A Distributed TCAM Coprocessor Architecture for Integrated Longest Prefix Matching, Policy Filtering, and Content Filtering

Zhiping Cai, *Member*, *IEEE*, Zhijun Wang, Kai Zheng, *Senior Member*, *IEEE*, and Jiannong Cao, *Senior Member*, *IEEE*

Abstract—Longest Prefix Matching (LPM), Policy Filtering (PF), and Content Filtering (CF) are three important tasks for Internet nowadays. It is both technologically and economically important to develop integrated solutions to the effective execution of the three tasks. To this end, in this paper, we propose a distributed Ternary Content Addressable Memory (TCAM) coprocessor architecture. The integrated solution exploits the complementary lookup load and storage load requirements of the three tasks to balance the lookup load and storage load among the TCAMs. A prefix filtering-based CF algorithm is designed to reduce the lookup load and a novel cache system is developed to dynamically handle the lookups from overloaded TCAMs. Simulations based on real-world traffic traces show that the proposed solution can perform all three tasks given a 10 Gbps line rate using only the resources required to perform just the CF task given a 10 Gbps line rate.

Index Terms—Longest prefix matching, policy filtering, content filtering, intrusion detection

1 Introduction

The survivability of the future Internet largely depends on whether it will be able to successfully address both security and performance issues. The Internet becomes more and more vulnerable due to the fast spreading of worms and other malicious attacks. It is under great stress to meet the ever-growing application demands while having to sustain 10 gigabit or even higher forwarding performance [1].

Traditional approaches to guarantee both security and high forwarding performance are in general complex and costly. For example, network intrusion detection systems are typically implemented by using specialized hardware (e.g., dedicated chips or separate boxes) for Content Filtering (CF), rather than standard components in an Internet router, adding complexity and integration/management costs. Packet classification has long been identified as one of the most critical and time-consuming data path functions, creating potential bottlenecks for high-speed packet forwarding.

To remove the potential bottlenecks, various algorithmic and hardware approaches have been developed, attempting to meet the targeted performance for different single packet forwarding or packet classification tasks, such as Longest Prefix Matching (LPM) or Policy Filtering (PF). However, implementing different solutions to execution of the multiple tasks is rather costly and sometimes even infeasible due to various resource constraints. Hence, it is desirable to develop integrated solutions for LPM, PF, and CF, matching 10 gigabit line rate or even higher.

It is a promising approach to integrate packet forwarding and packet classification tasks with Ternary Content Addressable Memory (TCAM). A TCAM-based solution is generally applied to different packet classification tasks. It allows parallel rule matching against all the rules in a rule table, offering the highest possible packet classification performance. As a result, TCAM coprocessors are widely used in the industry to perform multiple packet forwarding and classification tasks, which includes LPM [2], [3], [4], [5], [6], PF [7], [8], [9], and CF [10], [11], [12], [13]. However, the existing TCAM-based solutions have some drawbacks. On one hand, the solutions with a single-TCAM coprocessor cannot keep up with multigigabit line rate when performing multiple packet classification tasks. On the other hand, the existing distributed TCAM architectures are designed for single packet forwarding or classification task, and the expensive hardware hasn't been utilized efficiently to implement multiple packet classification tasks.

This paper aims at developing a distributed TCAM architecture to enable fast and integrated LPM, PF, and CF to match 10 gigabit line rates. Note that there are significant requirement diversities in terms of both lookup load and storage load among LPM, PF, and CF. For example, in the *de facto* CF engine Snort [14], there are about 5,000 distinct rules which typically require less than 1 Mbits TCAM storage, but it requires up to 1.25G lookups per second for

- *Z. Cai is with the Department of Network Engineering, School of Computer, National University of Defense Technology, Changsha, Hunan 410073, P.R. China. E-mail: zpcai@nudt.edu.cn.*
- *Z. Wang and J. Cao are with the Department of Computing, Hong Kong Polytechnic University, Mong Man Wai Building, Hung Hom, Kowloon, Hong Kong. E-mail: {cszjwang, csjcao}@comp.polyu.edu.hk.*
- *K. Zheng is with the System Research Group, IBM China Research Laboratory, Bldg 19 (the Diamond Building), Zhongguancun Software Park, No. 8 Dongbeiwang West Road, Haidian District, Beijing 100193, P.R. China. E-mail: zhengkai@cn.ibm.com.*

Manuscript received 11 Mar. 2011; revised 29 Nov. 2011; accepted 6 Dec. 2011; published online 22 Dec. 2011.
Recommended for acceptance by A. Shvartsman.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-03-0169.
Digital Object Identifier no. 10.1109/TC.2011.255.

10 Gbps wire-speed inspection. In a typically firewall (i.e., PF engine), hundreds of thousands of (5-tuple) rules are usually required, occupying tens of Mbits TCAM storage; only less than 25M lookups per second (note that only packet headers will be inspected in the PF task) is needed. For LPM, it requires only one TCAM lookup per packet, whereas the size of a prefix table is in general at least one order of magnitude larger than that of a rule table in PF.

Without an integrated solution, in order to match 10 Gbps wire speed, 13 TCAMs is needed. Each is capable for 100 *million lookups per second* (MLPS) [11], and several (depending on number of rules and IP prefixes)10 Mbits TCAMs for PF and LPM, respectively. But in an integrated solution, 13 TCAMs with 100 MLPS each and 2 Mbits storage can provide lookup bandwidth and storage space for all the LPM, PF, and CF tasks. Obviously, it is cost efficient to integrate these tasks together.

Our proposed solution exploits the complementary characteristics of the three tasks, namely LPM, PF, and CF, and executes them simultaneously. The prefixes, policy rules, and intrusion patterns are distributed to a set of TCAMs with balanced lookup load and storage load. We developed a prefix filtering algorithm to reduce the lookup load for CF, and a novel prefix-group (PG)-based cache scheme to dynamically handle CF lookups from overloaded TCAMs. Simulation results show that the integrated solution can significantly reduce the overall cost and still maintain a reasonable throughput.

The rest of the paper is organized as follows: Section 2 describes some related works. The details of the proposed integrated solution are given in Section 3. Section 4 presents the performance evaluation of the proposed solutions. Finally, the conclusions are drawn in Section 5.

2 Related Work

Significant research efforts have been made on the development of algorithms for efficiently using TCAM coprocessors for high-speed packet classification such as LPM and PF, and intrusion detection systems. TCAM is used to perform packet classification at wire speed has become the *de facto* technology for high-speed routers [15].

The research issues on TCAM coprocessor include power consumption [16], [17], throughput [18], database update [10], storage efficiency [2], [7], [9], [19], [20], [21], and new applications. Significant research efforts have been made to the optimize TCAM storage efficiency. Recently, multi-matching classification and policy table compacting problems were addressed in [2], [3] and [11]. Zheng et al. [6] proposed a distributed TCAM coprocessor architecture for LPM that matches OC-768 line rate. Spitznagel et al. [22] extended the LPM to general packet classification by organizing the TCAM as a two-level hierarchy. It can deliver high performance for even large filter sets. Faezipour and Nourani [3] designed a TCAM that can find all or multiple highest matches in a packet filter set to reduce the multimatch packet classification time.

Except LPM and policy filtering, another promising application of TCAM coprocessors is high-speed signature matching for intrusion detection [23]. The existing intrusion detection systems, such as the software-based solutions cannot keep up with gigabit line rates [23], [24]; those which are based on ASIC and FPGA [25], [26], [27] may incur large processing latencies for databases with a large number of signature patterns.

