A Memory-Efficient Parallel String Matching Architecture for High-Speed Intrusion Detection

Hongbin Lu, *Student Member, IEEE*, Kai Zheng, *Student Member, IEEE*, Bin Liu, *Member, IEEE*, Xin Zhang, and Yunhao Liu, *Senior Member, IEEE*

***Abstract*—The ability to inspect both packet headers and payloads to identify attack signatures makes network intrusion detection system (NIDS) a promising approach to protect Internet systems. Since most of the known attacks can be represented with strings or combinations of multiple substrings, string matching is a key component, as well as the bottleneck in NIDS to address the requirement of constantly increasing capacity. We propose a memory-efficient multiple-character-approaching architecture consisting of multiple parallel deterministic finite automata (DFAs), called TDP-DFA. By employing efficient representations for the transition rules in each DFA, TDP-DFA significantly reduces the complexity. We also present a novel scheme to share the storage of transition rules among multiple DFAs, substantially decreasing the total storage cost, and avoiding the cost increase being proportional to the number of DFAs. We evaluate this design through theoretical analysis and comprehensive experiments. Results show that TDP-DFA is able to meet the critical requirement of OC-768 wirespeed processing, as well as constituting a promising way for scaling up to cope with throughput over 100 Gb/s in the future.**

***Index Terms*—Computer network security, finite automata, parallel processing, site security monitoring, string matching.**

I. Introduction

As PROLIFERATION of Internet applications increases, security becomes a problem within network solutions. Intruders attempt to break into publicly accessible victim systems to misuse the functionality provided. Traditional network-based security devices such as firewalls, performing packet filtering on packet headers only, fail to identify attacks that use unsuspicious headers. By inspecting both packet headers and payloads to identify attack signatures, network intrusion detection system (NIDS) is able to discover whether hackers/crackers are attempting to break in or launch a denial of service (DOS) attack.

Because most of the known attacks can be represented with strings or combinations of multiple substrings, string matching is one of the key components in NIDS. String matching in NIDS is computationally intensive in that, unlike simple packet classifications, NIDS needs to scan both the headers and the payloads of each incoming packet for thousands of suspicious strings. Worse, the string lengths are variable. As a result, string matching has become the bottleneck in NIDS to address the requirement of constantly increasing capacity. In a popular NIDS, such as Snort [1], 70% of total execution time and 80% of instructions are for string matching routines [2]. Another challenge for string matching mechanism is its vulnerability to worst-case intended attacks. With knowledge of the rule set, attackers can easily overload the string-matching operations by generating worst-case input scenarios [2]. Thus, successfully dealing with the worst-case scenarios is of great importance for the string matching approach in NIDS.

Having been extensively studied for decades, existing solutions to string matching continue to suffer low efficiency and high storage consumption. Most of the previous works employ a single-character-approaching algorithm, in which the worst-case performance is strictly proportional to the memory frequency. Consequently, a NIDS dealing with OC-768 (40 Gb/s) capacity would require a memory accessing frequency of nearly 5 GHz, which is impractical with current technologies. Even with recent advances, the evolution of memory frequency is far slower than the increasing interface wirespeed [3].

Recently, researchers propose to increase the system throughput by adopting embedded memory techniques based on the following observations. First, using embedded memory will greatly decrease the accessing latency due to the elimination of I/O drivers and external wires. With modern application specific integrated circuit (ASIC) technology, high-speed embedded memory can be compiled with custom logic on the same silicon die. For example, embedded static RAM (SRAM) blocks are able to run as fast as 1 GHz [4]. Second, vast parallelism inside the chip can be exploited, which means hundreds of embedded memory blocks can work together without limits from off-chip bus bandwidth. However, before we can successfully use embedded memory to improve the string matching in NIDS, the necessary memory size for string matching must be minimized since embedded memory is very expensive. But in previously proposed approaches, the desired increase in string matching speed is linear with the extra storage space required, leaving only an exorbitant way to gain speed with memory space proportionally.

To address this issue, we propose a memory-efficient multiple-character-approaching scheme consisting of multiple parallel deterministic finite automata (DFAs), called transition-distributed parallel DFAs (TDP-DFA). The major contributions of this work are as follows.

1) From observations of overlapping relationships among real-life NIDS signatures, efficient representations for the

Manuscript received September 1, 2005; revised April 1, 2006. This work was supported in part by the National Science Foundation of China under Contract 60373007 and Contract 6057312, in part by the Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant 20040003048, and in part by the Tsinghua Basic Research Foundation (JCpy2005054).

H. Lu, K. Zheng, B. Liu, and X. Zhang are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: lu-hb02@mails.tsinghua.edu.cn; zk01@mails.tsinghua.edu.cn; liub@tsinghua.edu.cn; z-x02@mails.tsinghua.edu.cn).

Y. Liu is with the Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: liu@cs.ust.hk).

Digital Object Identifier 10.1109/JSAC.2006.877221

transition rules are designed to reduce the complexity of each DFA.

2) Leveraging the correlations among the inputs to the parallel DFAs, a novel scheme to share the storage of transition rules among multiple DFAs is presented, thus significantly reducing the total storage cost, and avoiding a cost increase proportional to the number of additional DFAs required.
3) Various performance metrics of this design are studied via theoretical analysis and comprehensive experiments. The results show that to meet the demand of OC-768 wirespeed (40 Gb/s) processing, less than 58 kB of on-chip binary content addressable memory (BCAM) and 188 kB of embedded SRAM are needed, and such requirements can be easily supported by today's on-chip memory techniques.

The remainder of this paper is organized as follows: Section II reviews the related works. Section III presents the details of TDP-DFA and analyzes its performance. Section IV shows the experimental results. Finally, a conclusion is drawn in Section V.

II. Related Works

Though many software-based string matching algorithms [5]–[9] have been developed in the past decades, it is generally believed that they cannot achieve multigigabit throughput. Recently, many hardware-based algorithms have been proposed, of which many solutions are based upon field-programmable gate array (FPGA) [10]–[17].

Due to the abundance of parallelism and programmability in FPGA, researchers can instantiate a large number of parallel processing units, with each unit being in charge of one or several patterns, thus significantly enhancing the throughput. However, such approaches require time-consuming recompilation and reconfiguration of the FPGA upon any change of the rule set.

To support fast dynamic update and remain high-performance, many algorithms resort to adopting fast memory component, e.g., SRAM, content addressable memory (CAM), or embedded memory. Yu *et al.* proposed a ternary content addressable memory (TCAM)-based scheme [18] that can successfully handle complex patterns such as arbitrarily long patterns, correlated patterns, and patterns with negation. The major concern is that every single-character processing requires a TCAM access on average, which means even an 800 MHz TCAM is employed, the capacity upper bound is limited to 6.4 Gb/s. Besides, many efforts have been made to optimize the Aho–Corasick algorithm [19] for ASIC solutions in order to benefit from its deterministic performance. Tuck *et al.* presented two schemes employing bitmap compression or path compression [20]. Aldwari *et al.* broke down the rule set into small ones according to the rules' associative packet type information in Snort database [21]. Tan *et al.* decomposed the whole DFA to eight sub-DFAs, with each in charge of a certain bit in every incoming character [22]. Cho proposed a memory-efficient string matching algorithm using hashing scheme [23]. These solutions dramatically reduce Aho–Corasick algorithm's storage cost and make it possible to implement the algorithm with on-chip memory. However, they can only process at most one character per clock cycle.

