Study of Flow Caching for Layer-4 Switching

Ye Tung and Hao Che
Department of Electrical Engineering
Pennsylvania State University
State College, PA 16802, USA

Abstract— **Next generation access routers and edge devices need to provide functionalities for layer-4 packet forwarding and firewall/security checks. Consequently, a challenging issue concerns how to achieve fast packet filtering and forwarding at low cost. This paper studies flow caching mechanisms for fast layer-4 packet forwarding. We show by model analysis that flow caching performance is not very sensitive to cache table lookup speed but it is sensitive to cache hit ratio. By making use of the available layer-4 information, we introduce two filtering modules to reduce the cache miss ratio. We demonstrate, by real trace simulation, that by adding these two filtering modules, the cache miss can be decreased by up to 50% and the requirement for full header filtering speed also been greatly reduced. The proposed flow caching mechanism is potentially useful for routers and switches where software based filtering modules are dynamically generated. It is also provide a cost-effective migration path for upgrading of the existing vastly installed base of routers with flow caching to value-added high speed routers with flow caching, which offer integrated/differentiated services.**

Keywords— **Flow caching, Layer-4 switching, Cache-hit/miss, Port information, Full header filtering**

I. Introduction

Next generation high speed routers and switches need to be able to provide functionalities, such as layer-4/layer-5 forwarding and firewall/security checks [2]. With these functionalities being added for packet processing, a challenging issue concerns how to achieve fast packet filtering.

There are two basic approaches to enable fast packet filtering with high dimensional filtering rules. One approach is to follow the traditional wisdom by using flow caching mechanism. In this approach, only the first packet of a flow needs full header filtering and table lookup. The rest of the packets of the flow are cut-through switched through flow cache entry lookups. The other approach is to allow packet by packet full header filtering and table lookup using the state-of-the-art software/hardware based fast packet filtering algorithms, e.g., [1], [3], [11]. This approach is desirable when the set of filtering rules are updated infrequently and a single multi-dimensional filtering table can be used for filtering. On the other hand, in lots of cases, due to relatively frequent administrative policy changes, the filtering rules need to be dynamically generated using, e.g., software plugins [4]. It is infeasible to build a large single static filtering table for fast packet filtering [4]. Multiple filtering table lookups need to be performed , which changes dynamically. Since relatively small numbers of concurrently active flows In a value-added software architecture for next-generation routers in [4], flow caching mechanism was successfully used to achieve much higher packet forwarding speed compared with the forwarding speed of the best-effort kernal.

Flow caching is an attractive solution for fast packet filtering on routers and switches also due to the following reasons. First, per packet full header filtering consumes more clock cycles as more filtering rules are added, whereas per packet cache table lookup is independent of the number of filtering rules in use. Second, from economic point of view, it is advantageous to exploit flow caching mechanism for value-added routers, simply because it provides a natural migration path for upgrading the vastly installed base of flow caching based routers to enable value-added services.

A major concern with flow caching is associated with cache miss penalty. For flow cache based packet forwarding, upon a packet arrival, the flow cache table is searched first. When a cache miss occurs, a full header filtering is performed for packet forwarding. This results in a cache miss penalty due to possibly large processing delay. Nevertheless, we note that the cache miss penalty would have little effect on the Quality of Service (QoS) of a connection for

0-7803-6494-5/00/$10.00 © 2000 IEEE

the following reasons. A cache miss occurs only for the first packet of a packet burst. This packet can either be the first packet of a connection or the first packet after the connection has been idle for a duration longer than the flow cache timeout value. For the former case, a small delay of the first packet of a connection should not have much impact on the overall QoS of the connection, especially when the first packet is a connection setup packet. For the latter case, a small delay of the packet does not do much harm to the QoS either because comparing with the large idle time which is on the order of at least several seconds, a small processing delay is negligible. Hence, the cache miss penalty should not be a primary concern for using flow caching in terms of QoS guarantee.

This paper aims at addressing the performance and design issues pertaining to flow caching for value-added routers. First, based on a simple queueing model, we quantitatively characterize the performance gain of the flow caching mechanism in terms of reduced number of clock cycles per packet forwarding at a given cache hit ratio. We show that flow caching performance is not very sensitive to the flow cache table lookup speed but it is sensitive to the cache hit ratio. Then, we show how the available layer-4 header information can be used to improve cache hit ratio. In particular, by making use of the source and destination port numbers from the transport layer header, we are able to further reduce cache hit miss by up to 50 %, resulting in at least a three-fold performance gain.

