35.5	61.4
% uncached flow packets	0.13	0.26	0.60	1.94	0.10	0.19	0.43	1.41

Table 2: Statistics for uncached flows

	Trace1				Trace2			
cache install probability	0.7	0.5	0.3	0.1	0.7	0.5	0.3	0.1
avg packets/flow before install	1.27	1.65	2.40	4.71	1.27	1.65	2.43	4.98

Table 3: Statistics for cached flows

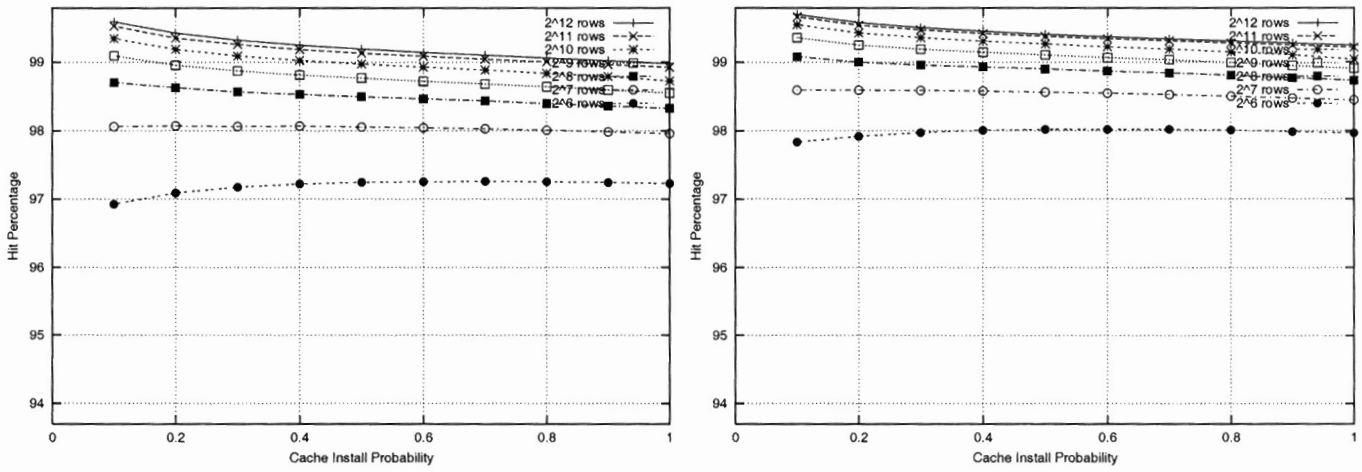

Figure 2: Hit percentage for cached flows

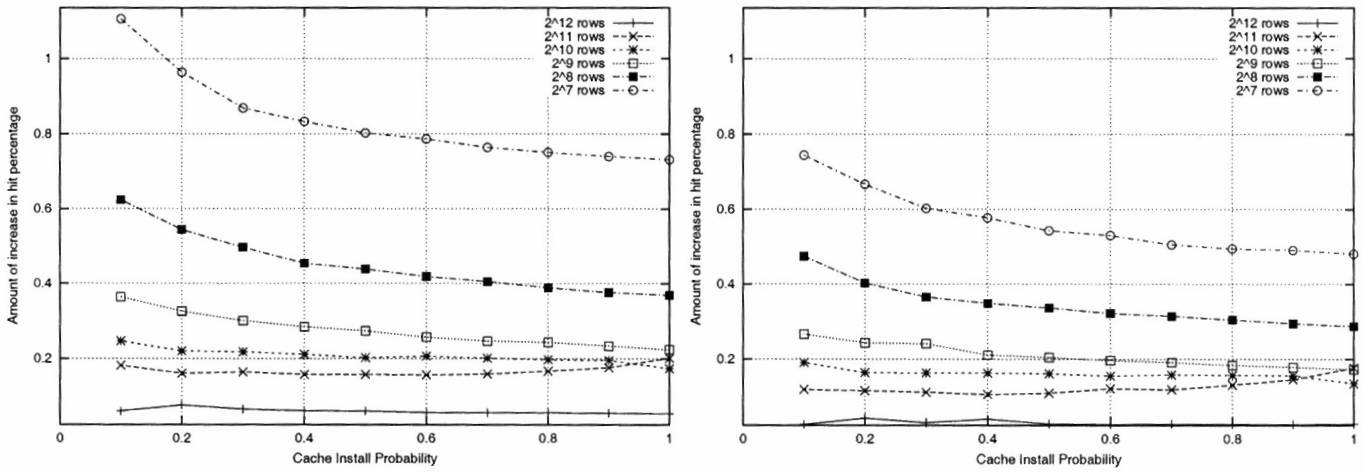

Figure 3: Hit percentage increase as cache size is doubled to 2^n rows

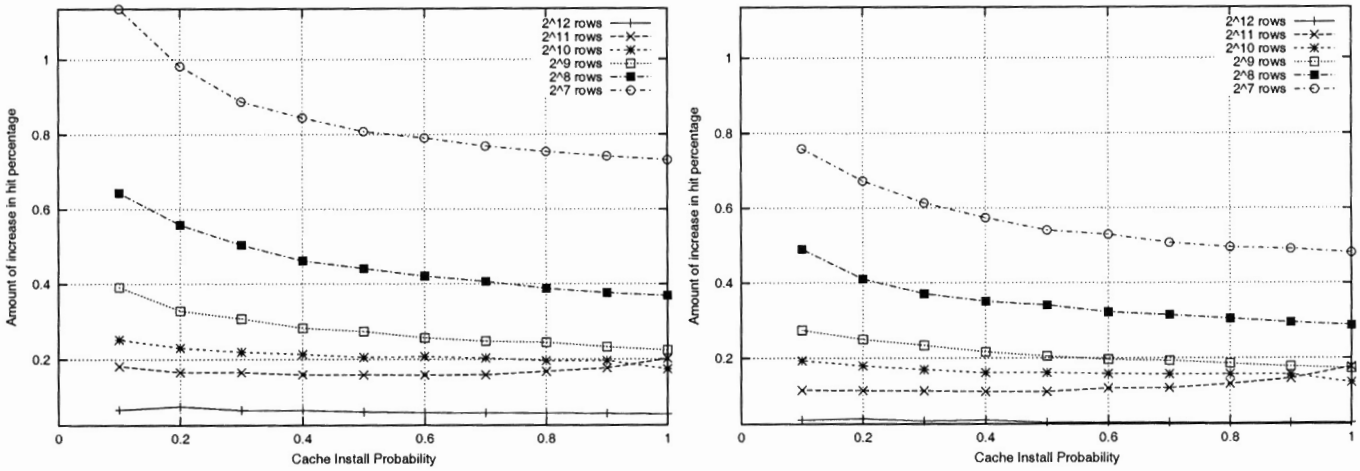

Figure 4: Hit percentage increase for cached flows as cache size is doubled to 2^n rows

