



Algorithms to speedup pattern matching for network intrusion detection systems[☆]



Kai Zheng^a, Zhiping Cai^{b,*}, Xin Zhang^c, Zhijun Wang^d, Baohua Yang^a

^a IBM China Research Lab, Beijing, PR China

^b School of Computer, National University of Defense Technology, ChangSha 410073, PR China

^c Carnegie Mellon University, Pittsburgh, PA, USA

^d Dep. of Computing, Hong Kong Polytechnic University, Hong Kong, PR China

ARTICLE INFO

Article history:

Received 8 April 2013

Received in revised form 13 February 2015

Accepted 15 February 2015

Available online 21 February 2015

Keywords:

Negative pattern

Exclusive matching

Pattern matching

Intrusion detection

ABSTRACT

High-speed network intrusion detection systems (NIDSes) commonly employ TCAMs for fast pattern matching, and parallel TCAM-based pattern matching algorithms have proven promising to achieve even higher line rate. However, two challenges impede parallel TCAM-based pattern matching engines from being truly scalable, namely: (1) how to implement fine-grained parallelism to optimize load balancing and maximize throughput, and (2) how to reconcile between the performance gain and increased power consumption both due to parallelism. In this paper, we propose two techniques to answer the above challenges yielding an ultra-scalable NIDS. We first introduce the concept of negative pattern matching, by which we can splice flows into segments for fine-grained load balancing and optimized parallel speedup while ensuring correctness. negative pattern matching (NPM) also dramatically reduces the number of Ternary Content Addressable Memory (TCAM) lookups thus reducing the power consumption. Then we propose the idea of exclusive pattern matching, which divides the rule sets into subsets; each subset is queried selectively and independently given a certain input without affecting correctness. In concert, these two techniques improve both the pattern matching throughput and scalability in any scenario. Our experimental results show that up to 90% TCAM lookups can be saved, at the cost of merely 10% additional 2-byte index table lookups in the SRAM.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

As a real-field tested mechanism, the network intrusion detection system (NIDS) has remained a prominent resort in industrial security practices. Though not bullet proof to every exploit, NIDS dramatically increases the bar for adversaries and diminishes the attack surface. As a result, NIDS empirically prevents a vast array of malicious intrusions that could potentially incur losses of hundreds of millions of dollars every year [1].

Despite its importance and widespread deployment, throughput and scalability have plagued NIDS due to the expensive *Pattern Matching* (PM) operations involved. PM requires linearly inspecting every packet payload for potential matches against

any rule (pattern) in a pattern set. Hence, PM is inherently resource-intensive regarding both computation and communication (i.e., I/O), when the traffic volume and the size of the pattern set are non-trivial. Though the *Ternary Content Addressable Memory* (TCAM) based PM algorithms tend to deliver superior throughput compared to pure software-based counterparts, the reported performance achieved to date is still a far cry from the rocketing network line speed. In a nutshell, such a performance limitation is attributed to the relatively low working frequency of TCAM chips and the high power consumption of TCAMs.

We observe that content pre-filtering and parallelism are two promising means to dramatically increase PM throughput for TCAM-based algorithms. First, content pre-filtering refers to the pre-processing of flows or pattern sets to reduce the overhead of the actual real-time PM operations. Second, exploiting parallelism has been a conventional tenet for achieving greater system efficiency/performance. Unfortunately, traditional pre-filtering schemes require sequential processing of the input flows and are difficult to be parallelized. In addition, parallelism must be deployed while retaining the scalability and correctness of the PM operations;

[☆] The preliminary version of this work has been published in the proceeding of IEEE INFOCOM 2010.

* Corresponding author. Tel.: +86 731 84573674; fax: +86 731 84573675.

E-mail address: zpcai@nudt.edu.cn (Z. Cai).

¹ This author is supported by the NSFC (Nos. 61379145, 61202429, 61363071, and 61272510).

this is highly challenging as parallel processing tends to increase power consumption and packets in a flow may mandate sequential scanning to detect malicious patterns that span across multiple packets.

To substantially speed up PM operations via parallelism while preserving scalability, we propose to partition both individual flows and the pattern set. First, an incoming flow is divided into consecutive segments, which can be inspected in parallel. Second, the entire pattern set is partitioned into multiple subsets, which can be looked up individually and selectively (as opposed to checking the entire pattern set every time), to provide extra dimensions for parallelism and to reduce the power consumption of hardware accesses. Implementing parallelism via such a two-level partitioning, however, creates two fundamental challenges for retaining PM correctness. First, partitioning the flows is non-trivial, since a sophisticated attacker can split a malicious pattern across multiple packets within a flow. Consequently, to capture such cross-packet dependencies (thus avoiding false negatives), flows are usually re-assembled and the packets in each flow are inspected sequentially [2]. Second, the partition of the pattern set should not cause false negatives. Moreover, it is challenging to minimize the power consumed by hardware accesses and to effectively balance the workload among the pattern subsets.

In light of the above observations, in this paper we develop an efficient parallel PM module based on TCAM coprocessors, which achieves high PM throughput and significantly reduces the number of TCAM accesses (thus minimizing the power consumption). We first introduce the novel concept of *Negative Patterns* (NPs) to address the challenge of partitioning flows while preserving correctness. Simply put, an NP is a string, any part of which (i.e., any substring of which) will not match any “normal” pattern in the given pattern set. Hence, by partitioning a flow only at locations within NPs appearing in that flow, it is ensured that no malicious patterns across multiple packets in a flow will be missed. We also show that with flow partitioning thanks to NPs and performing parallel inspection at flow-segment level, the number of TCAM accesses is considerably reduced and better load balancing is achieved compared to performing parallel inspection at the flow level.

Second, we propose the idea of *Exclusive Subsets* to efficiently partition the pattern set as follows. We identify all pairs of two patterns that will not be matched simultaneously within one TCAM lookup input, and separate the two exclusive patterns in such a pair into *exclusive subsets*. Intuitively, each exclusive subset can be looked up independently, since one TCAM lookup input can find matches in at most one of the exclusive subsets, thus saving power consumption. Finally we will show that the techniques of negative patterns and exclusive subsets are inherently complementary, enabling our integrated scheme to outperform previous approaches in any scenario (with either clean or dirty traffic).

1.1. Contributions

We believe this paper can advance the state-of-art NIDS in the following aspects.

- First, we achieve high throughput PM through parallel TCAMs with fine-grained load balancing at the granularity of flow segments.
- Second, both the NPs and exclusive subsets techniques reduce the number of TCAM accesses by 30–90% compared to the prior schemes.
- Third, we show that the two techniques in concert can yield desirable performance in any scenario (with either clean or dirty traffic).

- Finally, we perform extensive theoretical analysis and experimental evaluation to consolidate the effectiveness of the proposed techniques.

2. Related works

Pattern-matching schemes have been studied for a long time. The classic pattern matching schemes include those proposed by Aho and Corasick [3], Boyer and Moore [4], and their extensions/variations [5–11]. These schemes either require a large memory for software implementation or can hardly work with large pattern sets. Some pattern matching schemes [12,13] use heuristics to filter the strings that cannot be matched by any suspicious patterns. For example, these schemes use prefix/suffix of the patterns to do preprocess so as to improve the string matching speed. Regular expression matching and string matching [5–10,14–16] have also been widely studied to speed up the pattern matching performance. Very recently, a multi-core platform was developed for parallel pattern matching [17]. The Bloom filter is a kind of space efficient algorithm for pattern matching. It has also been proposed for intrusion detection in [18,19]. In [18], prefix Bloom filters are proposed to detect multi-packet intrusion signature patterns. In [20], a Bloom filter implemented with on-chip memory block is designed for fast pattern matching. However, due to the variable length of intrusion patterns, a large number of Bloom filters with different length may need to be constructed, and thus making them un-scalable.

To improve the matching speed, some hardware based solutions [19,21–27] using FPGA technology have been proposed for design high speed intrusion detection systems. Some of these approaches are claimed to be able to support 10Gbps wire speed by exploiting parallelism available in hardware implementations and using the state-of-the-art FPGA technology. But the intrusion patterns in the hardware need to be reconfigured when the patterns are updated, and hence these schemes cannot be used to support fast dynamic updates and also with huge implementation overheads in terms of both cost and time.

TCAM is a fast network search engine, and widely used for IP lookup [28,29], packet classification [31,32], and pattern matching [11,22,27,33,34]. In [22], a gigabit rate intrusion detection system using a single-TCAM coprocessor is proposed. The system can handle up to OC-48 (i.e. 2.5Gbps) line rates by using the currently fastest TCAM chips. Due to the need to inspect a data stream by shifting one byte at a time in the TCAM, the system requires high TCAM lookup throughput and also results in very huge power consumption. TCAM based regular expression matching approaches were developed in [27,29,30] to achieve high speed pattern matching. A covered state encoding scheme was designed for the Aho-Corasick multi-pattern matching by implemented in TCAM [11]. Recently, we proposed a TCAM architecture for integrated policy filtering, content filtering and longest prefix matching [34].

Different from the tradition filtering schemes, the proposed algorithm uses flow/content partition which is proven to have the same effect. The “negative patterns” are leveraged to inspect the packet streams and target at finding “negative sub-strings” which should not be included in any patterns, and thus indicates the valid partition at the positions. Since the NP matching is false negative tolerant, parallelism can be much easier to explore than the previous approaches.