The remaining sections are organized as follows. In Section 2, the problem of packet filtering with flow caching is formally defined. In Section 3, the packet filtering process is modeled as an M/H2/1 queueing system. The results are presented and their implications on the effective router design are discussed. In Section 4, two mechanisms for the enhancement of the performance for flow caching are proposed. In Section 5, the statistic analyses of the proposed mechanisms are given based on campus/backbone trace simulations. Finally, Section 6 gives the conclusions and future work.

II. Problem Formulation

Fig. 1 gives a schematic diagram for the major components of a value-added router with flow caching. The system is composed of four parts, including input ports, output ports, a packet full header filtering module, a flow cache table, and a switching fabric. The flow cache table contains flow entries, typically indexed by five header fields *(source address, destination address, source port number, destination port number, protocol type)*, and the packet forwarding information such as the output port number, the priority level, etc. When a packet arrives at an input port, its header information is first used to match with the flow cache entries. If a match is found, a cache hit occurs and the packet is switched to the output port queue according to the forwarding information specified in the matched flow entry. This is a fast path forwarding. If no match is found, a cache miss occurs and the packet header information is redirected to the packet full header filtering module for filtering. The forwarding information obtained from the filtering module is then used to switch the packet to the output port, a slow forwarding path. Then the forwarding information is cached in the flow cache table, creating a fast forwarding path for the subsequent packets of the flow.

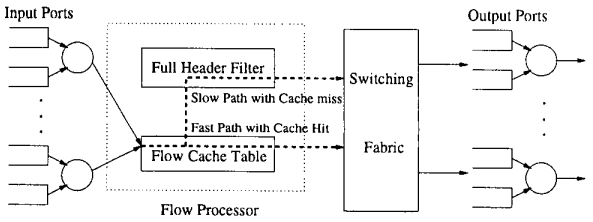

Fig. 1. Schematic functional modules of a router

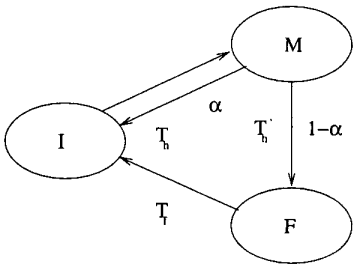

Fig. 2. A finite state machine for filtering system with flow caching

The two data paths can be summarized by means of a finite state machine as shown in Fig. 2. It is composed of an idle state **I**, a flow cache table matching state **M**, and a full header filtering state **F**. Upon a packet arrival, the system moves from state **I** to state **M**. If there is a cache hit, the system moves back to state **I** after T_h clock cycles. If there is a cache miss, the system moves to state **F** after T'_h clock cycles. Here T_h and T'_h are flow entry matching times with and without a cache hit, respectively. We assume that on average, a cache hit occurs with probability α and a cache miss occurs with probability $(1-\alpha)$, called

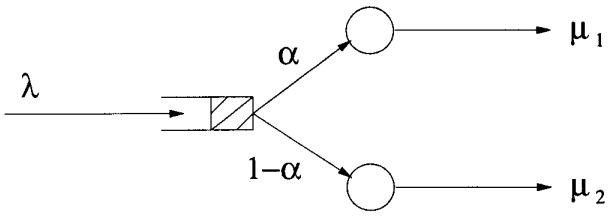

Fig. 3. M/H2/1 queueing model for filtering with flow caching

cache hit ratio and cache hit ratio, respectively. For a packet with a cache miss, the system stays in state **F** for T_f clock cycles before jumping back to state **I**. Here, T_f is the number of clock cycles required for a full header filtering. Flow caching is normally implemented using hashing and hash collision is resolved by storing all the entries in the same hash bucket using a single linked list. So in general, $T_h' \geq T_h$ with $T_h' \approx T_h$. For flow caching to be useful, we should have $T_h << T_f + T_h'$ or $T_h << T_f$ and α close to 1.

