COMCOM2107

Computer Communications 00 (2002) 000–000

computer communications

www.elsevier.com/locate/comcom

A flow caching mechanism for fast packet forwarding

Ye Tung*, Hao Che

Department of Electrical Engineering, State College, Pennsylvania State University, PA 16803, USA

Received 19 April 2001; accepted 12 November 2001

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 the flow caching mechanisms for fast layer-4 packet forwarding. We show by model analysis that flow caching performance is not very sensitive to the flow cache table lookup speed but it is sensitive to the cache hit ratio. By making use of the available layer-4 information, we introduce two filtering modules to enhance the cache hit ratio. We demonstrate, by real trace simulation, that by adding these two filtering modules, the cache miss ratio can be reduced by up to 50% and the full header filtering speed reduced by up to five-fold. The proposed flow caching mechanism is potentially useful for accessing routers and edge devices where costs are at a premium and where software based filtering modules are dynamically generated. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Packet forwarding; Layer-4; Filtering modules

1. Introduction

The state-of-the-art access or edge routers need to be able to provide functionalities, such as layer-4 or even layer-5 forwarding, as well as firewall/security checks at OC-12, OC-48, or even higher rate. With these functionalities being added for packet processing, a challenging issue concerns how to achieve fast packet filtering at wire-speed and low cost.

There are four 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, and the rest of the packets of the flow are cut-through switched through flow cache entry lookups.

The other three approaches allow packet-by-packet full header filtering and table lookup using the state-of-the-art software, hardware or firmware based fast packet filtering algorithms. The software based approaches existing today cannot live up to Gigabit wire-speed forwarding (requiring about 8 Mpps (Mega packets per second) for 40-byte size packets at OC-48 rate), e.g. Refs. [5–7].

An approach proposed by Lucent [1] describes a scheme optimized for implementation in hardware. It employs bit-level parallelism to match five header fields concurrently. The scheme is reported to support up to 512 filtering rules, classifying 1 Mpps with an FPGA device and five 1MB SRAM, which still falls short of Gigabit rate. The other proposal [7] can also be implemented in hardware. It takes advantage of the redundancy existing in the current Internet Service Provider's packet classifiers and designs a multistage algorithm, which allows fast packet classifications. The scheme is reported to classify 30 Mpps in pipelined hardware. However, this approach works well only when the rule changes are infrequent and it is optimized for packet classification in pure connectionless IP networks, not for other networks, e.g. Multiprotocol label Switching networks.

The most popular approach in industry today is the firmware based approach. It relies on an embedded processor to microcode a packet filtering engine, which is based on a ternary CAM for key search and a SRAM for holding the forwarding information. The search keys are stored in the CAM array. Given a packet header to classify, the CAM performs a comparison against all of its entries in parallel, and a priority encoder selects the first matching search key, which further points to the content of an entry in SRAM. This approach is straightforward and each search key match is performed in one clock cycle. However, ternary CAM is generally very expensive and it consumes a significant amount of power. Also, microcode budget is limited which may not enable wire-speed forwarding in the presence of a large number of filtering rules.

* Corresponding author.
E-mail address: yxt7@psu.edu (Y. Tung).

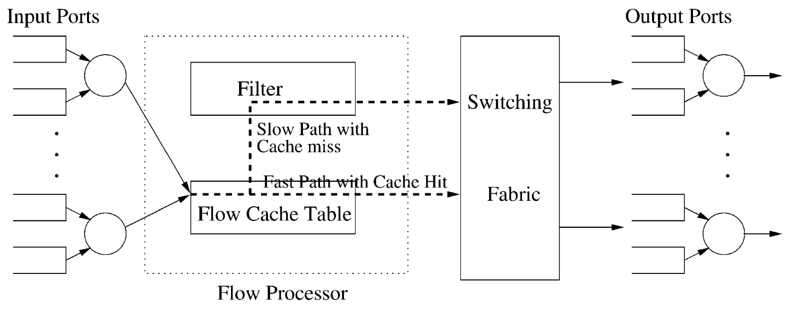

Fig. 1. Schematic functional modules of a router.

Due to relatively small number of concurrently active flows at the access or edge nodes compared to core routers, flow caching mechanism is a viable alternative to the earlier approaches for fast packet filtering with relatively low cost for access or edge routers. In a value-added software architecture for next generation access routers in Ref. [2], flow caching mechanism was successfully used to achieve a three-fold packet forwarding speedup compared with the forwarding speed of the best-effort kernal.

Flow caching is an attractive solution for fast packet filtering at access or edge routers 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, small cache table lookup and management algorithms can be efficiently implemented due to relatively small number of active flows to be handled at the edge. Third, 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 of 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 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 of 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 access/edge routers. First, based on a simple queuing 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 reduce cache miss ratio by up to 50%, resulting in at least a five-fold performance gain.