3. Design concepts

In this paper, we do not intend to take part in the debate on “which category of PM schemes are the best”, which has been an open topic in the literatures for years. Each category has its own

strength and weakness and, apparently, it depends heavily on the requirements of the specific NIDS under development.

In this paper, we base our work on [22] which uses TCAM as the key building block. Our focus is how to develop parallelism and on the other hand improve power efficiency. We will not address how to handle long patterns and composite patterns, since it has been very well solved in [22] (and will be mentioned in Section 4, in the proposed NIDS prototype, dedicated mechanism will be used to handle the general regular expression match). Before delving into implementation details, in this section we first sketch the high-level ideas of our key approaches, one by one.

3.1. Content pre-filtering vs. flow partition

One major challenge of traditional PM and/or content pre-filtering mechanisms comes from the fact that usually a specific flow stream has to be inspected sequentially (e.g., one byte examined at a time), in order not to miss consecutive sub-strings/patterns. Therefore, individual packets need to be first buffered and then put together in the original order to reconstruct the flow. To boost PM throughput, it is intuitive to deploy parallelism at the flow level, such that the packets within one flow can still be inspected sequentially. This straightforward measure nevertheless turns out to be inefficient, because the network packet flows² tend to vary significantly in size and duration, and large flows with up to several million bytes are not uncommon [35], depriving the flow-level parallelism of effective load balancing.

Flow partition, which partitions an individual flow into successive fragments, enables parallelism at a finer granularity and facilitates better load-balancing for PM. The key challenge lies in that, when segmenting the flows, we must guarantee that any suspicious pattern does not span more than one segment, so that each segment can be inspected individually (see Section 3.2).

Meanwhile, we also observe that, besides the load balancing advantage, flow partition can also provide a PM speedup, especially for TCAM-based schemes. Consider the example shown in Fig. 1. Given that the TCAM slot size is w bytes ($w = 7$ in the example), an input flow with 20 bytes requires 14 TCAM lookups. However, if we partition the flow into two 10-byte segments, then the inspections can be performed on the two segments in parallel, and the total number of TCAM lookups is reduced to only $4 + 4 = 8$. Generally speaking, given a flow Text T with length L , and the TCAM slot size w , the number of TCAM lookups required to inspect T is $L - w + 1$. If we partition T into several small consecutive segments $\{t_1, t_2, \dots, t_n\}$ ($\sum_{i=1, \dots, n} l_i = L$, where l_i is the length of the i th segment), then the number of TCAM lookups is reduced to,

$$\sum_{i=1, \dots, n} \text{Max}(l_i - w + 1, 1) \quad (3.1)$$

We can express the minimal number of TCAM lookups as,

$$\begin{aligned} \text{Min} \sum_{i=1, \dots, n} \text{Max}(l_i - w + 1, 1) &= n + \text{Min} \sum_{i=1, \dots, n} \text{Max}(l_i - w, 0) \\ &= n + \lceil L/w \rceil, \text{ iff } \forall i, l_i = w \end{aligned} \quad (3.2)$$

Eq. (3.2) indicates that if we could partition the flow content into segments with sizes as close to the TCAM slot size as possible, then the number of TCAM lookups would be minimized. In addition, note that the result of $L - w + 1 - \sum_{i=1, \dots, n} \text{Max}(l_i - w + 1, 1)$ reflects the number of TCAM lookups that can be saved by using partition,

² Note that a malicious sender can intentionally separate the attacking “fingerprints” across multiple packets to avoid being detected at each individual packet. Therefore, to combat such surreptitious attackers, sessions/flows must be reconstructed. For example, SNORT [11] uses several preprocessor plug-ins to defrag IP fragments and to rebuild the TCP flows/sessions before deploying PM against the network data.

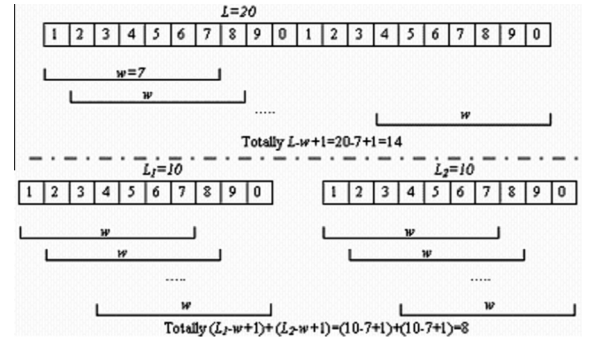


Fig. 1. Number of lookups reduces when the flow is partitioned.

compared to that without flow partition. The following calculation provides us with a clear estimation of such a reduction. Considering the case where we guarantee that $\forall i, l_i \geq w$ (i.e. not to partition the flow into segments shorter than w bytes), we further have,

$$\begin{aligned} n_{\text{save}} &= L - w + 1 - \sum_{i=1, \dots, n} \text{Max}(l_i - w + 1, 1) \\ &= L - w + 1 + nw - n - \sum_{i=1, \dots, n} l_i = (n - 1)(w - 1) \end{aligned} \quad (3.3)$$

Eq. (3.3) implies that the more segments we can partition the flow into, the fewer TCAM inspections will be needed (which is not directly related to the total flow length L); the larger the window size w is, the more significant the benefit can be. In this sense, flow partition can also be treated as a form of content pre-filtering for the traditional PM, since it tends to trade simple operations for expensive ones; the more positions we could find to partition the flow (without false negatives), the more TCAM inspections we can save, as long as the segment sizes are no smaller than w . We further identify the challenge for such inner-flow partition as twofold: firstly, how can we guarantee that no pattern will span across multiple segments? Secondly, how can we make the partition process efficient? The subsequent subsection sketches the *negative pattern* technique as the solution.

3.2. Negative pattern matching (NPM)

Traditional content pre-filtering methods try to find the occurrence of certain sub-strings from a given pattern set in an input stream, which is called “positive pattern filtering”. Here we propose the notion of *Negative Pattern* (NP) matching/filtering. Instead of finding the positive occurrence of the sub-strings in a given pattern set, we aim to unearth NPs that do not appear in the given set and that allow the flow content to be partitioned without false negatives. As aforementioned, flow partition based on negative pattern matching serves as a form of content pre-filtering. Let us first formally define a negative pattern as follows.

Definition 1. Pattern and Pattern matching: In computer science, pattern matching is the act of checking a sequence of tokens for the presence of some strings. In this paper, a pattern is refer to a set of specific ternary strings (i.e., simple strings with wildcards) in the traffic which indicate the probability of potential intrusion or abnormality of the network. And Pattern matching is referred to the action of identifying element(s) of a given pattern set that appears in the providing traffic strings.

Definition 2. Negative Pattern (NP): Given a pattern set $S = \{P_1, P_2, \dots, P_n\}$, a string np is defined as a negative pattern of S iff any k -byte ($k > 1$) suffix of np is not a sub-string of any $P_i \in S, i = 1, 2, \dots, n$.

For illustration, given that the pattern set S includes two patterns {pattern, attack}; the “positive” sub-string set of S is, $SS = \{pa, at, tt, te, er, rn, ta, ac, ck, pat, att, tte, ter, ern, tta, tac, ack, patt, atte, tter, tern, atta, ttac, tack, patte, atter, ttern, attac, ttack, patter, attern, attack, pattern\}$. According to Definition 2, we say pattern “negative” is a negative pattern of S , since none of its k -byte ($k > 1$) suffixes, i.e. any string in {ve, ive, tive, ative, gative, egative, negative}, is included in SS .

Theorem 1. *If we find a negative pattern np in the flow to be inspected, then it guarantees that no pattern(s) in S appear(s) across the adjacent segments after the partition, if only we cut the flow between the last two bytes of np .*

Proof. We prove Theorem 1 by contradiction. Suppose that there exists a pattern $p \in S$ across the partitioning point. Hence p has at least 2 bytes, i.e. the last 2 bytes of np , and therefore p must contain a 2-byte suffix of np . However, according to the definition, any 2-byte suffix of np must not be a sub-string of p . Therefore such a pattern $p \in S$ must not exist. \square

For instance, let us continue to use the pattern set $S = \{\text{pattern}, \text{attack}\}$. Given that the byte stream to inspect is $T = \dots \text{thisisapatternindicatingattacks} \dots$; according to the definition of a negative pattern, we know that {isi, sap, ... erni, ... inga, ...} are NPs of S , since any k -byte ($k > 1$) suffix of them is not included in SS . Therefore, according to Theorem 1, we say that, {...this, “isa”, “pattern”, “indicating”, “attacks...”} is a valid partition of T , which will not lead to false negatives if the partitions are inspected individually.

Definition 2 and Theorem 1 answer the first question in Section 3.1: how to partition the flow while guaranteeing that no potential pattern in the flow will be missed if each segment is to be examined separately. By Theorem 1, whenever an NP is found, the flow can be partitioned at a position within the NP (i.e., between the last 2 bytes).

So far, there are still problems remaining unsolved, i.e. how to build up an NP set and how to make the partition process efficient enough to achieve an overall performance gain. We will delve into the details of our NP matching implementation in Section 4; in the following, we first present the key ideas of pattern set partitioning with exclusive pattern matching.