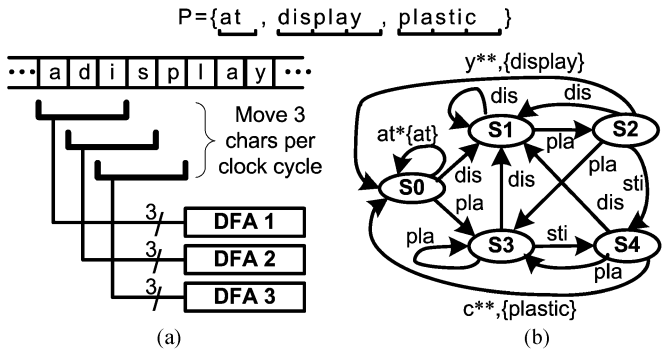


Fig. 1. The multiple-character-approaching model ($w = 3$).

With Bloom filters implemented with on-chip memory, Dharmapurikar *et al.* presented two detecting schemes [24], [25] that can process multiple characters per clock cycle and attain average throughput up to multigigabit with moderate memory consumption. But the proposed schemes are vulnerable to malicious attacks since in the worst case they must frequently access the relatively slow off-chip SRAMs to launch exact string comparisons. Tripp also presented a multiple-character-approaching solution [26] which splits the entire DFA into many tiny DFAs, where each DFA only searches for one or several signatures to cut down the sharply increasing complexity of the DFA caused by broadening the input width. To resolve the alignment problem, however, most of the signatures must be decomposed to multiple overlapped substrings, leading to a proportional relationship between the storage cost and the number of characters processed per clock cycle. In contrast, our proposed TDP-DFA minimizes the storage cost for each used DFA without affecting its performance, as well as keeping the overall storage cost sublinear in the throughput.

III. TDP-DFA Architecture

A. Basic Multiple-Character-Approaching Model

Aiming at processing multiple characters per clock cycle, we present a model as illustrated in Fig. 1, where w characters $(w = 3)$ are regarded as a *token*, and then each signature is decomposed into one or more tokens. Note that appropriate number of wildcards, i.e., "*"s may be padded to make the length of the signature an exact multiple of w. According to the Automata Theory [27], a corresponding NFA can be constructed and converted to a DFA for detecting the token sequences. As is the case in Fig. 1(a), the boundary of a single input window may not be aligned with the starting character of the pattern, hence we need to deploy w DFAs for detection in parallel.

However, such a naïve scheme lacks scalability and memory efficiency in that: 1) when constructed from a large string set, the DFA may contain too many transition rules and 2) multiple identical DFAs need be deployed. In what follows, we resolve these two concerns in Sections III-B and III-C, respectively.

B. Optimized Implementation for a Single DFA

In this section, we will exploit the redundancy among the transition rules within a single multiple-character-approaching

TABLE I
ANALYSIS OF THE TRANSITION RULE SET FOR SNORT 2.3.3

	w=10	w=20	w=30
Num. of transitions in uncompressed rule set	2973895	332564	168471
Num. of transitions to F	2971443	330423	166292
Num. of transitions to other than F	2452	2141	2179
Num. of transitions in compressed rule set	3841	2662	2505
Num. of characters in compressed rule set	39727	34089	34741
Num. of characters in all signatures	33793	33793	33793

DFA in pursuit of a storage-optimized implementation. We extract signatures from the Snort database and generate the corresponding DFA. Let F denote the set of states which can be reached from the starting state via only one transition. For example, in Fig. 1(b), $F = \{S_1, S_3\}$. The first three rows of Table I show the data measured under three different configurations. From the produced DFA, we learn that the number of transitions is extremely large and most of the transitions point to the states in F. Moreover, all transitions pointing to "the same state" in F are labeled with the same string. This case is also intuitively shown in Fig. 1(b), where all the states can jump to S_1 with the transitions labeled as "dis." The feature is explained as follows.

Let $\mathrm{TSeq}(S)$ denote the shortest token sequence able to drive the DFA from the root state to state S, e.g., $\mathrm{TSeq}(S_4)$ ="pla,sti" in Fig. 1(b). Let $\langle \mathrm{TSeq}(S_x), A\rangle$ denote the concatenation of $\mathrm{TSeq}(S_x)$ and token A. Then, in the DFA, $\mathrm{nextstate}(S_x, A) = S_y$ [1] if and only if (iff) $\mathrm{TSeq}(S_y)$ is the longest suffix of the sequence $\langle \mathrm{TSeq}(S_x), A\rangle$ among the "TSeq"s of all states. For example, in Fig. 1(b), $\mathrm{nextstate}(S_1,$ "pla") is S_2 instead of S_3 since $\mathrm{TSeq}(S_2)$ is a longer suffix of $\langle \mathrm{TSeq}(S_1),$ "pla"$\rangle$ than $\mathrm{TSeq}(S_3)$. For any two states S_x and S_y, it is obvious that the probability when $\mathrm{TSeq}(S_x)$ has a common token sequence with $\mathrm{TSeq}(S_y)$ turns very low when w is large. Therefore, $\mathrm{TSeq}(S_y) = A$ always holds when $\mathrm{nextstate}(S_x, A) = S_y$, which explains why most transition rules point to the states in F.

According to the feature just discussed, the transition rule set can be compressed by enabling wildcards to appear in the "current state" fields of transition rules. For example, state S_0, S_1, S_2, S_3, and S_4 in Fig. 1(b) all have transitions to S_1 labeled with "dis"; so these transitions can be replaced by a single compressed transition rule "$\mathrm{nextstate}(*, \mathrm{dis}) = S_1$". Introducing wildcards incurs the necessity of assigning to transition rules priorities as follows.

1) Transition rules containing no wildcards, e.g., "$\mathrm{nextstate}(S_1,$ "pla"$) = S_2$," have the highest priority.
2) Transition rules containing one or more wildcards in the labels, e.g., "$\mathrm{nextstate}(S_2,$ "y**"$) = S_0$," have the lowest priority.
3) Transition rules containing wildcards in the "current state" fields, e.g., "$\mathrm{nextstate}($*,"dis"$) = S_1$," have mid-priority.

[1]"$\mathrm{nextstate}(S, I) = N$" stands for the transition rule with "current state" field S, "input" field I, and "next state" field N.

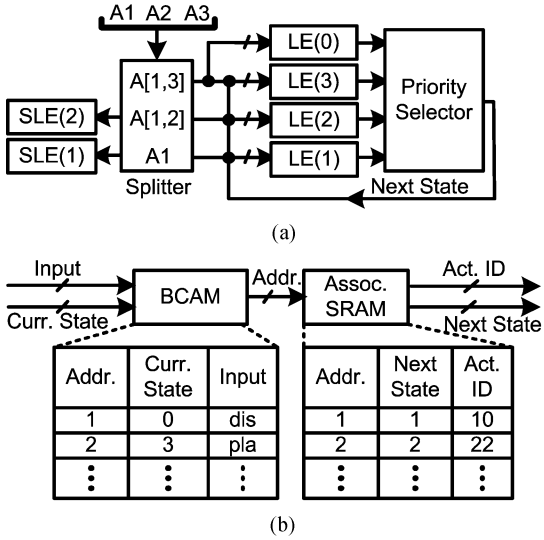


Fig. 2. Structure and implementation of a DFA.

As shown in the last three rows in Table I, after applying the compression scheme, the numbers of transitions are greatly decreased, and the counts of the characters included in these transitions are also reduced to the same magnitude as that of the overall signature set, indicating that the compression eliminates most of the redundancy.

Fig. 2(a) shows the implementation of a DFA, where $w = 3$ is still taken. The implementation consists of multiple lookup engines (LEs), of which LE(i) is responsible for matching against transition rules with "input" field length i, where $1 \leq i \leq w$, and LE(0) is further deployed for matching transition rules whose "current state" fields contain wildcards, so as to eliminate the redundancy in the transition rules. Note that in Fig. 1(b), the starting and ending states of the transition rules for short signatures (whose lengths are less than w, e.g., "at", and the rest can be called long signatures) are all S_0, so they can be directly matched, and thus extracted from the DFA to save storage for the two state IDs. In Fig. 2(a), the SLE(1) and SLE(2) are designed for detecting signatures with length 1 and 2, respectively.