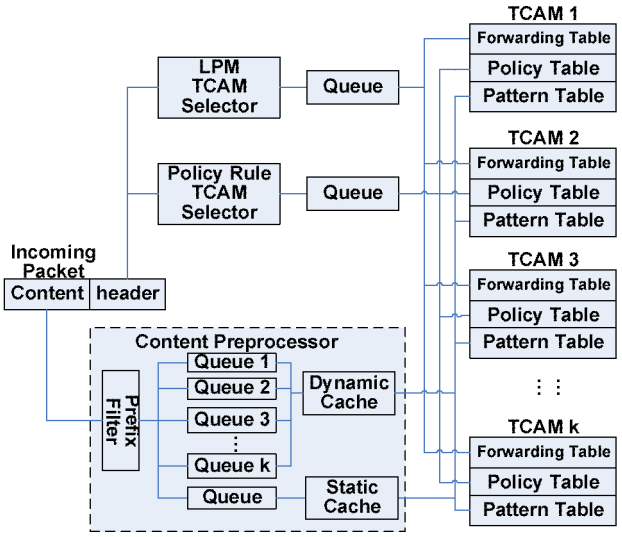


Fig. 1. A distributed TCAM coprocessor architecture for integrated three tasks.

The Bloom filter is a kind of space efficient algorithm for pattern matching. It has also been proposed for intrusion detection in [28], [29], and [30]. However, due to the variable length of intrusion patterns, a large number of Bloom filters with different length may need to be constructed, thus making them unscalable.

Yu et al. [13] proposed a gigabit rate pattern matching system with a single-TCAM coprocessor. The system can handle up to 2 Gbps line rates. Our work builds upon their work and addresses one shortcoming of their system, the need to lookup every W-byte (W is the TCAM slot width) string in a TCAM.

Weinsberg et al. [31] designed a Rotating TCAM (called RTCAM). The RTCAM algorithm enables the network intrusion prevention system to work at several gigabit line speed. Recently, Meiners et al. [32] proposed a hardware-based regular expression matching approach that uses TCAMs for intrusion detection. The method can achieve a potential high speed. To the best of our knowledge, there is no integrated solution allowing both multitask packet classification and intrusion detection simultaneously.

3 The Integrated Solution

3.1 Distributed TCAM Coprocessor Architecture

The proposed distributed TCAM coprocessor architecture for integrated LPM, PF, and CF is shown in Fig. 1. It includes multiple TCAMs, an *LPM TCAM Selector* (LPMS), a *Policy rule TCAM Selector* (PRS) and a *Content Preprocessor* (CP) which is composed of a *PRefix Filter* (PRF), caches and queues. Each TCAM has a forwarding table, a policy rule table, and a pattern table. A CP includes a PRF, an on-chip *Static Pattern Cache* (SPC), and an on-chip *Dynamic Pattern Cache* (DPC). The cache can be implemented by using small and fast on-chip TCAM.

In this paper, we present a general architecture and use the prefix filter as an example scheme for CF, any other alternative schemes for CF, PF, and/or LPM such as the schemes proposed in [32] can be incorporated into the proposed architecture.

When an incoming packet arrives, the 5-tuple in the header is parsed and sent to a PRS to select the corresponding TCAM for its PF. At the same time, the packet header is also parsed and sent to an LPMS to select the corresponding TCAM for its LPM. Meanwhile, the packet content is passed to a CP. 2-byte is inspected (prefix filtering) at a time, and the returning index indicates whether the byte stream needs to be looked up in a TCAM or not. If not, shift one byte and do the prefix lookup again, otherwise, the returned index shows which TCAM or cache should be looked up against for the bit stream starting at the 2-byte.

3.2 TCAM-Based Solutions for Content Matcher

Two challenges have been identified in the distributed TCAM coprocessor architecture design: 1) how to effectively reduce the number of TCAM lookups; and 2) how to distribute and balance the lookup load and storage load among TCAMs. In the following sections, we will design a CP and a set of rule/pattern distributing algorithms to handle the two challenges.

3.2.1 Pattern Prefix Group

We adopt the method proposed in [13] to arrange the patterns in the TCAMs. For the case of W-byte slot, each pattern is partitioned into multiple W-byte subpatterns (if the pattern has less than W bytes, just expand wildcard bytes to the end). For example, assume that there are three patterns $abcdefg$, $de1$, and $ababab$. To put the patterns into a TCAM with 4-byte slot, the patterns are cut to subpatterns (called prefix or suffix): $abcd$, $defg$, $de1*$, and $abab$. Here, $*$ represents a wildcard byte. The 2-byte prefix of the four subpatterns are $ab*$, $de*$, $de*$ and $ab*$, respectively, only two of which are distinct prefixes, i.e., ab and de. In the remaining part of the paper, we use pattern to represent both pattern and subpattern for the sake of simplicity. When a pattern is longer than W-byte, multiple TCAM lookups are needed. The process is as follows: when a prefix pattern is matched at the ith position of the packet, it is recorded in memory. When a suffix pattern is matched at position $i + j(0 < j \leq W)$ later, check if the matched prefix and suffix form a long pattern.

Definition 1 (Prefix Group and PG IDentity (PGID)). *A Prefix Group is composed of a set of patterns which have the same 2-byte-long prefix; the common 2-byte prefix of the patterns is denoted as the PG IDentity.*

In the previous example, the patterns are grouped into two prefix groups: the PG with PGID ab including patterns $abcd$ and $abab$, and the PG with PGID de including patterns $defg$ and $de1*$.

For an incoming string to be inspected, it may match some patterns in a PG only if its 2-byte prefix matches the PGID; if no PGID is matched, no further lookup will be needed. If a prefix pattern is matched at the ith position of the packet, it is recorded in the pattern hit list. If a suffix pattern is matched at position $i + j(0 < j \leq W)$ later, check if the matched prefix and suffix form a long pattern. If no more possible suffix can be merged for a previous matched pattern, the pattern is deleted from the pattern hit list.

In the proposed design, a PRF is composed of a direct accessible memory such as SRAM. All the patterns from a PG should be stored in the same TCAM. The PGID is used as an index to address the corresponding TCAM. For 2-byte PRF, only 64 KB memory is needed. The entry value of an index is set to 0 when the 2-byte string does not match a PGID of any PG implemented in the TCAMs or caches; otherwise, the value is set to the identity of the TCAM or the cache in which the PG is stored. For example, suppose there are only two PGIDs ab and de, and PG with PGID ab is stored in TCAM #1 and PG with PGID de is stored in TCAM #2, then in the PRF, the entry of ab and de will be set to be 1 and 2, respectively, and all the other entries will be set to be 0.

3.2.2 Patterns Distributing Schemes

The 2-byte prefixes are used as the PGIDs. All patterns with the same PGID will be grouped together. Then, all the PGs are distributed to K TCAMs with balanced lookup load and storage load. For clarity, we first describe the mathematical notation of the pattern distributing problem.

Let S be the set of PGs and N be the number of PGs; Q_k be the set of PGs placed in TCAM $\#k(k = 1, 2, \ldots, K)$; $L[i](i = 1, \ldots, N)$ be the lookup load of $PG[i]$; $S[j](j = 1, \ldots, N)$ be the number of patterns in $PG[j]$; $LL[k]$ be the lookup load that is assigned to TCAM $\#k$, namely, $LL[k] = \sum_{PG[i] \in Q_k} L[i]$. $SL[k]$ be the storage load that is assigned to TCAM $\#k$, namely, $SL[k] = \sum_{PG[i] \in Q_k} S[i]$.

The optimization problem for patterns distributing schemes is given by:

To find a K-division $\{Q_k, k = 1, \ldots, N\}$ of PGs that:
Minimize: $Max_{k=1,\ldots,K} LL[k]$ and $Max_{k=1,\ldots,K} SL[k]$
Subject To: $Q_k \subseteq S$, $\bigcup_{k=1,\ldots,K} Q_k = S$.

Note that the distribution scheme is a two-objective optimization problem, i.e., balancing the lookup load and the storage load. Consider each PG as an object and each TCAM as a knapsack, the pattern distribution problem is actually a variance of Knapsack problem, which can be proven to be NP-hard. Hence, we simplify the original two-objective optimization problem into a single-objective one by varying the priority of the two objectives. Then, a heuristic algorithm is developed to solve the problem.