3.3. Using multi-block TCAM

The full associativity of all TCAM entries gets in the way of making TCAM/Tagged-memory faster and scales better (with respect to both power consumption and timing criteria). Note that traditionally, the TCAM is actually a single block fully associated tagged memory. Reducing the capacity of the TCAM blocks will make it possible to deliver significantly higher working frequency, although this might reduce the number of entries that can be compared at a time. It is intuitive for us to think about using a TCAM with a set of small blocks rather than one with a single huge fully associative block. So that we can try to partition the pattern set into several subsets and distribute them among the blocks on a TCAM to increase the system frequency. With the aid of mature pipeline techniques (where the scan in the subsets are treated as the stages in the pipeline), system throughput can be improved accordingly. In this paper, our concepts and algorithms are instantiated on multi-block TCAMs which operate at a higher frequency; however, they are not limited to using the pipelining processing model. Furthermore, we argue that the performance gain by using the multi-block TCAM can pay off the implementation complexity introduced.

3.4. Exclusive subsets and exclusive matching (EM)

There might be numerous ways of partitioning a pattern set (e.g., so as to fit the subsets into the TCAM blocks or multiple single block TCAM chips, respectively). It might be interesting to ask “Are there partitioning schemes which make it possible to avoid going through all of the subsets for each specific lookup?” One might think of hash-based schemes (i.e., partitioning the pattern set based on the hash of certain fixed bits of each pattern) at first glance, which sound intuitive and straightforward. However, unfortunately, because of the following considerations, the hash-based pattern grouping schemes are not practical for PM.

- (1) Wildcards appear frequently in arbitrary locations in the patterns (also note that the presentation of the short patterns in TCAM always include several wildcard bits, as shown in [22]). Replication of the patterns might be required when the hash keys of the patterns include wildcards. For instance, given the pattern set {1*0*, *1*0, 1*00, 11*0} and

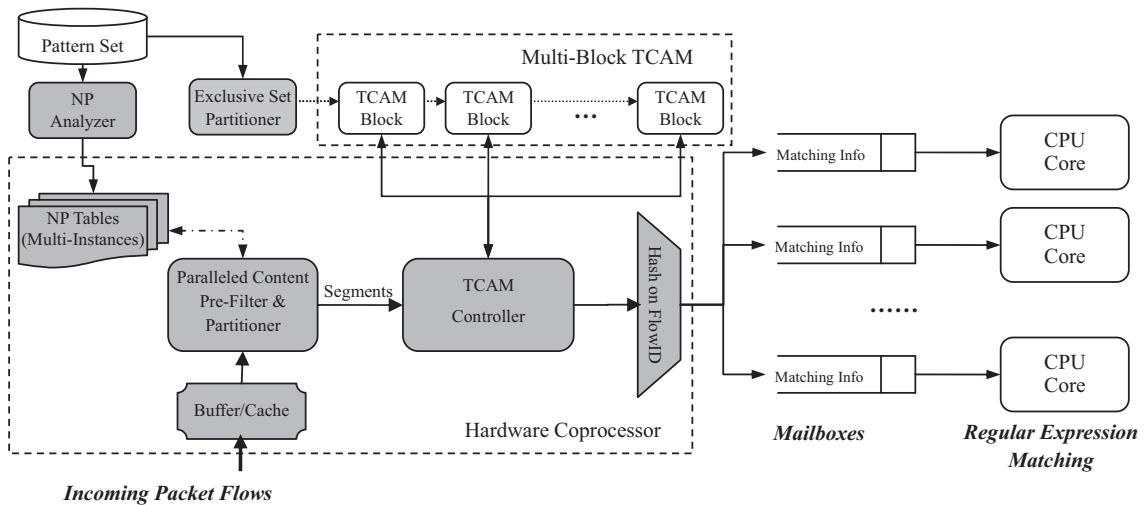


Fig. 2. The architecture of the proposed NIPS.

assuming the first 2 bits are chosen as the hash key, then the 4 patterns are grouped into $2^2 = 4$ subsets, i.e., $\{\emptyset\}_{00}$, $\{1*0\}_{01}$, $\{1*0*\}_{10}$, $\{1*0*,1*0,1*00,11*0\}_{11}$. We can see that 2 out of the 4 patterns are replicated, resulting in an expansion ratio of 50%. Moreover, the last subset includes all the patterns, which means the partition actually failed.

- (2) Load balancing tends to be a tough challenge for the hash-based partitioning scheme to solve. We note that the occurrence of contents (i.e., characters) in the traffic is usually unpredictable. Different pattern groups may therefore have unbalanced and fluctuating load ratio, and some pattern subsets may become access bottlenecks.

Now, we will go through a very simple example to gain some intuition in our search for an alternative to hashing. Consider a pattern set with two patterns $\{ax*c, x*yz\}$, and assume the width of the TCAM (i.e. the scan window) is $w = 7$ Bytes. We can say that the two patterns are matched exclusively for any traffic content/text in any cases, no matter how the patterns are presented in the TCAM, for example “ $ax*c***$ ”, “ $*ac*C**$ ”, ... or “ $****aC*c$ ”... The observation is that none of the prefixes of the patterns is also a suffix of the other, nor are they sub-strings of each other. Therefore, any content that matches both patterns at the same time must be at least $|ax*c| + |x*yz| = 8$ bytes, which is longer than the scan window size $w = 7$.

Definition 3. The “MinClen of Pattern A and Pattern B”, i.e. $\text{MinClen}(A, B)$, is defined as $|S|$ where S is a shortest string matching both A and B.”

For example, $\text{MinClen}(\text{“123ab”}, \text{“ab123”}) = 7$, since pattern “ab123ab” is (one of) the shortest pattern matching both “ab123” and “123ab”.

Definition 4. $A \nabla_w B$: Given pattern A and B, and the inspection window size w , symbol $A \nabla_w B$ denotes the case when A and B must be matched exclusively, i.e. there does not exist such a string P satisfying both $A, B \in \Omega(P)$ and $|P| \leq w$.

Theorem 2. $A \nabla_w B$ when $\text{MinClen}(A, B) > w$.

Proof. Assume that there exists a string S ($|S| = w$) which matches both Pattern A and B, and $\text{MinClen}(A, B) > w$. It is obvious that A and B must be sub-strings of S, i.e. S satisfies $A, B \in \Omega(S)$. However, according to the definition of “Minimum Combine Length”, any string S satisfies $A, B \in \Omega(S)$ should be longer than w , which conflicts with the assumption. Therefore such a string S does not exist, in another word, A and B are impossible to be matched at the same time. \square

Theorem 2 explains why “ $ax*c$ ” and “ $x*yz$ ” are matched exclusively given the scan window size $w = 7$, since $\text{MinClen}(\text{“ax*c”}, \text{“x*yz”}) = 8 > w = 7$.

Definition 5. $S1 \Delta_w S2$: Given the scan window size w , we say pattern subset $S1$ and $S2$ are matched exclusively with each other, (i.e. $S1 \Delta_w S2$) if and only if $\forall A \in S1, \forall B \in S2, A \nabla_w B$.

Definition 5 poses a formal criterion on when to leverage the “Exclusive” property to speed up the lookup, regardless of the traffic content within the w -byte scan window. Given the “exclusive” pattern subsets, $S_1, S_2, \dots, S_n, \forall 1 \leq i, j \leq n, S_i \Delta_w S_j$, since for any w -byte string (or those shorter than w -byte), match can be found in no more than 1 subset, all the subsets need not to be walked through all the time. The lookup process can be started by inspecting the traffic against the exclusive subsets sequentially, and terminates

whenever match is found; only when no match is found is it necessary to go through all the subsets. According to **Definition 5**, it is easily to obtain **Theorem 3** as follows:

Theorem 3. If $S1 \Delta_w S3$ and $S2 \Delta_w S3$, then $(S1 \cup S2) \Delta_w S3$.

In Section 4, we will discuss in details on how to partition a pattern set into exclusive subsets by using the idea of exclusive pattern matching, and make it actually practical.

3.5. The complementarity between np matching and exclusive matching

A very interesting fact of “exclusive matching” is that the “dirtier” the traffic content is, the greater the expected speedup, since dirty traffic leads to higher chance of getting a “hit”. The earlier a hit is found, the more TCAM accesses can be saved. This advantage of “exclusive matching” provides a very good countermeasure for the worst-case scenario in which the attackers generate dirty traffic/attacks to exhaust the NIPS/NIDS. On the other hand, in ordinary usage, when the traffic is typically very clean, attack patterns are seldom found. In this case, NP matching, i.e., the idea introduced in Sections 3.1 and 3.2, in turn results in performance gains. More theoretical analysis on a system leveraging the complementarity between NP matching and exclusive matching will be discussed in Section 5.

4. Implementation and prototype

4.1. Overall architecture and workflow

As depicted in Fig. 2, the proposed PM system consists of an NP Analyzer (NPA), an Exclusive Set Partitioner (ESP), a hardware accelerator, a multi-block TCAM module, and a general-purpose multicore processor.