The workflow of each DFA is described as follows. In each clock cycle, the *Splitter* receives an input from the window, and generates w prefixes with lengths ranging from 1 to w. Each of them is then simultaneously sent to the LEs or SLEs with corresponding lengths. Note that every LE also needs the current state as another input. When any entry is matched, the corresponding LE or SLE sends out the entry's associated *Action ID* to activate specified operations, or the *next state*, or both. Receiving the "next states" from all LEs, the *Priority Selector* always chooses the one with the highest priority for usage as the "current state" in the next clock cycle.

We then introduce the implementation of the LEs. As illustrated in Fig. 2(b), LE(3) is composed of a BCAM [28] along with an associated SRAM, and stores only the transition rules with length 3. The BCAM can search all of its entries in one clock cycle and report the address of the first matching one if any, driving the associative SRAM to send out the Action ID and the "next state." The other LEs and SLEs are built and work in a similar fashion except for a little variation: LE(0) lacks

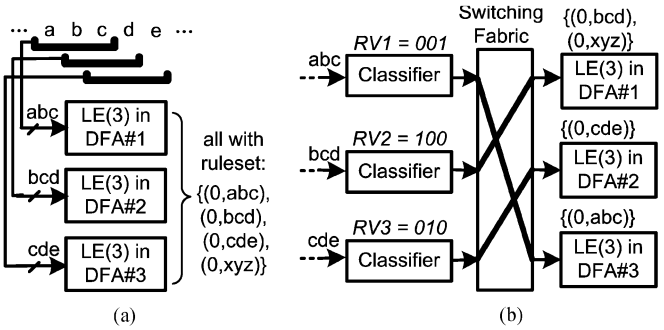


Fig. 3. An example of CDLE with $w = 3$.

the "current state" field; LE(1) and LE(2) have different widths of the "input" fields; besides, the SLEs vary in both different input widths and lack of the two-state fields. Although the DFA structure can be more easily implemented with TCAM, we still choose BCAM because it can benefit the resource sharing mechanism introduced in Section III-C.

C. Resource Sharing Among Isomorphic LEs

In this section, we first introduce a key observation that when leveraging the relationships among the parallel input strings, resource sharing is available among isomorphic LEs of different DFAs. Based on this observation, the overall storage cost of the system can be substantially reduced.

1) Composite Distributed Lookup Engine (CDLE): Given a set of signatures {abc, bcd, cde, xyz}, a multiple-character-approaching model can be constructed according to the preceding descriptions. For illustration, we take $w = 3$ and group the LE(3)s of the 3 DFAs together, as shown in Fig. 3(a), where LE entries are represented in the form of "('current state' number, 'input' field)," Note that the input windows of the three LE(3)s are mutually overlapped. This poses a restriction that not arbitrary two strings with length 3 can appear simultaneously in the three input windows. For example, "bcd" and "xyz" cannot emerge within one clock cycle, and hence the corresponding entry (0,bcd) and (0,xyz) will never be matched simultaneously in Fig. 3(a); in contrast, since the suffix "c" of string "abc" is identical with one of the prefixes of string "cde", entry (0,abc) and (0,cde) may be matched at the same time, e.g., in Fig. 3(a). We say two such entries are *correlated* to each other, and the method to recognize the *correlation* is elaborated in Appendix I. To utilize this feature for reducing the storage requirement, a scheme called CDLE is proposed in Fig. 3(b).

Distinguished by an explicit *Switching Fabric* bridging between the LEs and w parallel inputs, each input is preprocessed by one *Classifier* that produces a w-bit result vector (RV) for its input, where the ith bit (named $\mathrm{RV}[i]$) is set iff $\mathrm{LE}(w)$ in DFA# i contains an entry matching the corresponding input. The Switching Fabric is configured to transfer each incoming input to an LE containing a matching entry as indicated by the corresponding RV, establishing one-to-one mapping between the LEs and the Classifiers. We can see from the example in Fig. 3(b) that, the CDLE has less LE entries than the original scheme depicted in Fig. 3(a), while providing equivalent performance, since via enumeration it can be proven that, for any possible combination of inputs, there always exists an appointing scheme in the Switching Fabric satisfying: 1) all inputs are served in one cycle and 2) no two inputs are sent to the same LE. Such a scheme is named an "*Acceptable Matching*" in this paper. For a CDLE to emulate the original LEs in both functions and performance, the following two conditions must be met:

1) The *Existence Condition*: there always exists at least one *Acceptable Matching*;
2) The *Feasibility Condition*: in each clock cycle, the Switching Fabric can find at least one *Acceptable Matching* if it exists.

In the remainder of this section, the entry assigning (EA) algorithm assuring the *Existence Condition*, the scheduling algorithm satisfying the *Feasibility Condition* and the Bloom-filter-based scheme for implementing Classifiers in a CDLE will be introduced in turn.

2) The EA Algorithm: The objective of the EA algorithm is decomposed to the following.

Objective 1: Satisfying the *Existence Condition*.

Objective 2: Keeping the overall storage cost for the CDLE as small as possible.

Objective 3: Keeping the storage cost for every LE as balanced as possible. This objective is added to facilitate the hardware design for CDLE.

It will be shown in Section III–C3 that even though these objectives are achieved, the expected results are not sufficient to make the scheduling algorithm efficient enough, unless the following constraint is also applied, by which the temporal cost of the scheduling algorithm is reduced from $O(N^{2.5})$ to $O(N)$.

The *Fastidious Constraint*: each entry should be assigned only in the following two ways: 1) to a SINGLE LE and 2) to ALL LEs in the CDLE.

With the three objectives under the *Fastidious Constraint*, the problem confronted by the EA algorithm can be proven NP-Hard,[2] even when no account for *Objective 3* is taken. Therefore, we must turn to heuristic algorithms. The main idea of the proposed approximation algorithm is described as follows.

1) From all unassigned entries separate the following two categories in turn: a) *self-correlated* entries, of which every one is correlated with itself and b) *independent* entries, of which every one is not correlated with any other entries in the same CDLE, except the self-correlated entries.
2) Assign as many as possible of the remaining entries to the LEs without replication. In this process, any two correlated entries must not be assigned to the same LE.
3) Replicate each of the remaining entries to all LEs after step 2).
4) Replicate each of the self-correlated entries to all LEs.
5) Balance the sizes of the LEs with two measures:
   - first, limiting the maximal number of entries assigned to each LE in step 2);
   - second, using the independent entries to smooth the discrepancy among the entry numbers of all LEs.

The overall storage cost of the algorithm can be bounded by maximizing the entries assigned in step 2) and minimizing those

[2]Because of the page limit, we put the NP-Hard proof on [29].

(1) Separate all self-correlated entries and independent entries to *SC* and *I* respectively, and put the remaining ones to *R*.
(2) Call the algorithm in Fig. 5 to divide *R* into as few groups as possible.
(3) Assign entries in the largest w groups to w LEs respectively.
(4) Replicate every self-correlated entry and every entry in any remaining groups to all LEs.
(5) Use entries in *I* to balance the size of the LEs.

Fig. 4. The EA algorithm.

```
i=1; //Group index
while R ≠ φ do
  G_i = φ; //Create an empty group
  X = R;
  while X ≠ φ do
    //Add entries to group G_i
    remove an entry Ent_1 from X;
    if ∀Ent ∈ G_i, correlated(Ent,
                  Ent_1) = True then
      G_i = G_i ⋃ {Ent_1};
      R = R − {Ent_1};
    if ||G_i|| ≥ ||R|| / w then break;
      // Limit the max. group size
  i = i+1; //To next group
return {G_j | 1 ≤ j < i}
```

Fig. 5. A greedy grouping algorithm.