III. Model Analysis

First, we consider the case without flow caching. We model it as a simple M/M/1 queueing system with Poisson packet arrival rate and exponential service rate for packet full header filtering. Denote the average packet arrival rate as λ and the average filtering rate as μ^o. The performance is measured in terms of the tail distribution of the response time, i.e.,

$$P_T(t > \tau) \leq 10^{-\delta} \tag{1}$$

where t is the response time or the total time of a packet traversing the router. $P_T(t > \tau)$ is the tail distribution of t when t exceeds τ. τ and δ are design parameters.

The response time of the tail distribution for M/M/1 queue can be written as [7],

$$P_T(t > \tau) = e^{-\mu^o(1-\lambda/\mu^o)\tau} \tag{2}$$

A reasonable assumption is to choose δ so that when $\tau = \frac{1}{\mu^o}$ (the average full header filtering time), the equality in (1) holds. For instance, if we set $\lambda = 1$ *Mpps* (Mega packets per second) and $\mu^o = 3.3$ *Mpps*, we find that $\tau = 0.3$ μs and $\delta = 1.0$, i.e., the probability for the response time longer than the average full header filtering time is 10%.

Now let's model flow-caching based filtering as an M/H2/1 queueing system. The packet arrival process is still Poisson with average arrival rate λ. The service process is a phase-type with two phases as shown in Fig. 3. A packet has a cache hit probability α to be serviced with exponential service rate μ_1 and it has a cache miss probability $(1-\alpha)$ to be serviced with exponential service rate μ_2. With reference to the finite state machine in Fig. 2, we can write the mean service rates in terms of T_h, T_h' and T_f as follows,

$$\mu_1 = 1/T_h, \qquad \mu_2 = 1/(T_h' + T_f) \approx 1/(T_h + T_f) \tag{3}$$

For M/H2/1 queue, it can be shown that the tail distribution can be expressed as

$$P_T(t > \tau) = \frac{1-\rho}{B_1 - B_2}\left[\frac{B_1\gamma - \beta}{B_1}(1 - e^{-B_1\tau}) + \frac{-B_2\gamma + \beta}{B_2}(1 - e^{-B_2\tau})\right] \tag{4}$$

where

$$\begin{aligned} \gamma &= \alpha\mu_1 + (1-\alpha)\mu_2, \\ \beta &= \mu_1\mu_2, \\ \rho &= \lambda(\alpha\mu_1^{-1} + (1-\alpha)\mu_2^{-1}), \\ B_1 &= (\mu_1 + \mu_2 - \lambda + A), \\ B_2 &= (\mu_1 + \mu_2 - \lambda - A), \\ A^2 &= [\mu_1^2 + \mu_2^2 + \lambda^2 - 2\mu_1\mu_2 - 2\lambda(2\alpha - 1)\mu_1 \\ &\quad -2\lambda(1-2\alpha)\mu_2]/4. \end{aligned} \tag{5}$$

By setting $\lambda = 1$ *Mpps*, $\mu^o = 3.3$ *Mpps*, $\tau = 0.3$ μs, and $\delta = 1$, the triplet (μ_1, μ_2, α) can be calculated by substituting (4) into (1) with equality. we plot, in Fig. 4, μ_2 against α at different μ_1 values, where μ_1 takes values larger than μ^o. We first observe that the curves converge quickly to an asymptotic one as μ_1 increases. The flow cache table lookup rate faster than $\mu_1 = 2\mu^o$ helps very little in terms of reducing μ_2 or full header filtering rate for any α values.

For $\alpha = 0$, $\mu_2 \approx \mu^o = 3.3$ *Mpps* as expected. When α increases, μ_2 decreases pretty fast. At $\alpha = 0.9$ and $\mu_1 = 2\mu^o$, $\mu_2 = 0.8$ *Mpps*. Let $T_h' = T_h$, we found that $T_f = 1.27$ μs, or the full header filtering rate of

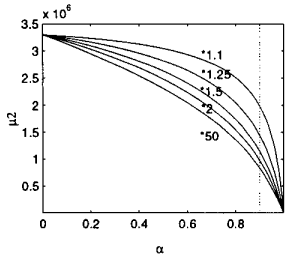

Fig. 4. μ_2 vs. α for various $\mu_1 = \mu^o * m$ ($m = 1.1, 1.25, 1.5, 2, 5, 50$) at wire-speed