The workflow comprises an off-line preprocessing stage and the subsequent real-time PM stage. In the off-line preprocessing stage, (i) NPA scans the pattern set to obtain the NPs and put them (there might be multiple instances for parallelism) into the accelerator; (ii) in addition, ESP inspects the pattern set, partitions it into exclusive subsets and stores them independently in the TCAM blocks.

In the real-time PM stage, the incoming flows will be dispatched into the hardware accelerator for content pre-filtering (i.e. flow partition) and be inspected for simple patterns in parallel on the exclusive subsets populated in the TCAM blocks. Then the matching results (e.g. which simple pattern has been matched at which position in the flow, etc.) will be delivered to the general-purpose processors for further post-processing such as a general

<i>p</i>	<i>a</i>	<i>t</i>	<i>t</i>	<i>e</i>	<i>r</i>	<i>n</i>
<i>a</i>	<i>t</i>	<i>t</i>	<i>a</i>	<i>c</i>	<i>k</i>	

Original Pattern Set

		NP?
...	...	
<i>t</i>	<i>a</i>	0
<i>t</i>	<i>b</i>	1
...	...	
<i>p</i>	<i>a</i>	0
...	...	
<i>x</i>	<i>y</i>	1
...	...	

2-Byte NP Table

Fig. 3. The data structure used by the proposed NP pre-filter.

regular expression matching and so on. Recall that in this paper we focus on simple PM in the hardware accelerator, which lies in the critical data path. In what follows we delineate each key component in the hardware accelerator, as well as the overall workflow.

4.2. The preprocessing stage

4.2.1. Construction of the NP table

Fig. 3 depicts the key data structure, i.e. the NP table, used by the proposed content pre-filter. Given a pattern set (in this example the set contains two patterns {pattern, attack}), the corresponding NP table enumerates all the strings that are NPs, according to Theorem 1. To avoid excessive overhead of table lookup, we use 2-byte NP in our prototype, which is proven to yield very good filtering effects according to our experimental results. For instance in Fig. 3, sub-string “ta” appears in pattern “attack”, so “ta” is not an NP. However, no suffix (>2 bytes) of “tb” appears in any pattern in the pattern set, so “tb” is an NP, and so forth. The algorithm for enumerating all the 2-byte NPs is straightforward and we omit its pseudo-code in this paper. Since the NP table is tiny enough (e.g. $2^{2 \times 8} = 64$ K bits when using 2-byte NPs) and each 2-byte string in the flow can be inspected independently, fast ASIC and multiple NP tables can be employed for parallel (and fast) NP table lookups.

4.2.2. Partitioning the pattern set

In Section 3.4, we briefed the idea of partitioning the pattern set into exclusive subsets, so that inspection may be performed on selective subsets to improve system throughput and reduce power consumption. To practically implement this idea, we bear the following goals in mind while designing the partitioning algorithm: (1) the number of “exclusive” subsets should be large enough to produce significant parallel speedup and power reduction; (2) the sizes of the subsets should be balanced; and (3) the partitioning process should take reasonable time to finish. Though (1) and (2) inevitably depend on the characteristics of a given pattern set, a smart partitioning algorithm can still push the results toward the optimum on a given pattern set.

We propose a heuristic partitioning algorithm running in polynomial time. The algorithm contains two high-level stages. Fig. 4 shows the pseudo code of the proposed partitioning algorithm. In

the first stage, the patterns are allocated one by one to either an existing subset or a newly created one based on the following principles.

- A. If the current pattern is exclusive with all previously allocated patterns, then a new subset will be created for the pattern.
- B. If the current pattern is not exclusive with some of the previously allocated patterns which are in the same subset, then the current pattern will be allocated to the subset.
- C. If the current pattern is not exclusive with some of the previously allocated patterns which belong to different subsets, then all the related subsets should be merged and the current pattern will be allocated to the merged subset.

In the second stage, the subsets with unbalanced sizes will be reformed into new ones with balanced sizes (correctness is guaranteed by Theorem 3). A classic greedy algorithm for the “knapsack problem” is used, where the subsets resulting from Stage 1 are regarded as the items and the target subsets are treated as the knapsacks.

4.3. Real-time pattern matching

4.3.1. Flow partition algorithm

According to the analysis in Section 3.1, the sizes of the generated segments should be as close to the TCAM window size (say, w bytes) as possible. Thus it is not necessary to partition each flow at every NP position, which will incur unnecessary overhead.

In this paper, a parallel partition algorithm (see Fig. 5) is proposed to partition the flows fast and efficiently. Suppose there are M parallel PM processing units. Then M locations (the length between two neighbored locations are $2w$) in the flow are inspected separately by the processing units. Based on the NP matching results of the M locations, a number of segments will be cut down at the NP positions. After that, the partitioned segment will be handled independently in parallel at the subsequent stages. If no NP is found at the initial locations, 1 byte will be shifted from previous locations, until NP is found or certain preset constraints are reached. This operation will be performed iteratively at each stage until either no segment is longer than $2w$ bytes or certain preset constraints are reached. At each stage, the new locations to be examined are selected by shifting 1 byte from each of the previous locations, respectively.

Fig. 6 gives an example of the above algorithm. In this example, the pattern set is {“pattern”, “attack”}, and assume that the flow content is “patternwithattacktheattackis...”. and $w = 8$. At the first stage, NP lookups are performed in parallel at the byte offsets $(w:w+1), (3w:3w+1), \dots, (2kw+w:2kw+w+1)$, simultaneously. A negative pattern hit is found at $(w:w+1)$, and hence the flow is split at “wi” to two segments as shown in Fig. 6. The procedure is similar for the second unit, and another negative pattern is found at $(3w+2, 3w+3)$. The partitioned segments will be further matched by TCAMs.

4.3.2. Pattern matching in the exclusive subsets

After the above pre-filtering process, the flow is partitioned into segments and ready for parallel inspections in the multi-block TCAM. Suppose that the pattern set has been divided into K exclusive subsets which are allocated to K blocks in the TCAM (one subset in each block). Generally speaking, for a flow segment with l ($l \geq w$) bytes, $(l - w + 1)$ input windows (each with w bytes) need to be inspected against 1 up to K exclusive subsets (i.e. TCAM blocks). Note that for a w -byte sub-string of the segment (see Fig. 7), the lookup time in the multi-block TCAM is

```

int ExclPart(PatternSet *PS, PatternSet *ExSubsets[], int n) {
Stage I:
  Init_all_subsets(ExSubsets[]);
  for each pattern in PS do {
    cs = next_empty_subset_in(ExSubsets[]);
    cs->add_pattern(pattern);
    for each non-empty subset in ExSubsets[] except cs do {
      if IsExclusive(cs, subset) == FALSE {
        Merge(cs, subset); //Merge subset into cs
        Clear(subset);
      }
    }
  }
Stage II:
  Sort the sets in ExSubsets in decreasing order of the cardinality
  for each non-empty subset in ExSubsets[] do {
    if subset is the n largest ones
      continue;
    else
      merge subset into the smallest one of the n largest subsets.
  }
}

```

Fig. 4. Pseudo-code for partitioning the pattern set.

```

Parallel_NP(byte *C, int length, int w) {
    i = 0; start = 0; end = 2w;
    for i = 0 to (length/2w - 1) do {
        if NP_Cut(C, start, end, i*2w, (i+1)*2w) {
            start = (i+1)*2w;
        } else {
            start = i*2w;
        }
        end = (i+2)*2w;
    }
}

NP_Cut(byte *C, int seg_start, int seg_end, int search_start, int
search_end) {
    for i from search_start to search_end do {
        if IsNP(C[i], C[i+1]) {
            TCAM_Match(C, seg_start, i)
            TCAM_Match(C, i+1, seg_end)
            return True
        }
    }
    return False
}

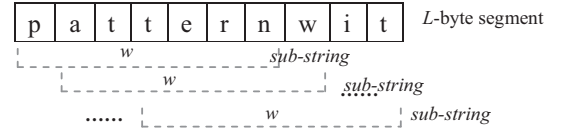
```

Fig. 5. Pseudo-code for the parallel NP matching scheme.

nondeterministic according to the “done-on-hit” principle introduced in Section 3. In the worst case $(l - w + 1)K$ TCAM lookups are required for the segment. We name these lookups the *Potential Lookup Tasklets* (PLTs) of the segment.

A TCAM controller is further deployed to dispatch and schedule the PLTs. In each TCAM cycle, the TCAM controller tries to dispatch as many pending PLTs as possible to the TCAM blocks concurrently. To maximize both the TCAM utilization and throughput, we set two dispatching rules: (1) only the PLTs corresponding to different TCAM blocks can be dispatched concurrently; and (2) one sub-string should be dispatched into multiple TCAM block/subsets (if necessary) only in a *sequential* manner, so that an early match in one pattern subset can prevent the lookups in other subsets according to the exclusive PM property. We observe that the above formulation is essentially one instantiation of the chessboard problem which frequently arises in the traditional switch fabric context. In this paper, a simple but effective round-robin-based scheduling scheme is used.

As depicted in Fig. 8, at the beginning, K w -byte ($K = 4$, $w = 7$ in this example) text sub-strings are extracted from the segment pool by the scheduler and loaded to the sub-string latches within the

Fig. 7. Term definitions: Text segment and its w -byte substrings.