The following 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 queuing system. The results are presented and their implications on the effective router design are discussed. In Section 4, two mechanisms to enhance 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.

2. Problem formulation

Fig. 1 shows a schematic diagram for the major components of a value-added router with flow caching. The system is composed of five 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* and *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 fast path forwarding. If no match is found, a cache miss occurs and the packet header

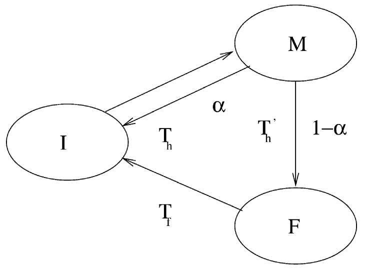

Fig. 2. A finite state machine for filtering system with flow caching.

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.

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 an average, a cache hit occurs with probability α and a cache miss occurs with probability $(1-\alpha)$, called cache hit ratio and cache miss ratio, respectively. For a packet with a cache miss, the system will stay at 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 $T_h \ll T_f + T'_h$ or $T_h \ll T_f$ and α close to 1.

3. Model analysis

First, we consider the case without flow caching. We model it as a simple M/M/1 queuing system with Poisson packet arrival rate and exponential service rate for packet filtering. Denote the average packet arrival rate as λ and the average filtering rate as μ^0. The performance is measured in terms of the tail distribution of the response time, i.e.

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

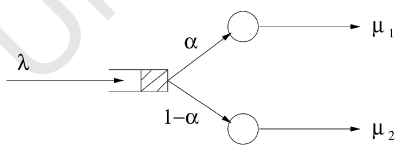

Fig. 3. M/H2/1 queuing model for filtering with flow caching.

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 [4]:

$$P_T(t > \tau) = \exp(-\mu^0(1 - \mu^0/\lambda)\tau). \tag{2}$$

A reasonable assumption is to choose δ so that when $\mu^0 = 1/\tau$ (i.e. a packet is processed with little queuing delay), the equality in Eq. (1) holds. For instance, if we set $\lambda = 1$ Mpps and $\mu^0 = 3.3$ Mpps, we find that $\delta = 1.0$.

Now, let us model flow caching based filtering by an M/H2/1 queuing 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

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

Using the same parameter setting, i.e. $\lambda = 1$ Mpps, and $\delta = 1$, the triplet (μ_1, μ_2, α) can be calculated by substituting Eq. (4) in Eq. (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^0. We first observe that the curves converge quickly to an asymptotic one as μ_1 increases. The flow cache table lookup rate faster than $\mu_1 = 2\mu^0$ helps very little in terms of reducing μ_2 or full header filtering rate for any α values.

For $\alpha = 0$, $\mu_2 \approx \mu^0 = 3.3$ Mpps is expected. When α increases, μ_2 decreases pretty fast. At $\alpha = 0.9$ and $\mu_1 = 2\mu^0$, $\mu_2 = 0.8 \times 10^6$. Let $T'_h = T_h$, we found that $T_f = 1.27$ μs, or the full header filtering rate, of 0.7 Mpps is about five times smaller than the required rate, $\mu^0 = 3.3$ Mpps for

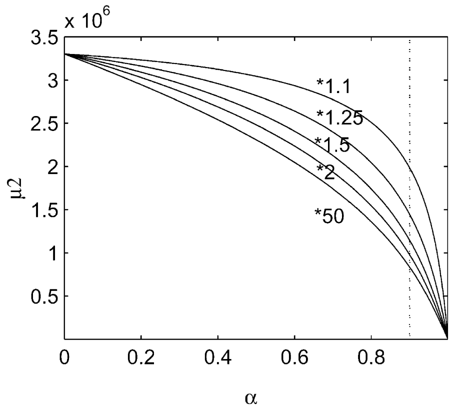

Fig. 4. μ_2 vs. α for various $\mu_1 = \mu^0 m$ ($m = 1.1, 1.25, 1.5, 2, 5, 50$) at wire-speed.

packet-by-packet full header filtering. Clearly, if $\alpha \geq 0.9$ and $\mu_1 \geq 2\mu^0$ are achieved, the flow caching mechanism becomes very effective.

The above analysis shows that *the flow cache table lookup rate does not have to be much faster than the full header filtering rate and it is the cache hit probability that matters*. This has two important implications. First, the speed of flow cache search algorithm is not critical for flow caching. Typically, a cache table entry search takes several clock cycles to finish using standard flow hashing algorithm whereas a full header filtering takes about 20 clock cycles [2]. Second, the key is to minimize the cache miss ratio. In Section 4, we shall design two schemes with low processing costs to greatly reduce the cache miss ratio.

4. 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.