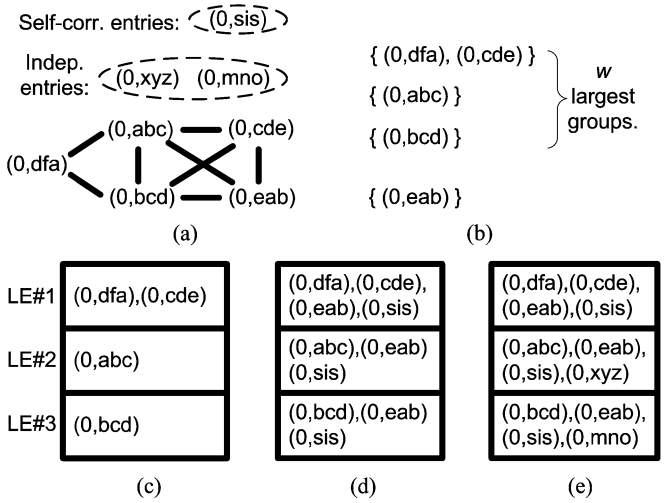


Fig. 6. Illustration of the EA algorithm for a CDLE with $w = 3$.

assigned in step 3). The algorithm is shown in Fig. 4 and an example is given in Fig. 6.

The EA algorithm can be justified according to the following two lemmas proven in Appendix II:

Lemma 1: When all LEs are empty, the *Existence Condition* is satisfied.

Lemma 2: Suppose that the *Existence Condition* is satisfied for any assigning scheme, where n entries in a given set have been assigned; then for a scheme with $n + 1$ assigned entries, the *Existence Condition* is still satisfied *if only* 1) the new entry

Pass 1:
for each ***selective input*** with RV[*i*]=1 **do**
appoint LE#*i* to it;
Pass 2:
for each input not appointed yet **do**
assign to it an LE not appointed;
mark this LE as appointed.

Fig. 7. The scheduling algorithm.

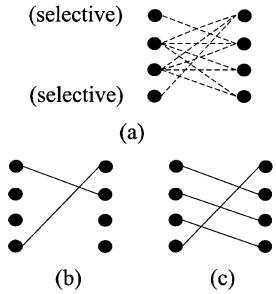


Fig. 8. An example of scheduling.

assigned to an LE is not correlated with itself or any existing entries inside *OR* 2) the new entry is replicated to all LEs.

3) The Scheduling Algorithm: The aim of the scheduling algorithm can be derived from a maximum-bipartite-matching problem [30] where each *Acceptable Matching* in the Switching Fabric corresponds to a maximum matching in a bipartite. One of the most up-to-date noniterative algorithms requires $O(N^{2.5})$ to solve this problem and is too complex for hardware implementation [31]. Many efficient iterative matching algorithms such as iSLIP [31] and DRRM [32] are proposed, and can achieve 100% throughput by making certain assumptions on the characteristics of the incoming traffic. Instead of resorting to these algorithms, the *Fastidious Constraint* is imposed on the EA algorithm to trade off its simplicity for the straightforwardness of the scheduling algorithm.

With *Fastidious Constraint* guaranteed, the entry or entries matching an input, if any, can only exist in one or all LEs. Therefore, the result RV for any input can only have 0, 1, or w bits set. We denote any input with only one bit set in the result RV by *selective input*. Then the *Acceptable Matching* can be calculated by first appointing LEs to the selective inputs, and then to the other inputs, as shown in Fig. 7 and illustrated in Fig. 8. Note that we assume that in Pass 2 the LE with a less significant order number is assigned with a higher priority. For example, in Fig. 8(c), the third LE is appointed before the fourth LE.

Since the EA algorithm has assured the *Existence Condition*, in Pass 1 of the scheduling algorithm, no two selective inputs can be appointed to the same LE. In other words, each LE is appointed with at most one input. This feature remains in Pass 2, as every LE is marked immediately after being appointed. Therefore, the final result is an *Acceptable Matching*. In addition, no input or LE is processed recursively in Pass 1 and Pass 2, which bounds the temporal cost of the algorithm to O(N).

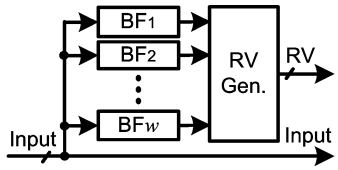


Fig. 9. The structure of a classifier.

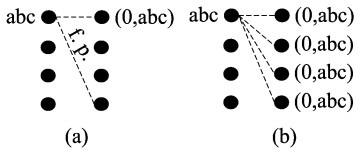


Fig. 10. Error caused by false positive (f. p.) and rescue.

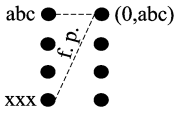


Fig. 11. LE contention.

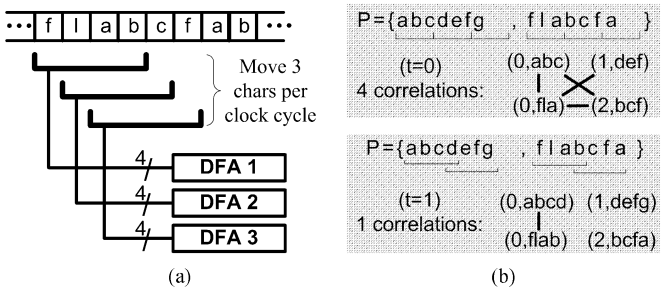


Fig. 12. Look-ahead characters.

4) The Implementation of Classifiers: We use Bloom filter (BF) [33], a widely used membership-querying solution to implement the Classifiers. A BF is a randomized data structure that can represent a set of elements, e.g., strings in this paper. A BF can be efficiently implemented based on embedded SRAM with the aid of state-of-art ASIC technology [34]. Let "program" refer to the operation to update a BF with a set of strings. Given a string X, a BF programmed with string set D definitely reports a hit if $X \in D$. But if $X \notin D$, it may still report a hit with probability p_f. This situation is called false positive, and p_f indicates the false positive rate.

Let m denote the number of RAM bits required, n denote the number of strings in the given string set D, and k denote the number of hash functions used in a BF. The false positive rate of the BF is approximately $(1-e^{-nk/m})^k$, when $m >> n$. And if p_f is minimized with respect to k, we get:

$$p_{f\_\min} = 2^{-k} = 2^{-\frac{m}{n}\ln 2} \quad [34]. \tag{3.1}$$

For simplicity, in our prototype, it is assumed that all BFs are configured with the same p_f small enough ($p_f \leq 0.001$).

The structure of a Classifier is shown in Fig. 9. For a given input I, the function of the *RV Generator* is twofold.

1) If all BFs or only one BF reports a hit, then for $1 \leq i \leq w$, the ith bit in the RV output is set iff the ith BF reports a hit.
2) According to the *Fastidious Constraint*, the entry or entries matching a given input can exist in either all LEs or merely one LE. Therefore, if more than one but not all of the BFs report hits, or every BF report a miss, all bits in the RV should be reset, causing the switching fabric to drop the corresponding input.

Since false positives are inherent in BFs, valid inputs may be dropped, as shown in Fig. 10(a). To avoid such situations, at the end of the algorithm for every CDLE, we should test each entry inside to check whether it would cause more than one but not all BFs to report hits; if so, the entry should be replicated to all LEs, as shown in Fig. 10(b), and the corresponding BFs should be modified.

On the other hand, false positives in the BFs may also break the *Existence Condition*, as shown in Fig. 11. Therefore, we must revise the scheduling algorithm to appoint at each clock cycle only one of the multiple inputs contending for an output, and run iteratively until every one of them is processed. In the worst case, w cycles are needed to resolve the contention. The performance degradation yielded will be analyzed in Section III-E.