TCAM Controller. During a TCAM cycle, the contents in the K latches will be matched against the K TCAM blocks, respectively. Whenever a match hit is observed, the matched sub-string will leave the system immediately as it needs no more TCAM lookups according to the exclusive pattern matching property. The scheduler will fetch another sub-string from the segment pool to replace the content in the corresponding latch. Meanwhile, the K round-robin switches will shift a number so that the unmatched sub-strings can be inspected in other pattern subsets in turn. If a sub-string remains unmatched when the corresponding switch shifts $K - 1$ times, it will leave the system with a *matching miss*. Finally, recall that only the matched sub-strings will be further delivered to the general-purpose processors for regular expression matching.

5. Performance evaluation

5.1. Theoretical analysis

The primary goal of the theoretical analysis is to show that the proposed scheme can achieve superior results with the aid of both NPM and EM, *regardless of* how dirty the incoming traffic is. Hence, the key input to the following analysis is a parameter that characterizes the degree of the dirtiness of the incoming traffic. To make the parameter most amenable to our specific TCAM-based mechanism, we devise the *dirty traffic ratio* (R_d) as the indicator of traffic dirtiness. Formally, R_d is defined as the number of inspections with PM hits (matches) to the total number of inspections required. For example, for a L -byte flow, we need $L - w + 1$ inspections; if we get m PM hits during the $L - w + 1$ inspections, the dirty traffic ratio is given by $m/(L - w + 1)$.

5.1.1. Additional overhead for NP matching

In the ideal case of hierarchical NP lookup (Fig. 5), an NP hit is found on each of the M positions being inspected, then only $\lceil L/w \rceil$ NP matches are required. For the general cases, let R_h ($0 \leq R_h < 1$) denote the hit rate of the NP matching, and let D_r (also defined as $nRound$ in Fig. 5) denote the number of stages/iterations in the “Hierarchical NP lookup” algorithm. Then the NP lookup times can be calculated as,

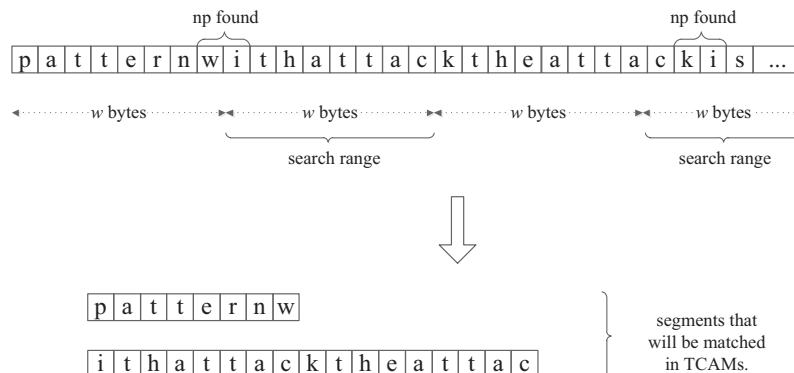


Fig. 6. A demonstration of the parallel NP lookup scheme.

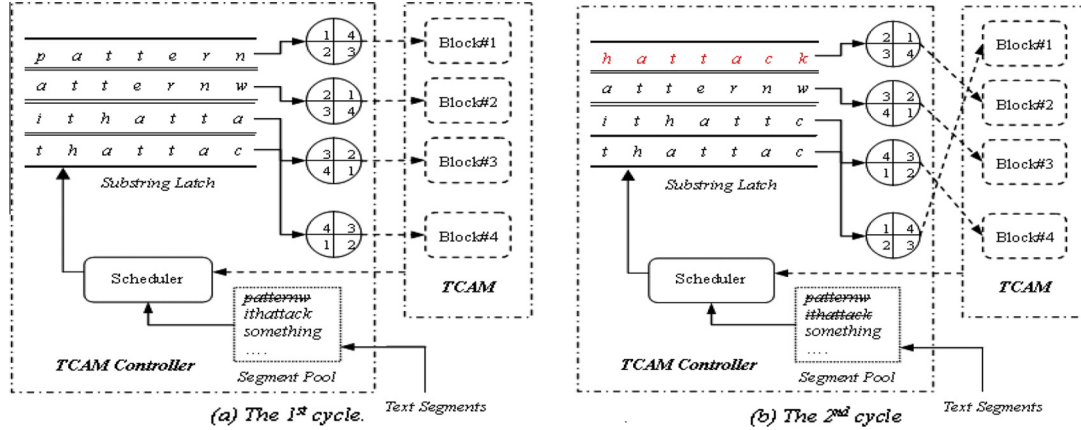


Fig. 8. An example of the TCAM controller. In this case, the content “pattern” in the 1st latch is reported as “matched in TCAM Block#1” in the 1st cycle. So in the next cycle, the latch will be reloaded, and the next (7-byte) sub-string to load should be “hattack” (in segment “ithattack”). In the 2nd cycle, the K Round-Robin switches all shift a number.

$$n_{NP} = \sum_{i=0}^{D_r-1} (1 - R_h)^i \times L/w = \frac{[1 - (1 - R_h)^{D_r}] \times L}{w \times R_h} \quad (5.1)$$

From Eq. (5.1), we can see that, the NP lookup overhead is disproportional to the window size w , and increases as R_h decreases. In the worst case that $R_h \rightarrow 0$, we have,

$$\lim_{R_h \rightarrow 0} \frac{[1 - (1 - R_h)^{D_r}] \times L}{w \times R_h} = \frac{L \times D_r}{w} \quad (5.2)$$

5.1.2. TCAM lookup savings

Note that, two factors collectively give rise to TCAM lookup overhead or power consumption: the number of input windows for inspections, and the number of TCAM entries triggered for each lookup. To capture the effects of both aspects, we invent a metric N_{pro} as the product of the number of inspections (denoted by N_{win}), and the average number of TCAM entries triggered for each inspection (denoted by N_{ent}). Suppose the original pattern set occupies $|S_p|$ TCAM entries (w bytes each) in total. Now we first evaluate N_{pro} for the Strawman approach where the flow is not partitioned using NP, nor is the pattern set partitioned. Given an L -byte flow, $L - w + 1$ inspections are needed, and each of them needs to match against all the $|S_p|$ entries: $N_{pro} = (L - w + 1) \times |S_p|$.

Now, with NP-based flow partition, we first reduce the number of lookups (in terms of number of inspection windows) as presented in Section 3.1:

$$N_{win} = \sum_{i=1}^n (l_i - w + 1) = L - n(w - 1) \quad (5.3)$$

where n denotes the number of segments the flow content is partitioned into. According to Eq. (5.1), we have,

$$n = n_{NP} \times R_h = [1 - (1 - R_h)^{D_r}] \times L/w \quad (5.4)$$

and according to both Eqs. (5.3) and (5.4), we obtain,

$$N_{win} = L \times \{1 - [1 - (1 - R_h)^{D_r}] \times (w - 1)/w\} \quad (5.5)$$

Then suppose with EM we partition the pattern set into K subsets, each taking up $|S_p|/K$ TCAM entries, and assume that each subset has equal probability to match/hit the input string. Given a dirty traffic ratio R_d , we further have,

$$\begin{aligned} N_{ent} &= \frac{|S_p|}{K} \left[\sum_{i=1}^{K-1} i \cdot \frac{R_d}{K} + K \cdot \left(1 - \frac{K-1}{K} R_d\right) \right] \\ &= |S_p| \frac{2K - (K-1)R_d}{2K} \end{aligned} \quad (5.6)$$

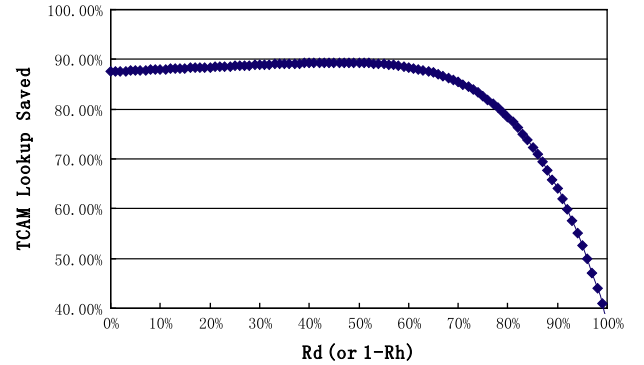


Fig. 9. Lookup saved as a function of traffic dirty ratio.

Therefore, the percentage of TCAM lookups that can be saved is estimated by,

$$\begin{aligned} N_{save\%} &= 1 - N_{win} \times N_{ent} / N_{pro} \\ &= 1 - \left\{ 1 - [1 - (1 - R_h)^{D_r}] \frac{w-1}{w} \right\} \frac{2K - (K-1)R_d}{2K} \\ &\quad \times \frac{L|S_p|}{(L - w + 1)|S_p|} \end{aligned} \quad (5.7)$$

From (5.7), we can see the favorable fact that,

$$\partial N_{save\%} / \partial R_h \geq 0 \quad (5.8)$$

and,

$$\partial N_{save\%} / \partial R_d \geq 0 \quad (5.9)$$

Also, it is quite straightforward that $dR_h/dR_d \leq 0$ (common sense). Eqs. (5.8) and (5.9) indicate that though the lookup reduction rate tends to decrease as the traffic become cleaner (i.e. $R_d \rightarrow 1$), however, since the NP hit ratio R_h will increase as the traffic becomes cleaner, we can still expect performance gain with the proposed scheme.