Algorithm 1 depicts the pseudocode of the PGs distribution algorithm. The number of TCAM chips, K, is an integer number, which should be no less than $\lceil (S_{CF} + S_{PF} + S_{LPM})/S_{TCAM} \rceil$, where S_{CF}, S_{PF}, and S_{LPM} are the size of patterns, policy rule sets and IP prefixes, respectively, while S_{TCAM} is the storage space of each TCAM. This will guarantee that the requested storage space for integrated tasks is met. Also note that a larger K value may result in system design flexibility for the further application, but also in higher cost.

Algorithm 1. The PGs Distribution Algorithm
Input:
The number of PGs, N;
The number of TCAMs, K;
The cost function of PG g_i, $Cost(i)$;

Output:
The set of PGs distributed to the kth TCAM, Q_k;
The sum of cost function in the kth TCAM, $G[k]$;
1: **for** $k = 1$ to K **do**
2: $Q_k = 0; G[k] = 0$;
3: **end for**
4: Sort $g_i\{i = 1, \ldots, N\}$ in decreasing order of $Cost(i)$;
5: **for** $i = 1$ to N **do**
6: sort $Q_k\{k = 1, \ldots, N\}$ in increasing order of $G[k]$;
7: **for** $k = 1$ to K **do**
8: $G[k] = G[k] + Cost(i)$;
9: $Q_k = Q_k \bigcup g_i$;
10: **end for**
11: **end for**
12: **return** $Q_k(k = 1, \ldots, K)$ and $G[k](k = 1, \ldots, K)$

The algorithm applies the greedy heuristic to distribute patterns to TCAMs. The scheme chooses a TCAM with minimum cost to load the next selected PG which has the maximum cost among all the undistributed PGs. The cost function varies with different optimized objectives. By considering the lookup load first, the cost function of a PG sums the lookups of all the patterns in a PG, while by considering the storage load first, the cost function sums the sizes of all the patterns in a PG.

The PG distribution algorithm distributes patterns to TCAMs with balanced lookup load or storage load based on history statistics. However, the statistical data are obtained from long-term history traces. So, it may not balance well for burst traffic. Hence, an on-chip *Dynamic Pattern Cache* is introduced to handle the lookup load from overloaded TCAMs.

3.2.3 Pattern Cache

Although the PG distribution algorithm can evenly allocate the average TCAM lookup load among the TCAMs, the short-term lookup load may be very dynamic and unbalanced. Hence, we design two caches for CP, i.e., *Static Pattern Cache* and *Dynamic Pattern Cache*, to handle burst traffic. The SPC is used to store the long-time popular PGs. For example, the string 00 is the highest occurrence among all the test traces in our experiments. Hence, the PG with PGID "00" is considered as a long-time hot PG and should be placed in the SPC. Note that the PGs in a SPC are not absolutely static, which have a considerable slow updating rate.

DPC is designed to handle the short-time burst lookups in a TCAM. Our experiments show that the TCAM lookup overflow cannot be avoided by increasing the number of TCAMs only. This is due to that some patterns may have rather high-frequency appearances in a very short time, but it may not be very popular in the long time. Such transient burst patterns may lead to a TCAM lookup load overflow. Higher performance cache is needed to handle with the burst patterns.

The traditional cache scheme may not be appropriate here to reduce the lookup load. For example, the cache scheme designed in [10] for LPM has a high cache hit ratio and can significantly reduce the number of TCAM lookups. But that method cannot be used here due to the rather low hit ratio in the cache for CF. For CF, the normal traffic has rather low probability to match an attacking pattern. Even for dirty packets with attacking patterns, they could only match a few attacking patterns, hence the pattern matching ratio may not be high either.

In our solution, for a byte string, we first do a 2-byte prefix filtering in a PRF. If the entry value is 0, we shift one byte for next string prefix lookup. Otherwise, the W-byte streaming is put to the queue indicated by the returned value. For each queue, we set two thresholds; if the number of bytes in a queue exceeds the first threshold, we count the frequency of the prefixes in the queue, and if the queue is over the second threshold (called "queue over limit"), the most popular PGs in the queue are copied to the DPC, the index in the PRF will be updated accordingly. The incoming string with matched PGID will only be looked up in the DPC. But this scheme needs to update the PRF whenever the cache is updated; moreover, updating PRF needs to suspend PRF lookup, thus significantly reducing the PRF lookup throughput.

We use another way to eliminate the update in the PRF. In our method, when a TCAM has over limit queue, all the byte strings sent to this TCAM are first looked at the DPC. If the string does not match any pattern, it will be looked up in the TCAM. As mentioned earlier, the hit ratio of matching an exact pattern is considerable low; hence, the cache filtering effect is low. In order to overcome the problem of low hit ratio, we introduce a delegate entry into each prefix group. The delegate entry of a group is a W-byte string starting with the first 2 prefix bytes and followed by $W - 2$ wildcard bytes *. The delegate entry has the lowest matching priority in a prefix group. If a W-byte string matches a delegate entry, it means that the byte string has been matched against all the patterns in the prefix group, and no matching pattern is found, therefore no more lookup is needed. When the queue length is below the first threshold, the bit string is directly looked up in the TCAM, the DPC is left for other over limited TCAMs.

Let us look at an example. Supposing a PG with PGID ab has two patterns ($ab12$ and $ab34$), the delegate entry $ab**$ is introduced. A string $ab12abcdabef$ comes, the 4-byte string in this PG: $ab12$, $abcd$ and $abef$, are filtered out by the DPC, and no more TCAM lookups are needed. Here, $ab12$ matches the exact pattern, $abcd$ and $abef$ match the delegated entry $ab**$. Without the delegated entry, the strings $abcd$ and $abef$ need to be looked up in the TCAM after the DPC lookup.

A lock-free TCAM update algorithm [18] is used for the DPC updating. In this updating scheme, the DPC updating is through a different interface from the data path interface so that the lookup process does not need to be intercepted during the updating.

The PGs in the DPC is updated when a newly counted PG in the over limited queue becomes more popular than some PGs in the DPC. The DPC is shared by all the TCAMs. This means when more than one queue are overflowed, the most popular PGs coming from these TCAMs can be stored in the DPC, and all the byte strings going to these TCAMs are first looked up in the DPC. Although the DPC is shared by all the TCAMs, it is only shared by few TCAMs at a time. This happens due to the fact that the total number of lookups in a unit time is upper bounded, when the lookup is significantly increased in one or two TCAMs, the other TCAMs should have reduced lookup load.

Note that the DPC could be extremely tiny in size. According to the analysis of the Snort patterns, the average size of a PG is about four patterns. The cache may hold a few PGs at the same time. This is due to the unbalanced lookup load which usually comes from a few prefixes getting high matches. If many PGIDs get high matching rate, it is with high possibility that these PGs are distributed to different TCAMs; thus, the lookup load can still be balanced. But for a few prefixes that get high matches, the lookup load may be unbalanced among the TCAMs, and loading them to DPC can significantly reduce the lookups in that TCAM.

3.3 Policy Rule Group and IP Prefix Group Distribution

The policy rule sets and the IP prefixes are also needed to be partitioned to different groups so that they can be distributed to different TCAMs.

First, by appropriately selecting the ID bits, a large rule set is partitioned into several Key-ID groups of similar sizes. The key-ID bits can be selected from the destination IP, source IP, and the protocol fields. The full set can be partitioned into 16, 32, or 64 groups with 4, 5, or 6 bits key-ID, respectively. Second, by applying certain load balancing and storage balancing heuristics, the rules (Key-ID groups) are distributed evenly to several TCAM chips. As a result, multiple packet classifications corresponding to different Key-ID groups can be performed simultaneously, which significantly improves the throughput performance without much additional cost.

For grouping the IP prefix, we directly use its left-most bits to be the selector index. For example, if using the first 8-bit of an IP prefix to be the index, the prefixes can be grouped into 256 groups. Applying certain load balancing and storage balancing heuristic algorithms, the grouped IP prefixes are distributed evenly to several TCAM chips.