5) Further Improvement on CDLE: In the aforementioned description of CDLE, two entries are correlated if only a suffix of one entry is the same as one of the prefixes of the other, e.g., (0,bcfa) and (0,abcd) when $w = 4$. This produces copious amounts of correlated entries, resulting in the low storage efficiency of the EA algorithm. For improvement, we introduce t additional input characters, called "look-ahead" characters, for each input window, as shown in Fig. 12(a), where $t = 1$. Let s denote the number of DFAs. We note that though w increases with t, s remains the same and satisfies $s = w - t$. Accordingly, we will use s and t instead of w to specify the system configuration in the rest of the sections.

After adding look-ahead characters, two entries are correlated only when an entry's suffix at least $t+1$ characters long is the same as one of the prefixes of the other. For example, when $s = 3$ and $t = 1$, though w still equals 4, (0, bcfa) and (0, abcd) are no longer correlated since "bcfa" and "abcd" cannot appear together in the input windows of Fig. 12(a). Another example is presented in Fig. 12(b). For the given signature set P, there are four correlations when $t = 0$, while the number of correlations decreases to 1 when $t = 1$. Understandably, the number of correlations will diminish as the value of t increases, however, also resulting in the redundant storage for part of characters. Therefore, the value of t should be carefully tuned, which is illustrated in experiments in Section IV.

D. Overall Architecture of TDP-DFA

In Section III-C, resource sharing is achieved among different LEs, resulting in the CDLE implementation. Similarly, resource sharing can also be introduced among SLE(j)s ($1 \leq j < w$) resulting in the implementation of a CDSLE(j), which differs from the CDLE in two aspects: 1) no entry inside includes any state ID and 2) the length of each input string is j. The two differences pose the limitation that a CDSLE(j) can only have at most $j - t$ parallel input windows, so as to guarantee that any two of them share a common substring at least t characters

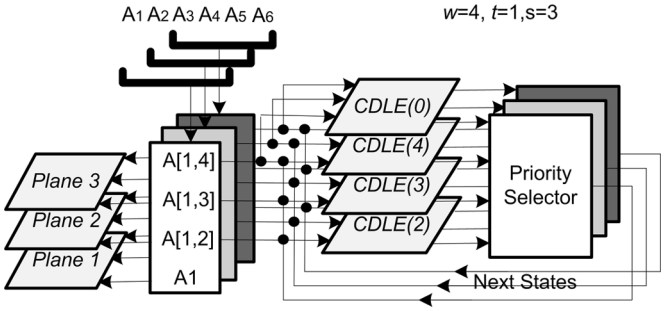


Fig. 13. An example of TDP-DFA.

long. Consequently, to emulate s SLE(j)s, at least $\lceil s/(j-t)\rceil$[3] CDSLE(j)s should be deployed in parallel, and such a substitution only makes sense when $j - t \geq 2$. Taking CDLEs and CDSLEs as building blocks, the architecture of TDP-DFA is illustrated in Fig. 13.

Comparing Fig. 13 with Fig. 2(a), we can see that TDP-DFA differs from multiple parallel DFAs in that: 1) all LE(j)s are substituted with a single CDLE(j) and 2) all SLE(j)s are replaced with a single Plane(j), which consists of $\lceil s/(j-t)\rceil$ CDSLEs if $j - t \geq 2$, or otherwise s original SLE(j)s. Since CDLEs and CDSLEs can emulate parallel LEs and SLEs, respectively, TDP-DFA resembles the naïve model of parallel DFAs in terms of both function and performance, while reducing the storage requirement by effective resource sharing.

E. Theoretical Analysis

Assume that a pipeline for TDP-DFA is constructed appropriately by interleaving among several input flows, and that BCAMs constitute the critical path in the circuit implementation, which means the highest clock frequency merely depends on the BCAM access delay. Hence, in the ideal case s LEs working in parallel can achieve a throughput up to s times faster than that of a single BCAM scheme. However, according to Section III–C4, due to the probability of false positive reports given by the BFs, we may not actually achieve a full speedup of s.

We start by formalizing the worst-case condition (WCC) for the string matching in the TDP-DFA as: for every clock cycle, each BF within all CDLEs reports a hit, i.e., no matching traffic can be excluded.

Under WCC, with respect to the Switching Fabric of a specific CDLE, in every clock cycle, each input port has a request. If no conflict occurs, every input is appointed to a distinct output port, and therefore all the matching requests can be performed in parallel within one clock cycle; otherwise multiple clock cycles may be required, e.g., if q inputs conflict with each other (i.e., they are appointed to the same output port), q clock cycles should be required to finish the corresponding string matching. So, it is straightforward that the worst-case processing latency of a CDLE actually depends on the maximum mutually conflicting input number of the corresponding Switching Fabric.

In the following, we will present the deduction of the performance of TDP-DFA under WCC. Note that in Fig. 13, each Priority Selector must wait for the results of all the $s+1$ CDLEs in the worst case, creating a bottleneck within the entire system. Let $C(i)_{\max}(1 \leq i \leq s+1)$ denote the maximum number of mutually conflicting input ports in the Switching Fabric of the ith CDLE, and $n(i)_{\text{fp}}$ denote the number of Classifiers incurring false positives in the ith CDLE, and the mathematical expectation of the processing latency (in terms of clock cycles) of TDP-DFA T_D, is given by

$$\begin{aligned}\cdot E(T_D) &= E[\operatorname*{Max}_i C(i)_{\max}] \\ &= \sum\nolimits_{n=0}^{s} n \times P(\operatorname*{Max}_i C(i)_{\max} = n).\end{aligned}$$

In order to obtain the probability $P(\operatorname*{Max}_i C(i)_{\max} = n)$, we first work out $P(C(i)_{\max} = n)$.

Since $P(n(i)_{\text{fp}} = n) = C_S^n p_f^n (1-p_f)^{S-n}$ and p_f in our prototype is very small ($p_f \leq 0.001$), we have

$$\frac{P(n(i)_{\text{fp}} = n)}{P(n(i)_{\text{fp}} = n-1)} = \frac{s-n+1}{n} \cdot \frac{p_f}{1-p_f} \ll 1.$$

So, $P(n(i)_{\text{fp}} = n) \leq P(n(i)_{\text{fp}} = 3) \ll 1$ when $n \geq 3$. And accordingly, if $n \geq 4$

$$\begin{aligned}P(C(i)_{\max} = n) &\leq P(n(i)_{\text{fp}} \\ &\geq n-1) \leq P(n(i)_{\text{fp}} \geq 3) \\ &\ll 1.\end{aligned}$$

Therefore, we ignore the influence of $P(C(i)_{\max} = n)$ when $n \geq 4$ for the simplicity of deduction.