It is rather hard to work out the precise relationship between R_d and R_h , which heavily depends on the distributions of both the flow contents and the characters in the patterns set.

For the sake of simplicity, we assume that $R_h = 1 - R_d^3$, and given $w = 8$, $D_r = w - 1$, $K = 4$ and plot $N_{save\%}$ as the function of R_d as depicted in Fig. 9. Note that this is only used as a reference to

³ Though this assumption is simple, it meets at least three facts/constraints, (1) R_h and R_d move in the opposite direction, (2) when $R_h = 0$, $R_d = 1$; (3) when $R_h = 1$, $R_d = 0$.

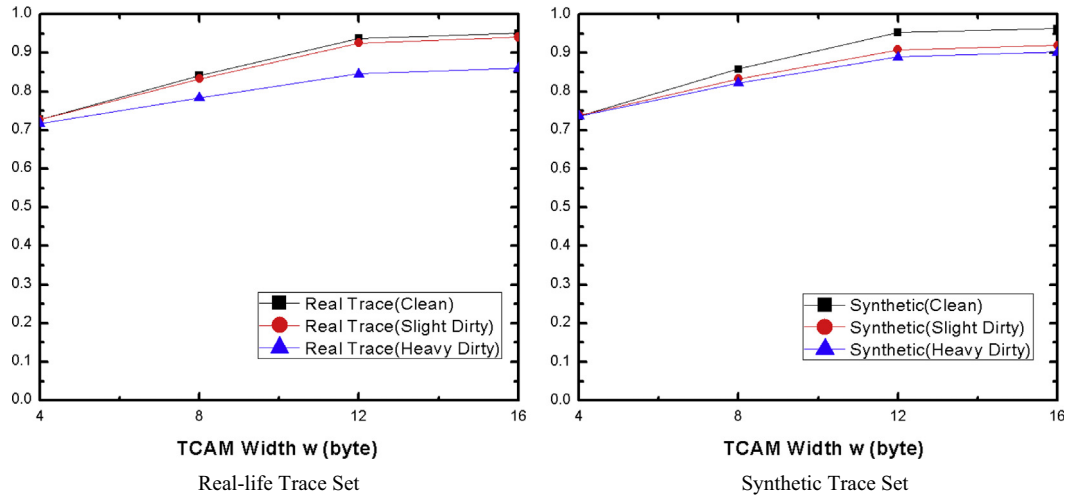


Fig. 10. TCAM-lookup-saved ratio with the SNORT-OFFI pattern set.

show the intuition. From the curve we can see that when R_d is less than 70%, the algorithmic gain in terms of “TCAM lookup saved” stays on a relatively high level, i.e. around 90%. The left-end (i.e. when $R_d \rightarrow 0$) of the curve indicates that the pre-filtering effect by NP matching avoids the majority of the unnecessary substrings being looked up in the TCAM blocks.

On the other hand, the curve drops significantly at the right-end when $R_d > 70\%$; this is unavoidable because match hits on the attacking patterns take place more and more frequently as R_d increases and the NP pre-filter becomes trivial since less and less positions in the flow can be found for partition.

However fortunately, since an early match in one pattern subset can prevent the lookups in other subsets according to the exclusive PM property. So as R_d increases, we can still benefit from exclusive PM. This explains why we can still observe a saving ratio around 40% at the extreme case when $R_d \rightarrow 100\%$ (which can be deduced according to Eq. (5.6) with ease).

5.2. Experimental analysis

5.2.1. Experiment setup

Performance may differ significantly across rule sets and traffic with different characteristics. The number of TCAM lookups is very important performance issue for using TCAM. The less TCAM lookups need, the less power consume and the higher line speed can be handled. There are various TCAMs in the market, such as NetLogic Microsystems [37] has the product NL9000 family which contain up to 1 million 40-bit entries, can perform up to 1200 million decisions per second, and. In this section, we are not evaluate the real power consumption, but the number of TCAM lookups which are related to it.

Only real-life data sets can shed light on the degree of such impacts. To make a horizontal comparison, we use both the pattern set from the official SNORT (SNORT-OFFI) website [2] (dated March 16, 2012) and the Bleeding SNORT rule set (SNORT-BLEE) [30] (dated June 7, 2009) in our experiments.

There are 61,234, and 63,527 2-byte negative patterns in these pattern sets, respectively. The long patterns are pre-partitioned into w -byte sub-patterns to fit in the Netlogic TCAM which is configured as 16 bytes width data slots, using the method proposed in [22]. For example in the case 3110 and 1326 sub-patterns are generated from SNORT-OFFI and SNORT-BLEE pattern set, respectively. In the case of SNORT-OFFI, the sub-patterns are grouped into 4 exclusive subsets which contain 964, 670, 670 and 710 sub-

patterns, respectively; and in the case of SNORT-BLEE, the sub-patterns are grouped into 4 exclusive subsets which contain 421, 287, 287 and 303 sub-patterns, respectively. We used a commodity desktop pc to run the partitioning algorithm. The laptop is with 2.0 GHz x86_64 CPU, 8G RAM. The execution of partitioning algorithm is less than two seconds, which is obviously not a significant overhead.

Both real-life traffic traces and synthetic traffic traces generated from real traces (injected with the dirty traffic patterns) are used in our experiments. The real-life traffic traces we used in the experiment include the 1st week (attack-free) and 5th week (with slight attacks) data set from MIT [36], and also the HoneyNet monthly data set from National University of Defense Technology (with relatively higher attack traffic, collected from April 16th to May 22nd, 2012). These sets of traces are used to demonstrate the performance of the proposed scheme in real-life.

Also, in order to be able to make a horizontal comparison and track the performance change when the traffic become dirty from clean, we also introduce two synthetic traces originated from real traces⁴ into our experiments. The dirty traffic ratio, i.e. R_d , of them are 30% and 50%, respectively.

In the following three subsections, we will investigate the three key performance metrics, respectively, namely, TCAM lookup saved, speedup ratio and NP lookup required.

For each one of these performance metrics, we arrange the results in two figures, one for each of the two pattern sets (as mentioned before). In each figure, six performance curves will be shown, and each one of them represents the performance result for one of the 6 traffic traces introduced above. In order to make it easier for comparison, we divide the 6 curves into two groups and depict them in two sub-figures, respectively, namely, one for the real-life traces and the other for the synthetic ones.

5.2.2. TCAM lookup saved

Figs. 10 and 11 depict the percentage of TCAM lookup saved, when using the proposed scheme (compared with the traditional approach, e.g. [22]).

Firstly we note that around 30–90% TCAM lookups can be saved in different experiments and configurations, indicating that the

⁴ The attack patterns from SNORT pattern set are randomly chosen and inserted into each packet in the clear trace collected from the 40Gbps backbone of China Telecom, Guangdong Province China, during the 2010 Asia Game.

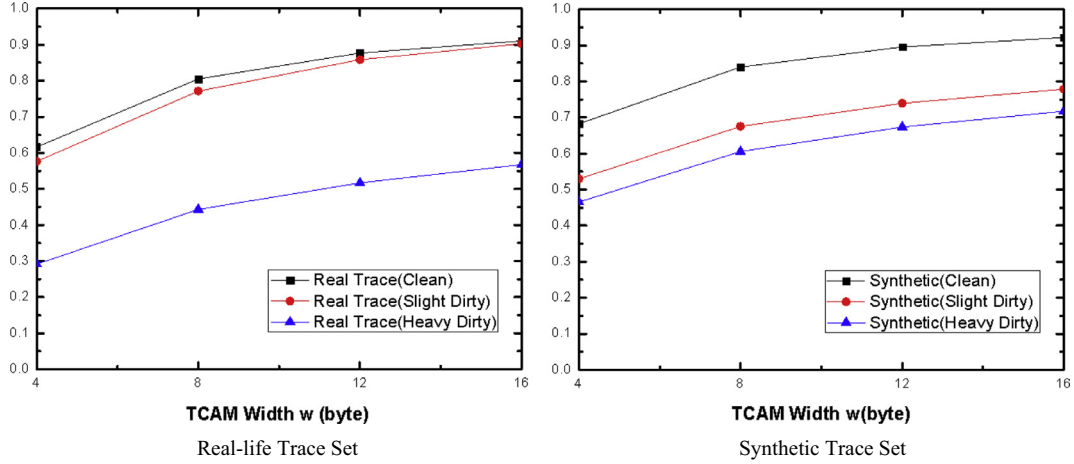


Fig. 11. TCAM-lookup-saved ratio with the SNORT-BLEE pattern set.