3.4 Integrated Solution for Longest Prefix Match, Policy Filtering and Content Filtering

The three tasks, LPM, PF, and CF, compete for TCAM lookup resource. They are related on the lookup requests. For traffic with small packets, it needs more LPM and PF lookups but less CF lookups (less content bytes). For line speed R and packets with minimum size 40 bytes (no content), the number of LPM and PF lookups is $R/40$, while no CF lookup is needed. For packets with maximum size M, the number of LPM and PF lookups is R/M, and the number of CF is $(R - R * 40/M)$. For packets with 200 bytes content, the number of CF lookups is 200 times as that of the LPM and PF lookups. Hence, the TCAM lookup loads mainly depend on the CF task. On the other hand, the size of IP prefix tables is in general at least one order of magnitude larger than those of the policy rule sets and intrusion patterns. The IP prefix tables will take a large portion of the TCAM storage space. Hence, the integrated solution must take advantage of such lookup and storage demands.

According to the sizes of patterns, policy rule sets and IP prefixes, the number of requested TCAM chips can be calculated. Then, all the PGs, policy rules, and IP prefixes are distributed to K TCAMs with balanced lookup load and storage load. We first describe the mathematical model of the distributing problem as following: Let S_{CF}, S_{PF}, and S_{LPM} be the set of PGs, policy rules, and IP prefixes, respectively. Q^k_{CF}, Q^k_{PF}, and Q^k_{LPM} be the set of the PGs, policy rules, and IP prefixes placed in TCAM $\#k(k = 1, \ldots, K)$. Let $SL_{CF}[k]$, $SL_{PF}[k]$, and $SL_{LPM}[k]$ be the storage load assigned to TCAM $\#k$ for the PGs, policy rules, and IP prefixes, respectively; $LL_{CF}[k]$, $LL_{PF}[k]$, and $LL_{LPM}[k]$ be the lookup load assigned to TCAM $\#k$ for PGs, policy rules, and IP prefixes, respectively.

The optimization problem to find a K-division of PGs, policy rules, and IP prefixes is stated as below:

Minimize: $Max_{k=1,\ldots,K}(SL_{CF}[k] + SL_{PF}[k] + SL_{LPM}[k])$ and $Max_{k=1,\ldots,K}(LL_{CF}[k] + LL_{PF}[k] + LL_{LPM}[k])$

Subject To:

$$Q^k_{CF} \subseteq S_{CF}, \bigcup_{k=1}^{K} Q^k_{CF} = S_{CF},$$

$$Q^k_{PF} \subseteq S_{PF}, \bigcup_{k=1}^{K} Q^k_{PF} = S_{PF},$$

$$Q^k_{LPM} \subseteq S_{LPM}, \bigcup_{k=1}^{K} Q^k_{LPM} = S_{LPM}.$$

The optimization problem of distributing PGs, policy rules, and IP prefixes is a generalized problem for only distributing PGs. We may duplicate the some popular PGs to every TCAM to balance the lookup load, but the burst lookups from unpopular PGs may still cause TCAM overflow as shown in the simulation later. The optimization problem can be shown to be also NP-hard. We designed three distribution algorithms to solve this optimization problem. The first and second algorithms distribute these tables according to their lookup load and storage load, respectively. The third algorithm takes both lookup load and storage load into account. The lookup load and storage load can be obtained from the history traces or the dynamically measured statistical data.

The pseudocodes of the three algorithms are shown in Algorithm 2 (the LF algorithm), Algorithm 3 (the SF algorithm), and Algorithm 4 (the LS algorithm), respectively. Here, M is the number of PF rule groups, and L is the number of IP prefix groups; $T[k]$, $P[k]$, and $R[k]$ are the sets of PGs, PF rule groups, and IP prefixes distributed to the kth TCAM, respectively; g_i, p_i, and r_i are the ith PG, PF rule group, and IP prefix group, respectively, and $Cost()$ is the cost function.

Algorithm 2. Distribution Algorithm based on the Balance-Lookup-Load-First (**LF**)

1: Set $G[k]$ to be the storage load of the kth TCAM;
2: Call the PGs Distribution Algorithm to get $T[k]$;
3: Sort $m\{m = 1, \ldots, M\}$ in decreasing order of p_m lookup load; If exists multiple p_m being equal, sort these $P[k]$ in increasing order of storage load;
4: **for** $m = 1$ to M **do**
5: Sort $k\{k = 1, \ldots, K\}$ in increasing order of $G[k]$;
6: **for** $k = 1$ to K **do**
7: $G[k] = G[k] + Cost(p_m)$;
8: $P[k] = P[k] \bigcup p_m$;
9: **end for**

10: **end for**
11: Sort $l\{l = 1, \ldots, L\}$ in decreasing order of r_l lookup load; If exists multiple r_l being equal, sort these r_l in increasing order of storage load;
12: **for** $l = 1$ to L **do**
13: Sort $k\{k = 1 \ldots, K\}$ in increasing order of $G[k]$;
14: **for** $k = 1$ to K **do**
15: $G[k] = G[k] + Cost(r_l)$;
16: $R[k] = R[k] \bigcup r_l$;
17: **end for**
18: **end for**
19: **return** $P[k]\{k = 1, \ldots, K\}, T[k]\{k = 1, \ldots, K\}$, $R[k]\{k = 1, \ldots, K\}$.

Algorithm 3. Distribution Algorithm based on the Balance-**S**torage-Load-**F**irst (**SF**)

1: Set $G[k]$ to be the storage load of the kth TCAM;
2: Sort $m\{m = 1, \ldots, M\}$ in decreasing order of p_m storage load; If exists multiple p_m being equal, sort these p_m in increasing order of lookup load;
3: **for** $m = 1$ to M **do**
4: Sort $k\{k = 1, \ldots, K\}$ in increasing order of $G[k]$;
5: **for** $k = 1$ to K **do**
6: $G[k] = G[k] + Cost(p_m)$;
7: $P[k] = P[k] \bigcup p_m$;
8: **end for**
9: **end for**
10: Sort $l\{l = 1, \ldots, L\}$ in decreasing order of r_l storage load; If exists multiple r_l being equal, sort these r_l in increasing order of lookup load;
11: **for** $l = 1$ to L **do**
12: Sort $k\{k = 1 \ldots, K\}$ in increasing order of $G[k]$;
13: **for** $k = 1$ to K **do**
14: $G[k] = G[k] + Cost(r_l)$;
15: $R[k] = R[k] \bigcup r_l$;
16: **end for**
17: **end for**
18: Sort $i\{i = 1, \ldots, N\}$ in decreasing order of g_i storage load; If exists multiple g_i being equal, sort these g_i in increasing order of lookup load;
19: **for** $i = 1$ to N **do**
20: Sort $k\{k = 1, \ldots, K\}$ in increasing order of $G[k]$;
21: **for** $k = 1$ to K **do**
22: $G[k] = G[k] + Cost(g_i)$;
23: $T[k] = T[k] \bigcup g_i$;
24: **end for**
25: **end for**
26: **return** $P[k]\{k = 1, \ldots, K\}, T[k]\{k = 1, \ldots, K\}$, $R[k]\{k = 1, \ldots, K\}$.

The LF algorithm balances the lookup load first, and its cost function counts the lookup load. Only considering the lookup load balance, it distributes CF patterns as a single CF task, similar to the PG distribution algorithm given in the Section 3.2.2.

The SF algorithm balances the storage load first, and its cost function counts the storage load. It distributes the CF patterns, PF rules, and IP Prefixes based on their storage load. According to the both algorithms, the three tasks are considered as a single task, and the patterns (rules, patterns or prefixes) are only distributed as their lookup or storage requests. The algorithms may balance one request (lookup or storage) well, but may perform poorly for the other request.