Suppose that for each input port with false positive happening, the probability to request each of the output ports is equal, and the decision result of any BF is independent of each other, then for any CDLE, we have

$$\begin{aligned}P_1 :&= P(C(i)_{\max} = 1) = \sum\nolimits_{j=0}^{s} P(n(i)_{\text{fp}} = j) \\ &\quad \times P(C(i)_{\max} = 1 | n(i)_{\text{fp}} = j) \\ &= P(n(i)_{\text{fp}} = 0) \times 1 + P(n(i)_{\text{fp}} = 1) \\ &\quad \times \frac{1}{s} + P(n(i)_{\text{fp}} = 2) \times \frac{2!}{s^2}^{*} + O(p_f^3) \\ &= (1-p_f)^s + \frac{C_s^1 p_f (1-p_f)^{s-1}}{s} \\ &\quad + \frac{2 \times C_s^2 p_f^2 (1-p_f)^{s-2}}{s^2} + O(p_f^3) \\ &\approx (1-p_f)^s + p_f; \\ P_2 :&= P(C(i)_{\max} = 2) = P(n(i)_{\text{fp}} = 1) \\ &\quad \times \frac{C_{s-1}^1}{s} + P(n(i)_{\text{fp}} = 2) \\ &\quad \times \frac{2! + C_2^1 \times C_{s-2}^1 \times C_2^1 + C_{s-2} \times C_{s-3}}{s^2}^{**} \\ &\quad + O(p_f^3) \\ &= \frac{C_{s-1}^1 C_s^1 p_f (1-p_f)^{s-1}}{s} \\ &\quad + \frac{C_s^2 p_f^2 (1-p_f)^{s-2} \times (s^2 - s)}{s^2} + O(p_f^3) \\ &\approx (s-1)p_f + \frac{s^2 - s}{2} p_f^2; \\ P_3 :&= P(C(i)_{\max} = 3) = P(n(i)_{\text{fp}} = 2)\end{aligned}$$

[3] $\lceil x \rceil$ means the minimum integer larger than x.

$$\times \frac{C_{s-2}^{1}}{s^2}^{***} + O(p_f^3)$$
$$= \frac{(s-2)\times C_s^2 p_f^2 (1-p_f)^{s-2}}{s^2} + O(p_f^3)$$
$$\approx \frac{s-2}{2} p_f^2;$$
$$\forall n > 3, P(C(i)_{\max} = n) = O(p_f^3) \to 0.$$

Note the following.

- When two false positives occur, $C(i)_{\max} = 1$ iff the corresponding two input ports request for the two output ports which are not requested by the other $s-2$ input ports without a false positive occurring.
- When two false positives occur, $C(i)_{\max} = 2$ iff: 1) the two corresponding input ports request for the same output port which is not requested by any of the other $s-2$ input ports without false positive occurring, OR 2) the two corresponding input ports request for different output ports, only one of which is also requested by one of the other $s-2$ input ports without false positive occurring, OR (3) the two corresponding input ports request for different output ports, both of which are also requested by one of the other $s-2$ input ports without a false positive occurring.
- When two false positives occur, $C(i)_{\max} = 3$ iff the two corresponding input ports request for the same output port which is also requested by one of the other $s-2$ input ports without a false positive occurring.

Now, we go on to calculate $P(\underset{i}{\text{Max}}\, C(i)_{\max} = n)$

$$P(\underset{i}{\text{Max}}\, C(i)_{\max} = 1) = P(\bigcap_i C(i)_{\text{Max}} = 1) = P_1^{s+1}.$$

Note: $\underset{i}{\text{Max}}\, C(i)_{\max} = 1$ iff for all the $s+1$ CDLEs, the corresponding $C(i)_{\max} = 1$

$$P(\underset{i}{\text{Max}}\, C(i)_{\max} = 2) = \sum_{i=1,\ldots,s+1} C_{s+1}^i P_2^i P_1^{s+1-i};$$
$$P(\underset{i}{\text{Max}}\, C(i)_{\max} = 3) = \sum_{j=1,\ldots,s+1} C_{s+1}^j P_3^j \times \left(\sum_{i=0,\ldots,s+1-j} C_{s+1-j}^i P_2^i P_1^{s+1-j-i} \right);$$
$$P(\underset{i}{\text{Max}}\, C(i)_{\max} > 3) = O\left(p_f^3\right).$$

So, we have

$$E(T_D) = E[\underset{i}{\text{Max}}\, C(i)_{\max}]$$
$$= \sum_{n=1}^{s} n \times P(\underset{i}{\text{Max}}\, C(i)_{\max} = n)$$
$$= P_1^{s+1} + 2 \times \sum_{i=1,\ldots,s+1} C_{s+1}^i P_2^i P_1^{s+1-i} + 3 \times \sum_{j=1,\ldots,s+1} C_{s+1}^j P_3^j \times \left(\sum_{i=0,\ldots,s+1-j} C_{s+1-j}^i P_2^i P_1^{s+1-j-i} \right) + O\left(p_f^3\right).$$

■

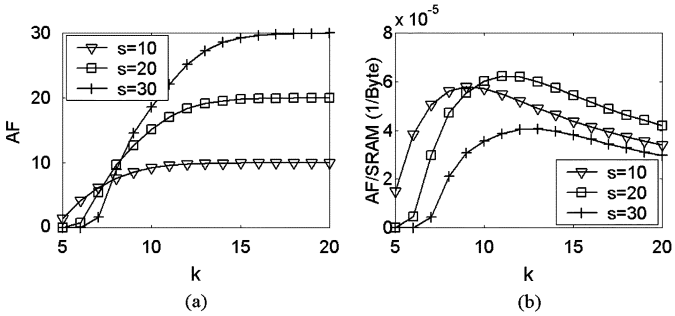


Fig. 14. Tradeoff for k.

We also define acceleration factor (AF) as $s/E(T_D)$, indicating the speedup between the naive BCAM scheme and TDP-DFA.

Note that, the performance deduction above virtually relies on the assumption that the hash functions used in the BFs are independent of the incoming traffic; thus no attacker is capable of generating malicious traffic to drive the BFs to encounter continual false positives. This can be realized by keeping secret the implementation details of the BFs, or building the BFs with reconfigurable hash functions which can be altered either manually or automatically.

IV. Experimental Results

Using the pattern set from Snort (version 2.3.3, released in April 2005), we extract 2234 distinct substrings containing 33 793 characters from the signature database.

In our prototype, the space for each state field in a CDLE entry is 2 bytes, allowing the maximum number of states up to 65 536. This is large enough considering the maximum number we measured in real cases is less than 6000. Similarly, the "Action ID" field in an entry of the associated RAM also occupies 2 bytes.

In what follows, we will present the experimental results measured under different configurations of TDP-DFA, which mainly depend on three key parameters: 1) k : the number of hash functions each Bloom Filter contains; 2) t : the number of look-ahead characters; and 3) s : the number of DFAs. Afterwards we will compare the results with other recent works.

Initially, we tune k. From (3.1), we see that although the increase of k diminishes p_f exponentially (and thus raises AF towards s, as shown in Fig. 14(a), it also enlarges the storage cost of the Bloom filters proportionally since $k = m \cdot \ln 2/n$. Fig. 14(b) shows when k ranges from 9 to 12, the optimal gain/cost ratio is attained. In our prototype, we choose $k = 12$ for simplicity, and the corresponding p_f equals $2^{-12} \approx 0.0002$.

Then, we tune parameter t. Fig. 15(a) shows the ratio of independent entries goes up rapidly when increasing t, as explained

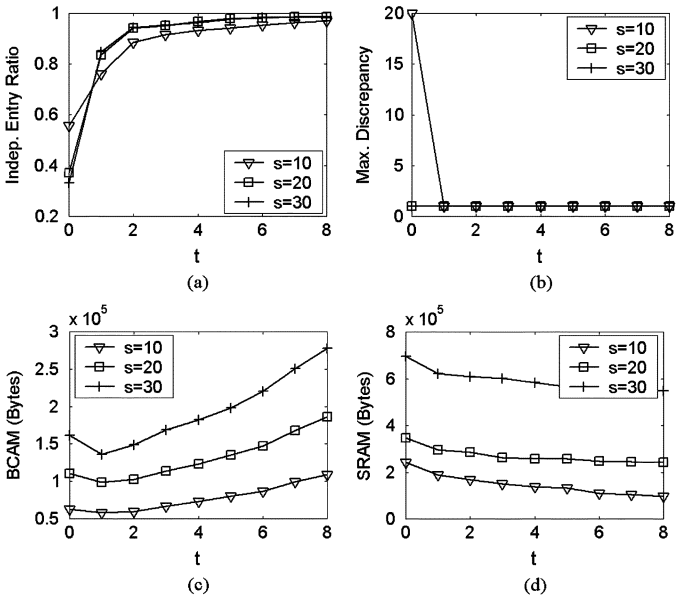


Fig. 15. Tradeoff for t.