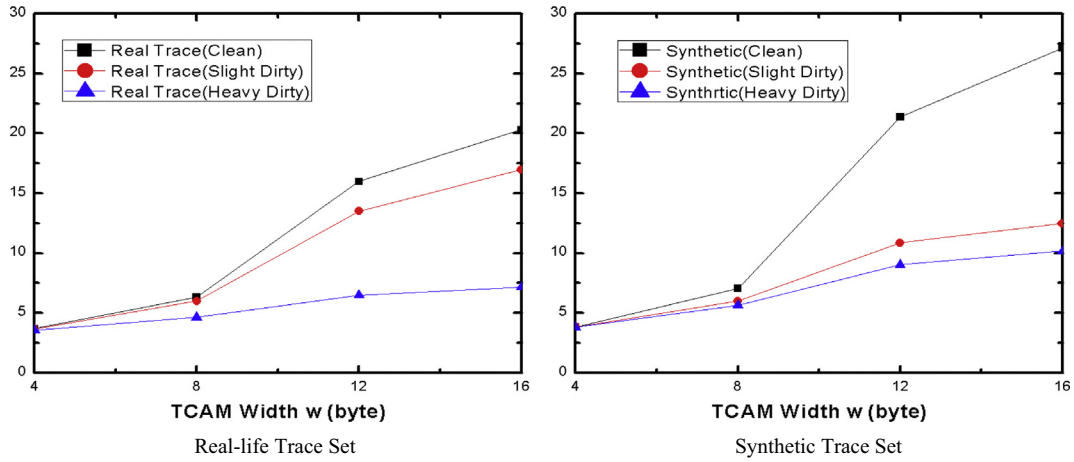


Fig. 12. Speedup ratio with the SNORT-OFFI pattern.

proposed scheme indeed optimizes the performance by saving TCAM lookups.

Secondly, we can see obvious trend of improvement with the increase of the inspection window size, w . This is coherent with our analysis in Section 3.1 and it is almost the dominating factor of the TCAM lookup reduction. In contrast, the traffic dirty ratio (R_d) has relatively slighter impact on the achievement. For example the percentage of TCAM lookup saved are 73%, 91%, 94% and 95% for clean traffic trace when $w = 4, 8, 12, 16$ respectively, while the percentage are 72%, 82%, 86% and 87% for heavy dirty traffic trace when $w = 4, 8, 12, 16$ respectively. When R_d varies from 0% (the real-life clean trace) to 50% (the dirtiest synthetic trace), TCAM lookup saved differ slightly from 81% to 96%, when $w = 16$. And this benefits from the complementarity between NPM and EM, as discussed in Section 3.5, i.e., the dirty traffic content leads to less TCAM lookup reduction by NPM, however at the same time, leads to more by EM.

Furthermore, when comparing the results in both Figs. 10 and 11, we can see that pattern sets (sizes) have relatively less impacts on the TCAM lookup reduction. Note that the reduction in TCAM lookup is presented as a relative ratio (to the baseline/traditional method). According to the results, we found that even better results can be achieved for the larger pattern set, SNORT-OFFI, indicating that the tougher the case is, the better performance can be achieved.

5.2.3. Speedup ratio

Figs. 12 and 13 depict the speedup ratio by using the proposed scheme. Speedup ratio is defined as $K \times N_{TCAM_lookup_before} / N_{TCAM_lookup_after}$, where $N_{TCAM_lookup_before}$ and $N_{TCAM_lookup_after}$ denote the number of inspections to be looked up in the TCAM before and after using the proposed approach, respectively⁵; K denotes the number of TCAM blocks or the number of exclusive pattern subsets. Note that parallelism is introduced when the TCAM is partitioned into blocks and inspections are performed in parallel ($K = 4$ in the deployed experiments). Speedup ratio indicates the performance gain in terms of PM throughput. We can see that for all cases significant performance gains ($2.5\text{--}27\times$) are achieved, especially for a larger TCAM window size, w . From the results, we note that better speedup factor can be achieved for cleaner traffic, which mainly owe to the NP scheme (please also refer to Section 3.5).

5.2.4. Overheads

Figs. 14 and 15 show the overheads when applying the proposed pattern matching scheme, i.e., additional memory lookup required to identify the negative patterns. The overhead is measured in terms of the ratio of identifying the negative patterns to

⁵ It is assumed that the working frequency of the multi-blocked TCAM is the same as the commodity ones. And actually it can be higher and result in additional performance gain.

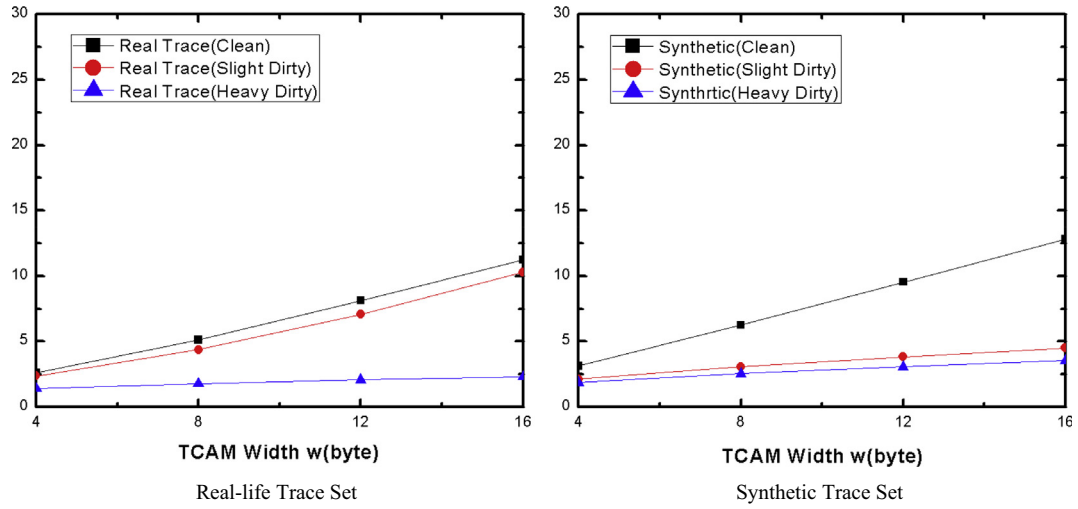


Fig. 13. Speedup ratio with the SNORT-BLEE pattern.

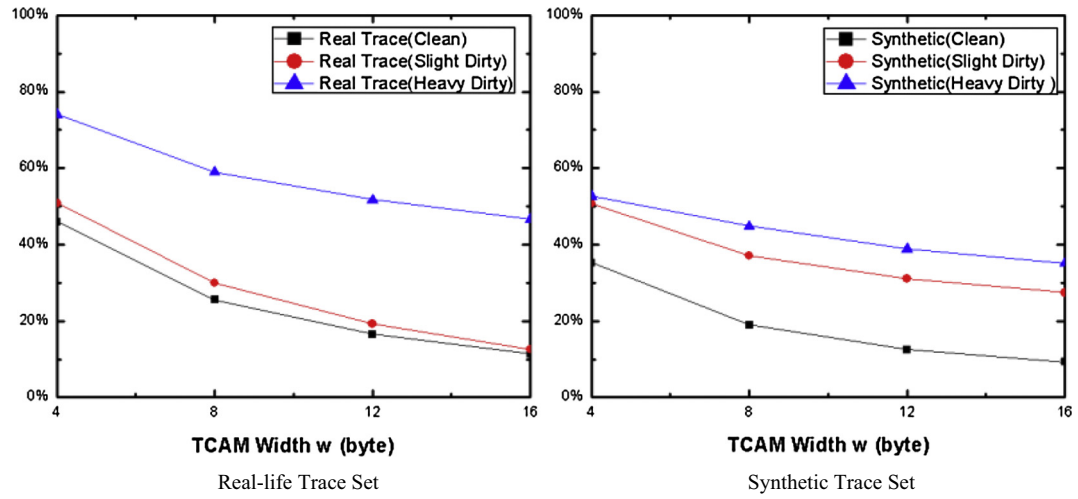


Fig. 14. NP lookup required with the SNORT-OFFI pattern.

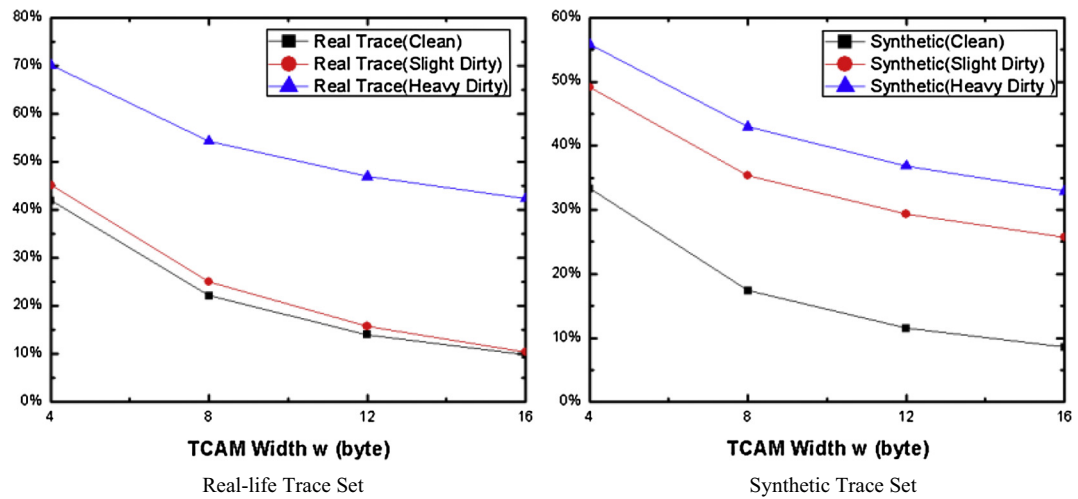


Fig. 15. NP lookup required with the SNORT-BLEE pattern.