To well balance both look up load and storage load, we propose the LS algorithm with considering the lookup load of CF, but the storage load of PF and LPM. Due to the high lookup request of CF, the LS algorithm distributes PGs on the basis of their lookup load. On the other hand, the PF and LPM need low lookup load but high storage load; hence, the LS algorithm distributes the policy rules and the IP prefixes on the basis of their storage load. Algorithm 4 shows the LS algorithm.

Algorithm 4. Distribution Algorithm based on both the Balance **L**ookup-load and **S**torage-Load (**LS**)

1: Set $G[k]$ to be the storage load of the kth TCAM;
2: Set cost function $Cost()$ to count the lookup load;
3: Sort $i\{i = 1, \ldots, N\}$ in decreasing order of g_i storage load; If exists multiple g_i being equal, sort these g_i in increasing order of lookup load;
4: **for** $i = 1$ to N **do**
5: Sort $k\{k = 1, \ldots, K\}$ in increasing order of $G[k]$;
6: **for** $k = 1$ to K **do**
7: $G[k] = G[k] + Cost(g_i)$;
8: $T[k] = T[k] \bigcup g_i$;
9: **end for**
10: **end for**
11: Set $G[k]$ to be the storage load of the kth TCAM;
12: Set cost function $Cost()$ to count the storage load;
13: Sort $m\{m = 1, \ldots, M\}$ in decreasing order of p_m storage load; If exists multiple p_m being equal, sort these p_m in increasing order of lookup load;
14: **for** $m = 1$ to M **do**
15: Sort $k\{k = 1 \ldots, K\}$ in increasing order of $G[k]$;
16: **for** $k = 1$ to K **do**
17: $G[k] = G[k] + Cost(p_m)$;
18: $P[k] = P[k] \bigcup p_m$;
19: **end for**
20: **end for**
21: Sort $l\{l = 1, \ldots, L\}$ in decreasing order of r_l storage load; If exists multiple r_l being equal, sort these r_l in increasing order of lookup load;
22: **for** $l = 1$ to L **do**
23: Sort $k\{k = 1, \ldots, K\}$ in increasing order of $G[k]$;
24: **for** $k = 1$ to K **do**
25: $G[k] = G[k] + Cost(r_l)$;
26: $R[k] = R[k] \bigcup r_l$;
27: **end for**
28: **end for**
29: **return** $P[k]\{k = 1, \ldots, K\}, T[k]\{k = 1, \ldots, K\}$, $R[k]\{k = 1, \ldots, K\}$.

4 Performance Evaluation

In this section, we evaluate and analyze the key performance metrics of the proposed integrated solution, including the effects of 2-byte prefix filtering, the pattern cache schemes, and the distribution algorithms. We also use the

TABLE 1
Lookup Distribution among TCAMs for Three Algorithms

	$LF Algorithm$			$SF Algorithm$			$LS Algorithm$		
	Trace 1	Trace 2	Trace 3	Trace 1	Trace 2	Trace 3	Trace 1	Trace 2	Trace 3
TCAM#1	25.0	25.1	25.8	29.4	32.1	30.8	25.1	25.7	26.0
TCAM#2	25.0	24.7	23.8	21.5	21.3	20.9	24.9	25.2	25.6
TCAM#3	25.0	24.9	24.5	22.7	22.4	23.3	24.9	24.0	24.7
TCAM#4	25.0	25.3	25.9	26.4	24.2	25.0	25.0	25.1	23.7

synthetic data traces to analyze the ability of handling attacking packets, which is generated by randomly inserting attacking patterns into the packets.

4.1 Experiment Setup

We use a cycle-accurate system simulation. A cycle time is set to 10^{-10} second, i.e., one second has 10G cycles.When a packet arrives, the destination IP and the five-tuple rule are extracted from the head and are sent to the corresponding TCAM for lookup. The first 2-byte content are looked up in the prefix filter, if no prefix is matched, one byte is shifted. Otherwise, the W-byte string is sent to the corresponding TCAM or cache queue. If the queue is overflowed, a bypass is counted.

We first use the traffic traces from "1999 DARPA Intrusion Detection Evaluation Data Set" provided by MIT [33]. We choose three traces. Each of them has about one millions of packets with contents. The first week and third week of the training data do not contain any attacks. The fifth week data contains 56 types of attacks. For simplicity, the first week (325 Mbytes), the third week (446 MBytes), and the fifth week (468 MBytes) traces are denoted as Traces #1, #2, and #3, respectively. These packets are inserted back-to-back with full line speed without considered their time stamps.

The pattern set from Snort 2.9 [14] (dated July 2011, including 5,038 distinct patterns) is used in the experiments. There are 7,553 8-byte patterns for the Snort pattern set, and 1,563 2-byte prefixes which are obtained from these 8-byte patterns. The policy rules are generated by using ClassBench [34] for evaluation. The route tables are obtained from RouteViews [35] in January 2011.

In the simulations, the proposed solution has a 256 MLPS SPC, a 256 MLPS DPC, and four 100 MLPS TCAMs. Each TCAM has 4 Mbits memory. The data bus is supposed to be 64 bits in width. The SPC is designed to store the most popular prefixes. As the 0x000 and 0x2020 are the most popular prefixes in the statistic data, they are stored in SPC. The others subpatterns are distributed to four TCAMs using the PG distribution algorithm.

In the proposed prototype, the maximum packet process delay is limited to 1 ms and the queue size is set to be 100K search key[1] (i.e., 8-byte strings). The first and second thresholds are set to be 75K and 80K 8-byte strings, respectively.

1. The maximum packet size is assumed to be $L_{max} = 1{,}518$ bytes, i.e., the same as the Ethernet Maximum Transmission Unit. The process delay for a packet with L byte includes: 1) the header processing delay; 2) the prefix matching delay and cache lookup delay, $2*(L - W + 1)/P$, where P is the matching speed for 10 Gbps link rate and $P = 1.25$ Gps; 3) TCAM lookup delay, $(L - W + 1)/S$, where S is the TCAM working frequency; 4) queue delay Q, i.e., the maximum queuing delay of a W-byte waiting in the buffer. Hence, the total delay is $2(L - W + 1)/P + (L - W + 1)/S + Q$. For $L = L_{max} = 1518$, $W = 8$, $P = 1.25G$, and $S = 100M$, and assuming that the maximum total delay budget is 1 ms (considering the constraints of the applications like VoIP), the maximum queue delay allowed is conducted to be $Q = 0.983$ ms. So, the queue size is estimated to be 0.983 ms$/(1/100$ MHz$) \approx 100$K 8-byte slots.

4.2 Effect of Prefix Filtering

Without prefix filtering, the number of TCAM lookups of a trace approximately equals to the number of the content bytes in the trace [10]. With the prefix filter, 61.4, 60.1, and 56.7 percent lookups can be filtered out in Traces #1, #2, and #3, respectively. It indicates that the TCAM lookup requirement can be reduced significantly with prefix filtering, and thus saving the TCAM lookups as well as TCAM power consumption.

4.3 Effect of the Distribution Algorithms

Now, we generate 100K (i.e., 10.4 Mbits) policy rules using ClassBench [34] for evaluation. Six bits (i.e., 64 policy rue groups) selected from the first, third and fifth bits of destination IP, the second and fourth bit of source IP, the fifth bit of the protocol fields, are used as the identity to locate the TCAM in which the rule is stored. The IP prefixes were obtained from RouteViews in January 2011. It includes about 133,761 (4.28 Mbits) route entries. The first byte is selected as the index for IP prefix distribution, i.e., there are 256 IP prefix groups. The lookup load distribution is measured with Trace #1.