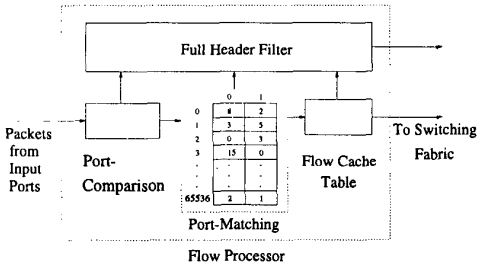

Fig. 5. A New Flow Caching Model

0.7 *Mpps*, which is about 5 times smaller than the required rate $\mu^o = 3.3$ *Mpps* for packet-by-packet full header filtering. Clearly, if $\alpha \geq 0.9$ and $\mu_1 \geq 2\mu^o$ can be achieved, the flow caching mechanism is very effective.

The above analysis shows that the flow cache table lookup speed does not have to be much faster than the full header filtering speed and it is the cache miss ratio that matters. In fact, flow cache search speed is in general much faster than full header processing speed (about 20:1 according to [4]). In the following section, we shall focus on the study of cache miss ratio and design two schemes with low processing cost to greatly reduce cache miss ratio.

IV. Flow Caching Mechanisms

In this section, we propose two mechanisms for the improvement of cache hit ratio. The idea is to take advantage of layer-4 header information which is to be processed for full header filtering. In particular, we make use of {*source port*, *destination port*} information in the transport layer header to improve cache hit ratio for value-added packet forwarding.

Note that flow caching works only for *long-lived* flows composed of many packets but not for *short-lived* flows. Hence, a key to improve cache hit ratio is to identify short-lived flows and put them through full header filtering without caching them. Based on real trace analyses, we find that in general, a packet with identical source port number and destination port number is a server-to-server application packet and the corresponding application flows are short-lived with only a few packets. An example is DNS application, which has identical source and destination port numbers and is very short-lived. Hence, our first mechanism is to add a port comparison module to identify server-to-server applications.

Observe that most of the client-server application packets have one of their port numbers, either source port number or destination port number, taking values between 0-1023 (with some exceptions), known as the well-known port numbers, and the other port number randomly assigned between 1024-65536, known as the unknown port numbers. Since the unknown port numbers are randomly assigned, the probability that two flows have the same unknown port number is small. Hence, our second mechanism is to keep a record of the unknown port numbers for all the flows in the cache table and do an unknown port number match before flow cache entry search.

Fig. 5 shows the two added modules based on the proposed mechanisms. The first module is called port-comparison module, where the value of the source port number of an arrived packet is compared with the destination port number of the packet. If the two port numbers are found to be identical, the packet is immediately passed to the full header filtering module. Otherwise, the larger port number of the two, which is the unknown port number, together with a single bit identifier is passed to the second module, called port-matching module. The one-bit identifier takes binary value 0 if the unknown port number is the source port number, otherwise it takes binary value 1. In the port-matching module, there is a 65537x2 table as shown in Fig. 5. The kth row contains the information for port number k. The first column corresponds to source port and the second column corresponds to destination port. Each table entry records the number of active flows in the flow cache table with source (destination) port number k. The unknown port number and the one-bit identifier are used as table indices to locate the entry with the same port number and identifier. If the entry value is zero, there must be no flow entry in the flow cache table that matches with the arrived packet and the packet header is passed to the filtering module for full header filtering. Otherwise, the packet header is forwarded to the flow cache table for flow cache entry search as shown in Fig. 5.

In summary, with port-comparison module, the flow caching is avoided for server-to-server short-lived applications and thus save flow cache memory for other application flows. The port-matching module further reduces the probability of flow cache miss. For layer-4 forwarding, both source and destination port numbers are to be filtered for the identification of end-to-end applications and the added processing overhead is negligible.