TCAM lookups required, which can be easily used to make a comparison with the performance gain (speedup or TCAM lookup reduced).

According to the figures, we found an obvious trend of the overheads that it decreases significantly as the inspection window size increases. This is also coherent with the estimation given by Eq. (5.2). Correlating the data of both TCAM lookup saved and overheads in terms of NP lookups required, taking the case of real-life trace with slight attacking traffic (using the SNORT-OFFI pattern set) for instance, when $w = 16$, we can see that the proposed scheme trades 11% 2-byte NP table lookups (see Fig. 14) for 93% 16-byte TCAM lookups (see Fig. 15), which is obviously a promising result. Note that the 2-byte NP lookups can be performed efficiently on the accelerator chip in parallel.

6. Conclusions

This paper aims to enhance the throughput and scalability simultaneously for pattern matching operations, which tend to be the performance bottleneck of NIDSes. We focus on exploring efficient parallelism in a TCAM-based PM scheme, which can deliver superior performance compared to pure software-based approaches. We observe and prove that, by partitioning flows into segments and performing parallel inspection at the flow-segment level, we achieve both better load balancing and significantly reduced number of TCAM lookups. To enable flow partition while preserving PM correctness, we propose the novel concept of “negative patterns”, by which flow partition will not miss malicious patterns spread across multiple packets in a flow.

Furthermore, we observe that, not all the rules in a pattern set (thus all the corresponding TCAM entries) need to be invoked simultaneously for matching against a certain TCAM input. Instead, we introduce the “Exclusive Matching” mechanism to partition the original pattern set into “exclusive” sub-tables, so that a given input can only result in matches in at most one of these exclusive sub-tables. In this way, only a subset of TCAM entries need to be invoked for each input simultaneously, dramatically reducing the TCAM power consumption and maximizing scalability.

Our theoretical analysis results show that significant reduction of TCAM lookups can be achieved regardless of the traffic content. We further launch extensive experiments using real traffic traces, and demonstrate that up to 90% expensive TCAM lookups can be saved, at the cost of merely 10% additional 2-byte index table lookups in the SRAM, in the scenarios with either clean or dirty traffic; as a promising result, significant speedup (2.5–27 \times) are achieved for PM performance. Finally, we anticipate the philosophies behind *negative patterns* and *exclusive matching* can obtain more general applicability to other parallel systems.

References

- [1] 2005 FBI Computer Crime Survey. <<http://www.digitalriver.com/v2.0-img/operations/naievigi/site/media/pdf/FBIccs2005.pdf>>.
- [2] SNORT – the de facto standard for intrusion detection/prevention. <<http://www.snort.org>>.
- [3] A.V. Aho, M.J. Corasick, Efficient string matching: an aid to bibliographic search, *Commun. ACM* 18 (6) (1975) 333–340.
- [4] R.S. Boyer, J.S. Moore, A fast string searching algorithm, *Commun. ACM* 20 (10) (1977) 762–772.
- [5] M. Becchi, P. Crowley, A hybrid finite automaton for practical deep packet inspection, in: ACM International Conference on emerging Networking Experiments and Technologies Conference (CoNEXT), 2007.
- [6] Kun Huang, Linxuan Ding, Gaogang Xie, Dafang Zhang, Alex X. Liu, Kavé Salamatian, Scalable TCAM-based regular expression matching with compressed finite automata, in: ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS), 2013.
- [7] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, J. Turner, Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection, ACM SIGCOMM, 2006.
- [8] R. Smith, C. Estan, S. Jha, XFA: faster S signature matching with extended automata, in: IEEE Symposium on Security and Privacy (S&P), 2008.
- [9] N. Tuck, T. Sherwood, B. Calder, G. Varghese, Deterministic memory-efficient string matching algorithms for intrusion detection, in: IEEE INFOCOM, 2004.
- [10] F. Yu, Z. Chen, Y. Diao, T.V. Lakshman, R.H. Katz, Fast and memory-efficient regular expression matching for deep packet inspection, in: ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS), 2006.
- [11] S. Yun, An efficient TCAM-based implementation of multi-pattern matching using covered state encoding, *IEEE Trans. Comput.* 61 (2) (2012) 213–221.
- [12] C.J. Coit, S. Staniford, J. McAlerney, Towards faster string matching for intrusion detection or exceeding the speed of SNORT, in: DARPA Information Survivability Conference and Exposition (DISCEX II'01), 2001.
- [13] C. Kruegel, F. Valeur, Stateful intrusion detection for high-speed networks, in: IEEE Symposium on Research on Security and Privacy, 2002.
- [14] Chia-Mei Chen, Ya-Lin Chen, Hsiao-Chung Lin, An efficient network intrusion detection, *Comput. Commun.* 33 (4) (2010) 477–484.
- [15] Phurivit Sangkatsanee, Naruemon Wattanapongsakorn, Chalermopol Charnsripinyo, Practical real-time intrusion detection using machine learning approaches, *Comput. Commun.* 34 (18) (2011) 2227–2235.
- [16] D. Pao, N. Lam Or, C. Ray, C. Cheung, A memory-based NFA regular expression match engine for signature-based intrusion detection, *Comput. Commun.* 36 (10–11) (2013) 1255–1267.
- [17] Bleeding snort. Blog about computer and network security and protection. <<http://www.bleedingsnort.com>>.
- [18] N.S. Artan, H.J. Chao, TriBiCa: trie bitmap content analyzer for high-speed network intrusion detection, in: IEEE INFOCOM, 2007.
- [19] S. Dharmapurikar, P. Krishnamurthy, T.S. Sproull, J.W. Lockwood, Deep packet inspection using parallel bloom filters, *IEEE Micro* 24 (2004).
- [20] S. Dharmapurikar, J.W. Lockwood, Fast and scalable pattern matching for network intrusion detection systems, *IEEE J. Select. Areas Commun.* 24 (10) (2006) 1781–1792.
- [21] C.R. Clark, D.E. Schimmel, Scalable pattern matching for high speed networks, in: IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04), 2004.
- [22] F. Yu, R.H. Katz, T.V. Lakshman, Gigabit rate packet pattern-matching using TCAM in: Proceedings of the Network Protocols, 12th IEEE International Conference on (ICNP'04) IEEE Computer Society, 2004, pp. 174–183.
- [23] B.L. Hutchings, R. Franklin, D. Carver, Assisting network intrusion detection with reconfigurable hardware, in: IEEE symposium on Field-Programmable Custom Computing Machines (FCCM'02), Napa, USA, 2002.
- [24] Y.H. Cho, W.H. Mangione-Smith, Fast reconfiguring deep packet filter for 1+ gigabit network, in: IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), 2005.
- [25] J.v. Lunteren, High-performance pattern-matching engine for intrusion detection, in: IEEE INFOCOM, 2006.
- [26] S. Dharmapurikar, J. Lockwood, Fast and scalable pattern matching for content filtering, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2005.
- [27] G. Vasilidis, M. Polychronakis, S. Ioannidis, MIDEA: a multi-parallel intrusion-parallel intrusion detection architecture, in: ACM Conference on Computer and Communications Security (CCS), 2011.
- [28] K. Zheng, C. Hu, H. Lu, B. Liu, A TCAM-based distributed parallel IP lookup scheme and performance analysis, *ACM/IEEE Trans. Network.* 14 (4) (2006) 863–875.
- [29] Chad R. Meiners, Jignesh Patel, Eric Norige, Eric Torng, Alex X. Liu, Fast regular expression matching using small TCAM, *IEEE/ACM Trans. Network.* 22 (1) (2014) 94–109.
- [30] K. Peng, S. Tang, M. Chen, Chain-based DFA deflation for fast and scalable regular expression matching using TCAM, in: ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems, 2011.
- [31] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, E. Porat, On finding an optimal TCAM encoding scheme for packet classification, *IEEE INFOCOM 2013* (2013) 2049–2057.
- [32] Kirill Kogan, Sergey Nikolenko, Ori Rottenstreich, William Culhane, Patrick Eugster, SAX-PAC (Scalable And eXpressive Packet Classification), in: SIGCOMM 2014, ACM, New York, NY, USA, pp. 15–26.
- [33] K. Peng, Q. Dong, M. Chen, TCAM-based DFA deflation: a novel approach to fast and scalable regular expression matching, in: 19th International Workshop on Quality of Service, (IWQoS), 2011.
- [34] Ziping Cai, Zhiyun Wang, Kai Zheng, Jiannong Cao, A distributed TCAM coprocessor architecture for integrated longest prefix matching, policy filtering and content filtering, *IEEE Trans. Comput.* 62 (3) (2013) 417–427.
- [35] N. Dukkkipati, N. McKeown, Why flow-completion time is the right metric for congestion control, *ACM SIGCOMM Comput. Commun. Rev.* 36 (2006) 59–62.
- [36] MIT DARPA Intrusion Detection Data Sets. <<http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html>>.
- [37] NetLogic Microsystems. <<http://www.broadcom.com/products/Knowledge-Based-Processors/Layers-2-4>>.