We distribute the PGs, policy rules, and IP prefixes to four TCAMs, respectively, according to the three algorithms described on Section 3. The lookup load and storage load distributions are shown in Tables 1 and 2, respectively. Although the lookup load distribution is based on Trace #1, very similar distribution can be observed in other two traces. The LF algorithm achieves the best lookup load balance but the worst storage load balance. The SF algorithm achieves the best storage load distribution but results in the worst lookup distribution. The SF algorithm considering both the balance lookup load and storage load can achieve both the close to best lookup load and storage load distribution. It is due to most of lookup load coming from CF task, and most of storage load coming from the policy rules and IP prefixes. Hence, it can achieve good balance for both lookup load and storage load among the TCAMs.

4.4 Effect of Static Cache and Dynamic Cache

In this case, we first examine the cache effect and then evaluate the performance of the whole system. We set the proposed prototype with four TCAMs, a SPC and a DPC

TABLE 2
Storage Distribution among TCAMs for Three Algorithms

	$LF Algorithm$	$SF Algorithm$	$LS Algorithm$
TCAM#1	27.0%	25.0%	25.1%
TCAM#2	23.8%	25.0%	24.9%
TCAM#3	22.8%	25.0%	24.9%
TCAM#4	26.4%	25.0%	25.1%

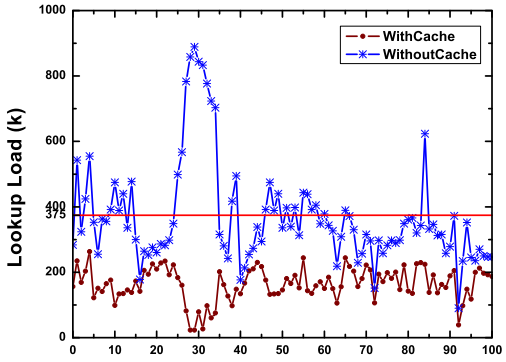


Fig. 2. Time-varying lookup load distribution for TCAM #1.

(the total capacity is $4 * 100M + 2 * 256M = 912M$) and a counterpart system with 10 TCAMs but without SPC and DPC (called Pure TCAM system). After prefix filtering of CF, the average TCAM lookup load is less than the lookup capacity in the both two systems. We use Trace #1 as the input data, which has 468 Mbytes, transmitted with 10 Gbps wire speed, or 1.25 GBytes per second. The Trace lasts for $468M/1.25G = 0.3744$ seconds. We count the number of TCAM lookups in every 3.744 milliseconds.

Figs. 2 and 3 show the lookup load distribution of TCAMs #1 and #2 in every 3.744 millisecond intervals, respectively. Note that a TCAM lookup capacity in such a period is $100\text{M} \times 0.003744 = 374{,}400$ times. From Figs. 2 and 3, we know that in both TCAMs #1 and #2, the lookup load is always less than the capacity (red solid line) at all the time for the system with caches, but it is over the capacity at sometime for both TCAMs #1 and #2 in the system without cache. The results show that only increasing the number of TCAMs cannot avoid TCAM lookup overflow, which is due to the short-time transient burst patterns. However, this transient burst patterns can be effectively handled by cache (its lookup capacity is 894,327 lookups in the period), hence the lookup load in TCAM remains reasonably low even when such burst patterns appear.

When the buffers of TCAMs overflow, the incoming contents cannot be inspected and have to be bypassed, thus incurring the risk of those undetected attacks. The *content bypass ratio*, defined as the ratio of the number of bypassed content bytes to the total number of content bytes passing through the system, is used as the key metric to evaluate the performance of the proposed cache mechanism.

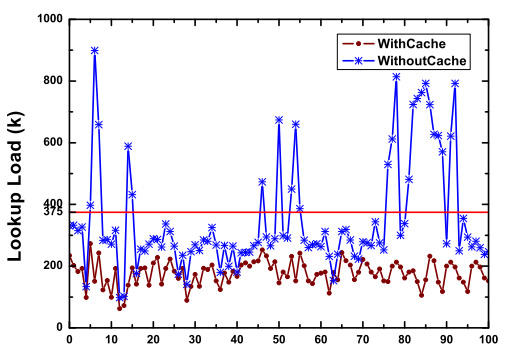


Fig. 3. Time-varying lookup load distribution for TCAM #2.

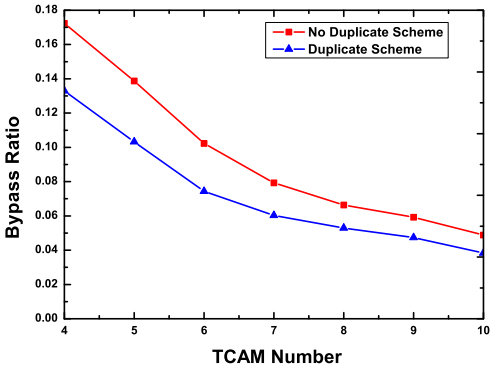


Fig. 4. Bypass ratio of pure TCAM system by varying the number of TCAMs.

We increase the number of TCAMs in the Pure TCAM system from 4 to 10. For each number of TCAMs, the PGs, policy rules, and IP prefixes are redistributed according to the LS Algorithm.

In all the test traces, the proposed prototype has zero bypass ratio, and its bypass ratio is not shown here. The bypass ratio of Pure TCAM system without duplicated patterns is shown in Fig. 4. From Fig. 4, we know that the bypass ratio decreases as the number of TCAMs increases. But it cannot be zero even if the number of TCAMs reaches 10. For 10 TCAMs, the total lookup capacity is 1,000 MLPS which exceeds the capacity of the proposed prototype (912 MLPS), its bypass ratio is still 3.8 percent. This happens due to the fact that the burst lookup load goes to a few PGs which may be stored in the same TCAM, hence purely adding TCAMs cannot handle such burst lookups.

To balance the lookup load of the prefix groups with high matched ratio, we duplicate the most popular PGs to every TCAM in Pure TCAM system (called duplicated scheme). When these prefix groups come, the selector can choose the TCAM with lightest lookup load to handle them. Fig. 4 (lower blue curve) shows that the duplicated scheme can reduce the bypass ratio, but the ratio still cannot reach zero, i.e., still cannot avoid overflow. This happens due to the fact that the burst lookup load coming from some unpopular PGs (i.e., a PG has very low lookup ratio in all the traces, but very high in a short period). So, the DPC is very useful to dynamically balance the lookup load, particularly for short-time burst lookup load coming from some unpopular PGs.

4.5 Distributed Parallelism versus Straightforward Parallelism

To evaluate the performance of the distributed scheme, we compare it to a straightforward duplicating scheme. In the straightforward duplicating scheme, all CF patterns are duplicated to every TCAM. So, it can process multiple string lookup in parallel. Four 100 MLPS TCAMs are used for the three tasks. We inspect the bypass ratio of the two schemes in 10 Gbps line speed. Table 3 shows the performance of them. As our expectation, the bypass ratio of the straightforward duplicating scheme is lower than that of the distributed scheme. As the burst patterns appear, all the four TCAMs can deal with the burst pattern. But it still cannot obtain enough high performance to zero bypass ratio without caches. The

TABLE 3
Bypass Ratio for Straightforward Parallelism versus Distributed Parallelism

	Straightforward Parallelism		*Distributed Parallelism*	
	Without Cache	With Cache	Without Cache	With Cache
Bypass Ratio	9.4	0.0	12.3	0.0

lower bypass ratio costs more TCAM storage, and hence high power consumption for lookups. In this case, the distributed scheme only needs about 48 Kbits storage (as the current Snort rule size), while the straightforward one needs about 192 Kbits storage. As the pattern size expands, more storage for straightforward one is needed. From the results, we can also see that the cache is effective in reducing the TCAM lookups. The distributed scheme can achieve the same effect as the straightforward duplicating scheme by using cache. As a result, the distributed scheme with cache outperform the straightforward one in both the storage and power consumption while achieving the same bypass ratio.