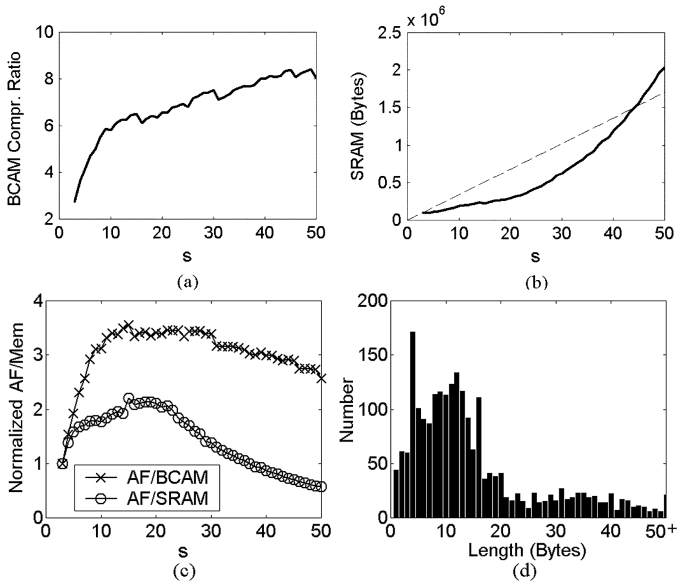


Fig. 16. Tradeoff for s.

in Section III–C4. The abundance of independent entries effectively contributes to the balancing effect of the EA algorithm. As shown in Fig. 15(b), when $t \geq 1$, the maximum discrepancy among the numbers of entries allocated to each LE in a CDLE remains minimized as 1. However, an increase in t has side effects. Fig. 15(c) and (d) show that increasing t is unfavorable since it raises BCAM consumption more quickly than it reduces SRAM consumption. This is because increasing t also incurs redundant storage of some characters in multiple transition rules. Taking a comprehensive view on Fig. 15, it is obvious that $t = 1$ is the most optimal value, since it not only maintains good balancing effect, but also avoids demanding much more BCAM consumption than $t = 0$.

The last key parameter is s. Let *BCAM compression ratio* denote the ratio of BCAM requirement before to after resource sharing. Fig. 16(a) shows that the ratio increases with the increment of s, and grows over 6 when s exceeds 10, demonstrating the good scalability of TDP-DFA.

However, raising s is not always preferable to scale up the system. Fig. 16(b) shows that when s grows over 45, the SRAM requirement no longer maintains the sublinear relationship with s. Fig. 16(c) further explains the tradeoff for s with two normalized gain/cost ratios. We can see that the ratios approach the maximum when s increases to about 20, but falls for higher values of s. The reason for this phenomenon is interpreted as follows.

In TDP-DFA, the short signatures are stored less efficiently than the long ones since multiple identical CDSLEs for detecting short signatures need to be deployed in parallel, as explained in Section III-C. Based on the length distribution of signatures in Snort shown in Fig. 16(d), we find that the length of most signatures ranges from 1 to 20. This explains why the two ratios in Fig. 16(c) fall when $s \geq 20$.

With two parameters appropriately tuned as $k = 12$ and $t = 1$, in Table II three different configurations of TDP-DFA are compared with a few recent hardware-based string matching mechanisms. Besides listing the performance and storage cost, we also compute the memory efficiency of each solution, which is defined by (*Rule set Size* $\times$ *Throughput*)/*Memory Requirement*, as shown in the last two columns. It is shown that, with far less storage cost and a practical memory frequency, TDP-DFA can manage string matching throughput and memory efficiency remarkably higher than the other solutions. Furthermore, it not only satisfies the critical requirement of OC-768 wirespeed, but also demonstrates a promising way for scaling up to cope with throughput over 100 Gb/s in the future.

V. Conclusion

NIDS has been widely accepted as an effective way to defend Internet systems from attacks. Due to the ever-growing requirement on the capacity of NIDS, string matching algorithm, a key component, has become the bottleneck to further improvement in the performance of NIDS. To address this issue, we propose a memory-efficient multiple-character-approaching architecture suited for ASIC implementations, TDP-DFA. We introduce parallel DFAs with overlapping input windows to achieve the goal of processing multiple characters in each clock cycle. By slight modification to the straightforward representation of the transition rules, the complexity of each DFA is distinctively reduced. We also identify the correlations among the inputs to the parallel DFAs, and propose a novel scheme using a Switching Fabric-based structure and Bloom filter-based Classifiers for sharing most of the transition rules among them, thus significantly reduce the overall storage cost. Performance evaluation shows that our design can deal with throughput well exceeding OC-768 wirespeed even in the worst case. TDP-DFA meets the size limitation of embedded memory, and is able to be implemented on-chip with current ASIC technology.

Appendix I
Computation for Function *Correlated*

We start by defining a Boolean function $SameType(Ent_1, Ent_2) = True$ iff Ent_1 and Ent_2 are of the same type (i.e.,

TABLE II
COMPARISONS WITH OTHER WORKS

	Rule-set Char. Num.	Storage Cost(Byte)		Memory Freq. (Hz)		Worst-case Throughput	Memory Efficiency (char.*Gbps/Byte)	
		(B/T)CAM	Emb. SRAM	(B/T) CAM	Emb. SRAM		(B/T) CAM	Emb. SRAM
Yu[18]†	22k†	(T) 295K	Negligible	250M	250M~500M	2Gbps†	(T) 0.15†	-
Aldwari[21]	~20k‡	-	3M	-	625M	5Gbps	-	0.03
Tuck[20]	19k	-	1.1M	-	Not published.	7.8Gbps	-	0.13
Tan [22]	~22k‡	-	0.4M	-	1.25G	10Gbps	-	0.55
Cho[23]	22k	-	0.1M	-	893M	7.1Gbps	-	1.56
TDP-DFA (s=10)	33k	(B) 58K	188K.	600M*	600M*	46Gbps	(B) 26.17	8.07
TDP-DFA (s=20)	33k	(B) 99K	296K.	600M*	600M*	88Gbps	(B) 29.33	9.81
TDP-DFA (s=30)	33k	(B) 136K	621K.	600M*	600M*	128Gbps	(B) 31.06	6.80

Notes: * : Embedded (B/T)CAM blocks are now available with maximum frequency over 800MHz [35], so we assume that our BCAM blocks running at 600MHz are practical. †: Yu's work can also handle the regular expressions in Snort rule set, which are still not taken into account in the "Rule-set character number" field. Besides, we believe by replacing the external TCAM with embedded TCAM blocks running at 800MHz, its throughput can rise to 6.4Gbps and result in higher TCAM memory efficiency as 0.48. ‡: The numbers are deduced from the paper but not directly cited.

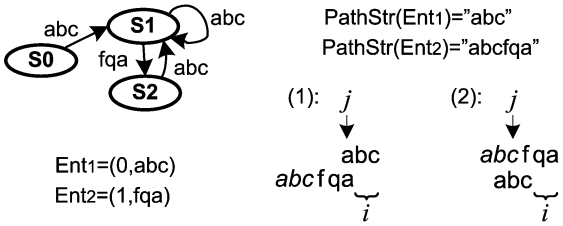


Fig. 17. Illustration of computing function correlated. ($s = w = 3$).

with or without the "current state" field) and with the same length in each field. Then, the function *correlated* can be computed as follows.

Let $PathStr(Ent)$ denote the concatenation of the shortest string able to drive DFA from root state to $Ent's$ "current state" and $Ent's$ "input" field. For example, $PathStr$((1,pla))="displa" in Fig. 1.