The processing overhead for port-comparison and port-matching involves at most one 16-bit comparison and one bit setting, and one table indexing and one 16-bit comparison, respectively. Obviously, the added

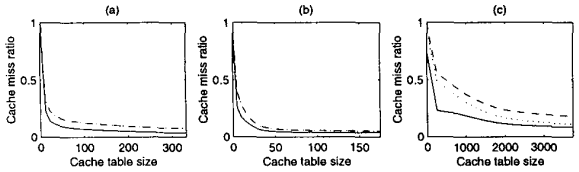

Fig. 6. Cache miss ratio vs. cache table size("- -" for traditional flow caching, "..." for flow caching with port-comparison, solid line for flow caching with port-comparison and port-matching. (a) cisco-trace, (b) lbl-trace, (c) fixwest-trace.)

processing overhead is small compared with flow cache table entry search. Note that if flow hashing combined with a link list for collided flow entries per hash key is used for flow cache management, a flow cache entry search involves a hash key calculation, a hash key indexing, and a link list search with five fields matching for each searched entry. The added overhead for the management of the port-matching module is also small. Upon each flow cache entry deletion (addition), the corresponding table entry value in the port-matching module is decremented (incremented) by 1. Each of these operations involves one comparison, one bit setting, one table indexing, and one decrement/increment. Since flow cache entry updating interval is at a much longer time scale compared with packet processing, this added overhead for flow cache management is negligibly small.

V. Simulation and Performance Analysis

Since cache search speed is not a major performance constraint, our trace simulation focuses on finding α values at different cache table sizes. Two campus traces and one backbone trace are used for simulation. The traces are referred to as $cisco-trace$, $lbl-trace$ and $fixwest-trace$, respectively. The $cisco-trace$ is a 20-minute trace collected from a 100-BT campus network at Cisco Systems Inc. on March 4, 1997. The $lbl-trace$ is a 16-minute trace collected from a 100-BT at Lawrence Berkeley Laboratory (LBL) on July 14, 1997. The $fixwest-trace$ is a 20-minute trace collected from the FDDI Internet backbone at FIXWEST on Oct. 21, 1996. The utilizations at the time of data collections are 5.5%, 4.0%, and 27.3%, respectively.

First, we need a flow cache entry timeout mechanism. We use a simple adaptive flow entry timeout algorithm as proposed in [5]. Namely, the timeout value T_n is periodically re-assigned at time n ensuring that the flow cache utilization ρ is high. This mechanism is found to offer very close performance to the least recently used (LRU) algorithm with much lower computational complexity. Since the flow cache utilization is positively correlated with T_n, T_n can be updated simply based on the following control scheme,

$$T_n = \begin{cases} T_{n-1} + \Delta T, & \text{if } \hat{\rho}(n-1) \le \rho_{min}; \\ \max\{T_{n-1} - \Delta T, T_{min}\} & \text{if } \hat{\rho}(n-1) \ge \rho_{max}; \end{cases} \quad (6)$$

with $0 < \rho_{min} < \rho_{max} < 1$. Here, T_{min} serves as the lower bound to avoid the thrashing effect. In order to avoid over-reaction to small demand variations, we introduced a first-order low-pass filter operation to damp the variation in $\rho(n)$,

$$\hat{\rho}(n) = (1-\omega)\hat{\rho}(n-1) + \omega\rho(n) \quad (7)$$

where ω is the weighting factor taken values between 0 and 1. One can strengthen the damping by choosing a small ω. In our simulation, $\omega = 0.5$, $\Delta T = 2$ sec, $\rho_{max} = 0.98$, and $\rho_{min} = 0.9$.

The cache miss ratio is calculated as a function of flow cache table size for each trace. Three cases are studied, i.e., the traditional flow caching, the flow caching with port-comparison, and the flow caching with both port-comparison and port-matching. The results are shown in Fig. 6. With port-comparison, nearly 20% of the total packets from the backbone trace are directly sent to the filtering module and the cache miss ratio almost drops by half at flow cache size of 3755. But for the two campus traces, the port-comparison does not help much. The total numbers of packets which are directly sent to the filtering module are less than 0.5% for both cases. The cache miss ratios stay almost the same. This is due to the fact that there is huge amount of server-to-server traffic in the backbone environment, but not in a campus environment. With the port-matching module being added, one can see significant improvements for all the traces. For the $cisco-trace$, at the flow cache table size of 332, cache miss ratio drops from 8.2% (without proposed modules) to 3.6%(with both port-comparison and port-matching). Considering total system packet flow, the total cache miss ratio is reduced by 50%. For the $lbl-trace$, at the flow cache table size of 175, cache miss ratio drops from 6.8% to 3.6%. And the total cache miss ratio is reduced by 40%. For internet backbone trace $fixwest-trace$, at the flow cache table size of 3755, cache miss ratio is down from 17.9% to 7.3%. Because of the huge amount of one time packets, total cache miss ratio doesn't change much.