4.6 Throughput

We generate some synthetic data to evaluate the overall packet inspection throughput and the ability of handling attack packets of the proposed prototype. The *throughput* is defined as the number of bits which can be handled by a system in a second. The attack patterns of Snort rules are randomly chosen as the attacking patterns. The attack patterns are inserted into every packet of Trace #1 and the dirty data percentage is defined as the ratio of attack pattern bytes to packet content bytes. We set an allowable upper bounded bypass ratio here. We vary the dirty data percentage by 39, 47, and 59 percent. These percentages correspond to inserting 5, 10, and 15 attacking patterns to each packet, respectively.

The four curves in Fig. 5 depict the inspection throughput by setting the upper bounded bypass ratio as 0, 1, 2, and 5 percent, respectively. The results show that the proposed solution can handle up to 18.2 Gbps line rate with 0 percent bypass ratio for normal traffic. The lookup throughput is reduced to about 6.1 Gbps for 60 percent dirty data traffic. To handle OC-192 line rate, it can suffer more than 35 percent dirty data with four 100 MLPS TCAMs plus 256 MLPS SPC and DPC. In the real world, the attack data ratio is usually just a few percentages (most traffic are normal traffic), hence the proposed solution with four TCAMs plus SPC and DPC can support OC-192 line rate.

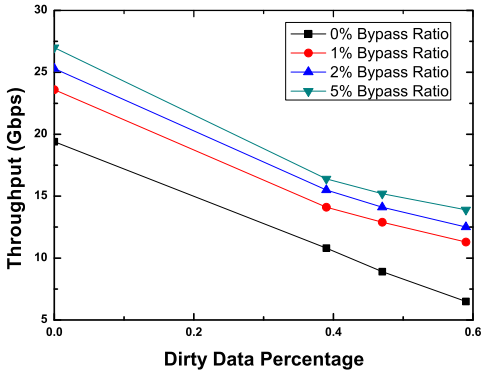


Fig. 5. Throughput by varying the dirty data percentage and bypass ratio.

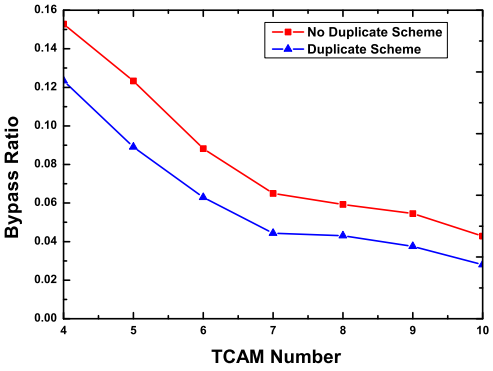


Fig. 6. Bypass ratio of pure TCAM system by varying the number of TCAMs for recent Internet traffic.

4.7 Test on the Real Recent Internet Trace

The traces in the above experiments are old. As the Internet changes dramatically, the traffic patterns may have significant differences. Here, we downloaded some recent traffic to evaluate the proposed solution. We use the real Internet trace from HuNan Province, China, which have 3.7 GBytes size. It includes a 30-minutes data collected from 10 Gbps backbone.

We study the bypass ratio and the throughput on the basis of the recent Internet traffic in this case. Similarly, the proposed solution has zero bypass ratio, which is not shown in the Fig. 6. Compared the results in Fig. 6 with those in Fig. 4, we can conclude that the bypass ratio is about 2 percent higher for recent traces than that for the old traces. This may be due to that the recent normal traffic have some more patterns matching some attack pattern prefixes.

Fig. 7 shows the throughput of the proposed solution varying with the percentage of bypass ratio. From the figure, it could be seen that the throughput is almost the same as that in the old traces. The results show that the proposed architecture is able to be used in the current Internet for executing the integrated tasks.

4.8 Cost Analysis

Consider a system with N_p bytes policy rules, N_c bytes attacking patterns, N_r route entries, R Gbps line speed, and

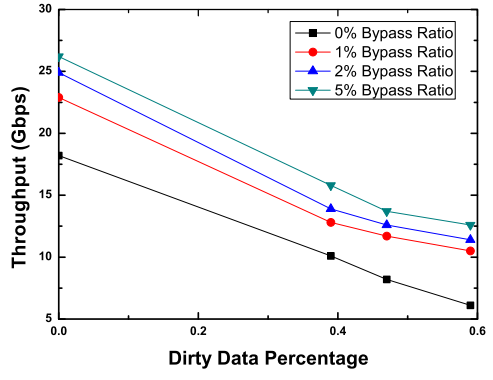


Fig. 7. Throughput by varying the dirty data percentage and bypass ratio.

TCAMs with S byte storage, the speed of L lookups per second. Suppose that the PF lookup can be handled by a single TCAM and the storage of CF can be loaded to a single TCAM. For PF only, it needs $T_p = \lceil N_p/S \rceil$ TCAMs to store all the rules, and for CF, it needs $T_c = \lceil R/8L \rceil$ TCAMs to handle the lookups. For LPM, it needs $T_R = \lceil N_r \times 4/S \rceil$. The total number of TCAMs for nonintegrated solution is $T_p + T_c + T_R$.

Suppose the cache system can efficiently deal with the worst case, and the prefix hit ratio is β. The number of needed TCAMs can be reduced to $T_c^{'} = \lceil \beta R/8L \rceil$. For the integrated solution, the number of TCAMs is the $\max\{T_p, T_c^{'}, T_R\}$. Hence, the saved number of TCAMs for the integrated solution is $T_p + T_c + T_R - max\{T_p, T_c^{'}, T_R\}$. For example, suppose there are 2,000 patterns with 40 bytes each, and 1 million policy rules, 155K IP prefixes. For 10 Gbps line rate, using TCAM with 100 MLPS and 10 Mbits storage, the number of TCAMs needed is reduced from 31 to 13. If the percentage of lookups reduced by the prefix filter is no less than 60 percent, the number of needed TCAMs will be reduced to 11.

The total storages of TCAMs for the proposed distributed architecture will be reduced from 265 to 11 Mbits. So, the proposed architecture for the integrated three tasks is much more cost effective.

5 Conclusions

We proposed a novel distributed TCAM coprocessor architecture for the integration of LPM, PF, and CF. First, a prefix filter is introduced to reduce the TCAM lookups for CF. Then, a set of algorithms is proposed to balance the lookup loads among the TCAMs, including a novel cache system and several heuristic algorithms. The cache system includes a SPC and a DPC to adaptively balance the long-time and the short-time lookup load, respectively. The proposed heuristic algorithms distribute patterns/rules to the TCAMs to balance the lookup load, as well as the storage load. The integrated solution exploits the complementary characteristics of the three tasks and significantly reduces the overall cost. Simulations based on the real-world traffic traces show that the proposed solution can match 10 Gbps line rate for all the three tasks simultaneously with similar cost as that of executing CF task only. For the next step, we will extend our design to integrate more tasks such as address resolution.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61070198, 61070037, 61170288, 61170287, and 60903040.

References

[1] V. Paxson, K. Asanovic, S. Dharmapurikar, J. Lockwood, R. Pang, R. Sommer, and N. Weaver, "Rethinking Hardware Support for Network Analysis and Intrusion Prevention," *Proc. USENIX Workshop Hot Topics in Security,* pp. 1-6, 2006.

[2] A. Bremler-Barr and D. Hay, "Space-Efficient TCAM-Based Classification Using Gray Coding," *IEEE Trans. Computers,* vol. 61, no. 1, pp. 18-30, Jan. 2012.

[3] M. Faezipour and M. Nourani, "Wire-Speed TCAM-Based Architectures for Multimatch Packet Classification," *IEEE Trans. Computers,* vol. 58, no. 1, pp. 5-17, Jan. 2009.

[4] M. Akhbarizadeh, M. Nourani, R. Panigrahy, and S. Sharma, "TCAM-Based Parallel Architecture for High-Speed Packet Forwarding," *IEEE Trans. Computers,* vol. 56, no. 1, pp. 58-72, Jan. 2007.

[5] K. Zheng, H. Che, Z. Wang, and B. Liu, "PPC-RE: TCAM-Based Distributed Parallel Packet Classification Algorithm with Range-Matching," *IEEE Trans. Computers,* vol. 55, no. 8, pp. 947-961, Aug. 2006.