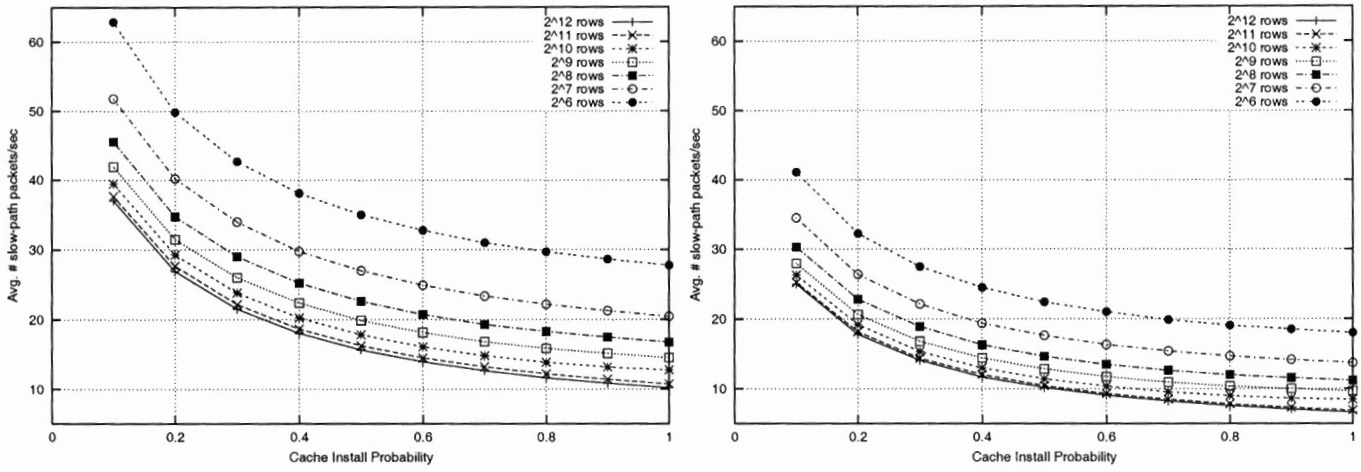

Figure 5: Slow-path packets/sec

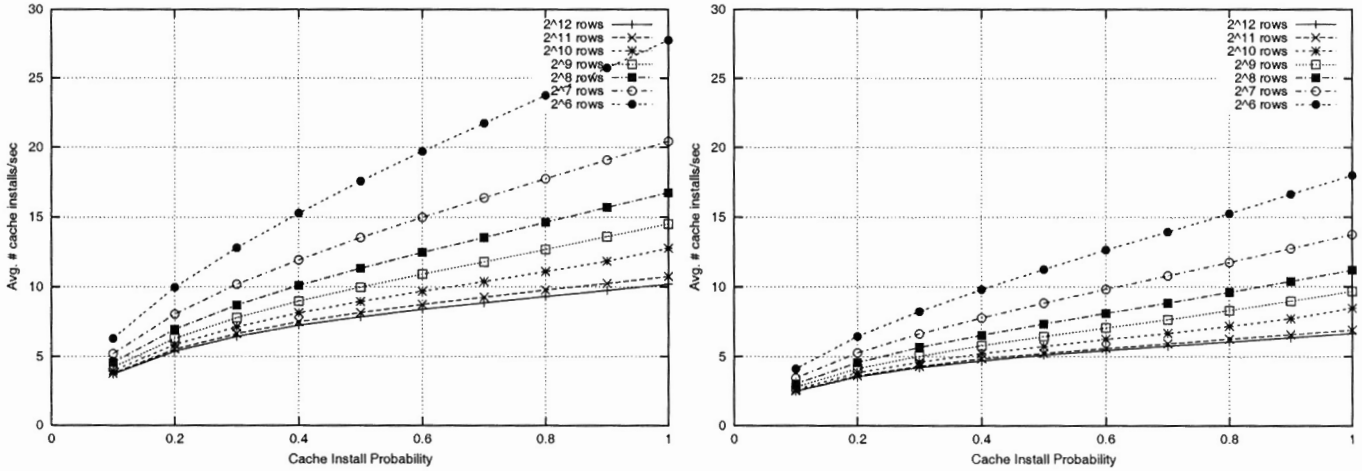

Figure 6: Cache installs/sec

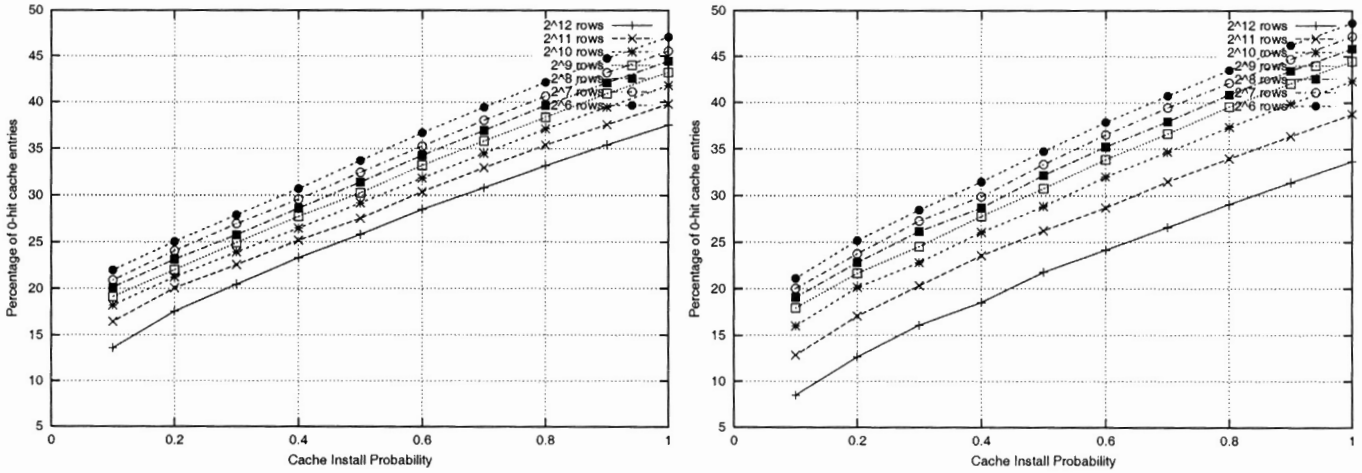

Figure 7: Percentage of cache installs with no hits