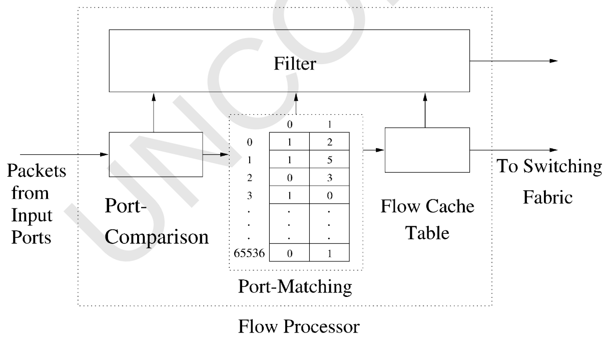

Fig. 5. A new flow caching model.

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 and 1023 (with some exceptions), known as the well-known port numbers, and the other port number randomly assigned between 1024 and 65,536, 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 existing in the cached flows 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 onto 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 onto 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 $65{,}537 \times 2$ table as shown in Fig. 5. The kth row contains the information for the port number k. The kth entry of the second column 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 number is zero, there must be no flow entry in the flow cache table that matches with the arrived packet and the packet header to be passed onto the filtering module for full header filtering. Otherwise, the packet header is forwarded to the flow caching module for flow cache entry search as shown in Fig. 5 random assignment of unknown port numbers.

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

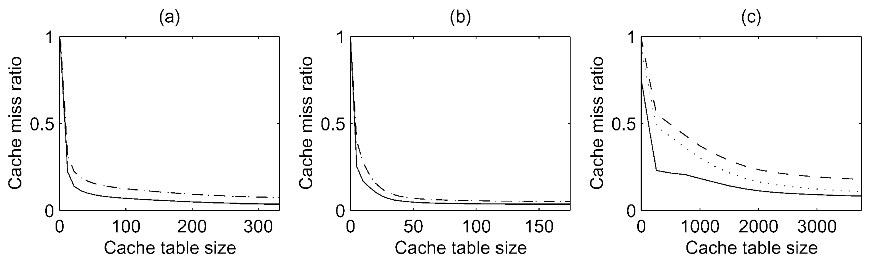

Fig. 6. Cache miss ratio vs. cache table size ('- -' for traditional flow caching, '…' for flow caching with port-comparison, solid line for flow caching with port-comparison and port-matching; (a) cisco-trace, (b) lbl-trace, and (c) fixwest-trace.

the identification of end-to-end applications for both full header filtering and flow cache table lookup. 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 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.

5. 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 min trace collected from a 100-BT campus network at Cisco Systems Inc. on 4 March, 1997. The lbl-trace is a 16 min trace collected from a 100-BT at Lawrence Berkeley Laboratory (LBL) on 14 July, 1997. The fixwest-trace is a 20 min trace collected from the FDDI Internet backbone at FIXWEST on 21 October, 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 Ref. [3]. 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,

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

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 $\rho(n)$,

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

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

The cache miss ratio $(1-\alpha)$ 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. However, 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 a 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. The cache miss ratio drops from 35% for the backbone trace to over 50% for cisco-trace. The 10% cache miss ratio requirement derived in Section 4 can be easily met, at the flow cache table sizes of 35 for cisco-trace, 22 for lbl-trace, and 1855 for fixwest-trace, respectively. For the two campus traces, α values are well above 0.9 even at rather small flow cache table sizes. In fact, α reaches 96.4% at the flow cache table size of 332 and 175 for cisco-trace and lbl-trace, respectively. For internet backbone trace fixwest-trace, α reaches 93.7% at flow cache table size of 3755.

6. Conclusions

We have designed and simulated a new flow caching mechanism for next generation access routers and edge devices. First, we use a simple queuing 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 simulations demonstrated that the cache miss ratio can be reduced by up to 50% with the proposed mechanisms.

References

[1] T.V. Lakshman, D. Stiliadis, High-speed policy-based packet forwarding using efficient multidimensional range matching, Proceedings of ACM SIGCOMM'98, September, 1998.
[2] D. Decasper, Z. Dittia, G. Parulkar, B. Plattner, Router plugins: a software architecture for next generation routers, Proceedings of ACM SIGCOMM'98, September, 1998.
[3] H. Che, S.Q. Li, MPOA flow classification and design, Proceedings of IEEE INFOCOM'99.
[4] L. Kleinrock, Queuing Systems, vol. I, Wiley, New York, 1975.
[5] M. Waldvogel, A. Brodnik, S. Carlsson, S. Pink, Small forwarding tables for fast routing lookup, Proceedings of ACM SIGCOMM'97, September, 1997.
[6] M. Waldvogel, G. Varghese, J. Turner, B. Plattner, Scalable high speed IP routing lookup, Proceedings of ACM SIGCOMM'97, September, 1997.
[7] P. Gupta, N. McKeown, Packet classification on multiple fields, Proceedings of ACM SIGCOMM'99, August, 1999.