[6] K. Zheng, C. Hu, H. Lu, and B. Liu, "TCAM-Based Distributed Parallel IP Lookup Scheme and Performance Analysis," *ACM/IEEE Trans. Networking,* vol. 14, no. 4, pp. 863-875, Aug. 2006.

[7] R. Cohen and D. Raz, "Simple Efficient TCAM Based Range Classification," *Proc. IEEE INFOCOM,* pp. 461-465, Mar. 2010.

[8] O. Rottenstreich and I. Keslassy, "Worst-Case TCAM Rule Expansion," *Proc. IEEE INFOCOM,* pp. 456-460, Mar. 2010.

[9] Y. Chang, C. Lee, and C. Su, "Multi-Field Range Encoding for Packet Classification in TCAM," *Proc. IEEE INFOCOM,* pp. 196-200, Apr. 2011.

[10] H. Che, Z. Wang, K. Zheng, and B. Liu, "DRES: Dynamic Range Encoding Scheme for TCAM Coprocessor," *IEEE Trans. Computers,* vol. 57, no. 7, pp. 902-915, July 2008.

[11] A. Bremler-Barr, D. Hay, D. Hendler, and R. Roth, "PEDS: Parallel Error Detection Scheme for TCAM Devices," *IEEE/ACM Trans. Networking,* vol. 18, no. 5, pp. 1665-1675, Oct. 2010.

[12] Z. Wang, H. Che, J. Cao, and J. Wang, "TCAM-Based Solution for Integrated Traffic Anomaly Detection and Policy Filtering," *Computer Comm.,* vol. 32, no. 17, pp. 1893-1901, Nov. 2009.

[13] F. Yu, R. Katz, and T. Lakshman, "Gigabit Rate Packet Pattern Matching Using TCAM," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* pp. 174-183, 2004.

[14] SNORT system, http://www.snort.org, 2012.

[15] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs," *Proc. ACM SIGCOMM,* pp. 193-204, 2005.

[16] V. Ravikumar, R. Mahapatra, and L. Bhuyan, "EaseCAM: An Energy and Storage Efficient TCAM-Based Router Architecture for IP Lookup," *IEEE Trans. Computers,* vol. 54, no. 5, pp. 521-533, May 2005.

[17] W. Lu and S. Sahni, "Low-Power TCAMs for very Large Forwarding Tables," *IEEE Trans. Networking,* vol. 18, no. 3, pp. 948-959, June 2010.

[18] Z. Wang, H. Che, M. Kumar, and S. Das, "CoPTUA: Consistent Policy Table Update Algorithm for TCAM without Locking," *IEEE Trans. Computers,* vol. 53, no. 12, pp. 1602-1614, Dec. 2004.

[19] A. Liu, C. Meiners, and E. Torng, "TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs," *IEEE/ACM Trans. Networking,* vol. 18, no. 2, pp. 490-500, Apr. 2010.

[20] R. McGeer and P. Yalagandula, "Minimizing Rulesets for TCAM Implementation," *Proc. IEEE INFOCOM,* pp. 1314-1322, Apr. 2009.

[21] H. Hwang, S. Ata, K. Yamamoto, K. Inoue, and M. Murata, "A New TCAM Architecture for Managing ACL in Routers," *IEICE Trans. Comm.,* vol. E93-B, no. 11, pp. 3004-3012, 2010.

[22] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* pp. 120-131, Nov. 2003.

[23] L. Foschini, *Stateful Intrusion Detection in High-speed Networks: A Formalization and Analysis of High-speed Stateful Signature Matching for Intrusion Detection.* VDM Verlag, 2009.

[24] C. Kruegel and F. Valeur, "Stateful Intrusion Detection for High-Speed Networks," *Proc. IEEE Symp. Research on Security and Privacy,* pp. 285-293, 2002.

[25] C. Lin, C. Huang, C. Jiang, and S. Chang, "Optimization of Pattern Matching Circuits for Regular Expression on FPGA," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 15, no. 12, pp. 1303-1310, Dec. 2007.

[26] H. Wang, S. Pu, G. Knezek, and J. Liu, "A Modular NFA Architecture for Regular Expression Matching," *Proc. Ann. ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays (FPGA),* pp. 209-218, Feb. 2010.

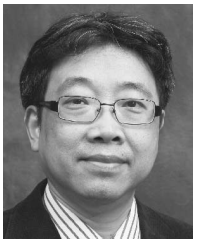
[27] I. Sourdis, D. Pnevmatikatos, and S. Vassiliadis, "Scalable Multi-gigabit Pattern Matching for Packet Inspection," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 16, no. 2, pp. 156-166, Feb. 2008.

[28] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep Packet Inspection Using Parallel Bloom Filters," *IEEE Micro,* vol. 24, no. 1, pp. 52-61, Jan./Feb. 2004.
[29] P. Lin, Y. Lin, Y. Lai, Y. Zheng, and T.-H. Lee, "Realizing a Sublinear Time String-Matching Algorithm with a Hardware Accelerator Using Bloom Filters," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 17, no. 8, pp. 1008-1020, Aug. 2009.
[30] A. Goel and P. Gupta, "Small Subset Queries and Bloom Filters Using Ternary Associative Memories, with Applications," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS),* pp. 143-154, 2010.
[31] Y. Weinsberg, S. Tzur-David, D. Dolev, and T. Anker, "High Performance String Matching Algorithm for a Network Intrusion Prevention System (NIPS)," *Proc. Workshop High Performance Switching and Routing,* pp. 151-157, June 2006.
[32] C. Meiners, J. Patel, E. Norige, E. Torng, and A. Liu, "Fast Regular Expression Matching Using Small TCAMs for Network Intrusion Detection and Prevention Systems," *Proc. 19th USENIX Conf. Security,* pp. 111-126, Aug. 2010.
[33] "MIT DARPA Intrusion Detection Data Sets," http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html, 2012.
[34] D. Taylor and J. Turner, "ClassBench: A Packet Classification Benchmark," *IEEE/ACM Trans. Networking,* vol. 15, no. 3, pp. 499-511, June 2007.
[35] RouteViews: http://archive.routeviews.org/, 2011.

Zhiping Cai (M'08) received the BS, MSc, and PhD degrees in computer science from National University of Defense Technology, Changsha, China, in 1996, 2002, and 2005, respectively. Now, he is an associate professor of the School of Computer, National University of Defense Technology. His current research interests include network security and network virtualization. He is a member of the IEEE.

Zhijun Wang received the PhD degree in computer science and engineering from The University of Texas at Arlington, 2005. Now, he is an assistant professor of the Department of Computing, The Hong Kong Polytechnic University. His current research interests include high-speed network, network security, traffic control, data management in mobile networks and peer-to-peer networks.

Kai Zheng (M'02) received the MS and PhD degrees both in computer science from Tsinghua University, Beijing, China, in 2003 and 2006, respectively. He is currently working with IBM Research China, as a research staff member. His research interests include high-speed packet forwarding, network intrusion detection system, and data center networking. He is currently working on novel data center network control models and new security service models. He is a senior member of the IEEE.



Jiannong Cao received the BSc degree in computer science from Nanjing University, China, and the MSc and PhD degrees in computer science from Washington State University, Pullman. He is currently a chair professor and the head in the Department of Computing at Hong Kong Polytechnic University. His research interests include mobile and pervasive computing, computer networking, parallel and distributed computing, and fault tolerance. He has published more than 250 technical papers in the above areas. He is a senior member of the China Computer Federation, the IEEE, the IEEE Computer Society, and the IEEE Communication Society, and a member of the ACM. He has served as an associate editor and a member of editorial boards of several international journals, including the *IEEE Transactions on Parallel and Distributed Systems, Pervasive and Mobile Computing*, and *Wireless Communications and Mobile Computing*.