In today's communication networks as has also been observed in [10], most of the congestions which we have seen today are taken place at the servers or some bottleneck links rather than high speed links. This phenomenon suggests that in the future high speed networks, high packet rate will be mainly contributed by long-lived flows composed of a large number of packets, rather than large volume of flows which are short-lived. These long-lived flows are mostly generated by a few applications which call for high volume data transfer. Internet-2 [9] currently under development is a good example. With OC-48 link speed deployed across the nation, it becomes one of the Internet-2's primary goal to encourage research Institutions and Universities to develop high volume data applications to fill up the big pipes.

In summary, in future high speed networks, α values can be expected to be larger than its current values. Hence, high performance can be expected when flow caching mechanism is used in future networks.

VI. Conclusions and Future Work

We have designed and simulated a new flow caching mechanism for next generation access routers and edge devices. First, we use a simple queueing model to characterize the performance of flow caching in terms of cache hit ratio, full packet header filtering speed, and flow cache table lookup speed. We demonstrated that the flow cache table lookup speed is not a major performance constraint for flow caching and it is the cache miss ratio that matters. By making use of layer-4 header information, we proposed to add two filtering modules to greatly enhance the performance of flow caching for layer-4 packet forwarding. The empirical analysis based on both campus and backbone trace simulation demonstrated that the cache miss ratio can be reduced up to 50% with the proposed mechanisms.

Our future research will attempt to achieve the following goals,

1. **Design fast algorithms for flow cache table entry search**

For example, there are many algorithms for the hash key design [8] and the effectiveness of these algorithms are highly dependent on the index pattern. An interesting problem to be explored is to minimize the collision by designing an optimal key algorithm. To further reduce the flow cache table size, port number information can be further explored. For example, for client-server applications such as HTTP, flows from the client to the server are generally short-lived, whereas the server-to-client flows tend to be long-lived. A simple scheme is to send client-to-server packets directly to the filtering module without flow caching.

2. **Design effective flow cache table management algorithms**

There are a variety of ways to optimize the flow cache management. In our previous work on IP switch and MPOA flow cache management design [6], [5], we proposed flow timeout based algorithms. In fact, there are many other means to further optimize the flow cache management such as multi-stage caching and application-based caching.

3. **Explore the performance bounds**

With a well designed set of algorithms, we shall explore the worst-case performance bounds and give algorithm design guidelines to achieve these performance bounds.

References

[1] T. V. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," in *Proceedings of ACM SIGCOMM'98*, 1998.

[2] S. Keshav and R. Sharma, "Issues and Trends in Router Design," *IEEE Communications Magazine*, May, 1998.

[3] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable High Speed IP Routing Lookup," in *Proceedings of ACM SIGCOMM'97*, 1997.

[4] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner, "Router Plugins: A Software Architecture for Next Generation Routers," *IEEE/ACM Transactions on Networking*, 2000

[5] H. Che and S. Q. Li, "MPOA Flow Classification and Design," in *Proceedings of IEEE INFOCOM'99*, 1999.

[6] H. Che, S. Q. Li, and A. Lin, "Adaptive Resource Management for Flow-Based IP/ATM Hybrid Switching Systems," *IEEE/ACM Transactions on Networking*, Vol.6 , No. 5, p. 554, Oct. 1998.

[7] L. Kleinrock, "Queueing Systems," Volume I, by John Wiley & Sons, Inc. 1975.

[8] R. Jain, "A Comparison of Hashing Schemes for Address Lookup in Computer Networks," *IEEE Transactions on Communications*, Vol. 40, No. 3, p. 1570, Oct. 1992.

[9] *http://www.internet2.edu*

[10] A. Odlyzko, "The Economics of the Internet: Utility, Utilization, Pricing, and Quality of Service," *http://www.research.att.com/~amo.*

[11] M. Waldvogel, A. Brodnik, S. Carlsson, and S. Pink, "Small Forwarding Tables for Fast Routing Lookup," in *Proceedings of ACM SIGCOMM'97*, 1997.