Let $l_1 = PathStr(Ent_1)$ and $l_2 = PathStr(Ent_2)$. Let s denote the number of DFAs, as defined in Section III–C5. As illustrated in Fig. 17, two entries Ent_1 and Ent_2 satisfy $correlated(Ent_1, Ent_2) = True$ iff $SameType(Ent_1, Ent_2) = True$ and either of the following two conditions is met:

1) $\exists i, 1 \le i \le s-1, \forall j,$ if $1 \le j \le LEN(l_1)$, then $i+j > LEN(l_1)$or $l_1(j) = l_2(LEN(l_2)+i-LEN(l_1)+j)$;
2) $\exists i, 1 \le i \le s-1, \forall j,$ if $1 \le j \le LEN(l_2)$, then $i+j > LEN(l_2)$or $l_2(j) = l_1(LEN(l_1)+i-LEN(l_2)+j)$.

APPENDIX II
PROOF OF LEMMAS

Initially, let *configuration map* (CM) describe the state of the Switching Fabric in a certain clock cycle, i.e., to specify the destination LE for each input. A CM is a map defined as follows:

$$\mathrm{CM}(i) = \begin{cases} j, & \text{if the } i\text{th input should be sent to LE\#}j \\ 0, & \text{if the } i\text{th input should be dropped.} \end{cases}$$

According to the definition of *Acceptable Matching*, given a set of parallel inputs, if CM is an *Acceptable Matching*, it must satisfy that $\forall i, \forall j,$ if $i \neq j$,

$$\text{then } \mathrm{CM}(i) \neq \mathrm{CM}(j) \text{ or } \mathrm{CM}(i) = \mathrm{CM}(j) = 0. \qquad \text{(II.1)}$$

Proof of Lemma 1: When all LEs are empty, no input should be sent to any LE for matching, and thus all inputs should be dropped by the Switching Fabric immediately. Hence, the Existence Condition is satisfied.

Proof of Lemma 2: It should be proven that, if exactly $n+1$ entries have been assigned under the assumption in Lemma 2, at least one *Acceptable Matching* can be found for any given inputs $\{I_1, I_2, \ldots, I_w\}$. According to the assumption in Lemma 2, at the moment between the nth and the $(n+1)$th entry assignments, the *Existence Condition* is satisfied for the given inputs $\{I_1, I_2, \ldots, I_w\}$, where we denote one of the existing *Acceptable Matching(s)* by a *Configuration Map* AM_n.

In what follows, it is proven that if the $(n+1)$st entry (denoted by E_{new}) is assigned obeying to either *Principle A* or *Principle B*, then an *Acceptable Matching* can always be constructed by slight revision of AM_n.

First, suppose E_{new} is assigned to the rth LE according to *Principle A*. Since E_{new} is not correlated with itself, it can be matched by zero or merely one input in $\{I_1, I_2, \ldots, I_w\}$. If there is none, it is obvious that AM_n is still an *Acceptable Matching* for $\{I_1, I_2, \ldots, I_w\}$, and so $\mathrm{AM}_{n+1} = \mathrm{AM}_n$. Otherwise, supposing the kth input matches E_{new}, we construct a CM as follows:

$$\mathrm{CM}(i) = \begin{cases} \mathrm{AM}_n(i), & \text{if } i \neq k \\ r, & \text{if } i = k \end{cases}.$$

Since no entry in LE #r is correlated with E_{new}, no input other than I_k should be sent to LE #r. In addition, because AM_n is an *Acceptable Matching*, based on (II.1) and (II.2), we get $\forall i, \forall j,$ if $i \neq j$, then $\mathrm{CM}(i) \neq \mathrm{CM}(j)$ or $\mathrm{CM}(i) = \mathrm{CM}(j) = 0$.

This indicates that CM is an *Acceptable Matching* for the given input set $\{I_1, I_2, \ldots, I_w\}$ and so $\mathrm{AM}_{n+1} = \mathrm{CM}$.

Second, suppose E_{new} is assigned to all LEs according to *Principle B*. If there is no input equal to E_{new}, it is obvious that AM_n is still an *Acceptable Matching* for $\{I_1, I_2, \ldots, I_{\mathrm{w}}\}$. Otherwise, we denote the inputs equal to E_{new} by $J_1, J_2, \ldots, J_{\mathrm{x}}$, and the remaining inputs by $J_{\mathrm{x}+1}, J_{\mathrm{x}+2}, \ldots, J_{\mathrm{w}}$. Since E_{new} is replicated to all LEs, $J_1, J_2, \ldots, J_{\mathrm{x}}$ can be sent to any LE without causing a miss. Therefore, if $J_{\mathrm{x}+1}, J_{\mathrm{x}+2}, \ldots, J_{\mathrm{w}}$ are appointed according to $\mathrm{AM}_n, J_1, J_2, \ldots, J_{\mathrm{x}}$ can be appointed to the remaining LEs one to one.

Let $L = \{1, 2, \ldots, \mathrm{w}\} - \{\mathrm{AM}_n(J_{\mathrm{x}+1}), \mathrm{AM}_n(J_{\mathrm{x}+2}), \ldots, \mathrm{AM}_n(J_{\mathrm{w}})\}$ and denote L as $\{L_1, L_2, \ldots, L_{\mathrm{w}}\}$. We construct a CM as follows:

$$\mathrm{CM}(i) = \begin{cases} \mathrm{AM}_n(i), & \text{if } I_i \text{ does not match } \mathrm{E_{new}} \\ L_c, & \text{if } \exists c, I_i \text{ and } \mathrm{L}_c\text{are the same input.} \end{cases} \tag{II.3}$$

Based on (II.1) and (II.3), again we get

$$\forall i, \forall j, \text{if} i \neq j, \quad \text{then } \mathrm{CM}(i) \neq \mathrm{CM}(j) \text{ or } \mathrm{CM}(i) = \mathrm{CM}(j) = 0 \ldots$$

This indicates that CM is an *Acceptable Matching* for the given input set $\{I_1, I_2, \ldots, I_{\mathrm{w}}\}$ and so $\mathrm{AM}_{n+1} = \mathrm{CM}$.

For any given input set, with the assumption in Lemma 2, we can follow the above procedure to construct a corresponding *Acceptable Matching* when exactly $n + 1$ entries have been assigned, and hence Lemma 2 is proved.

REFERENCES

[1] *Snort-the de Facto Standard for Intrusion Detection/Prevention*, [Online]. Available: www.snort.org

[2] S. Antonatos, K. G. Anagnostakis, and E. P. Markatos, "Generating realistic workloads for network intrusion detection systems," presented at the Proc. ACM Workshop on Software and Performance, Redwood Shores, CA, 2004.

[3] S. Iyer, A. Awadallah, and N. McKeown, "Analysis of a packet switch with memories running slower than the line rate," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 529–537.

[4] *Embedded Memory*, [Online]. Available: http://www.ti.com/research/docs/cmosmemory.shtml

[5] G. Navarro and M. Raffinot, *Flexible Pattern Matching in Strings-Practical On-Line Search Algorithms for Texts and Biological Sequences*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[6] C. J. Coit, S. Staniford, and J. McAlerney, "Towards faster string matching for intrusion detection or exceeding the speed of snort," in *Proc. DARPA Information Survivability Conf. Exposition (DISCEX II'01)*, 2001, pp. 367–373.

[7] M. Fisk and G. Varghese, "Fast content-based packet handling for intrusion detection," UCSD, UCSD Tech. Rep. CS2001–0670, 2001.

[8] K. G. Anagnostakis, E. P. Markatos, S. Antonatos, and M. Polychronakis, "E^2 XB: A domain-specific string matching algorithm for intrusion detection," presented at the 18th IFIP Int. Information Security Conf., Athens, Greece, 2003.

[9] R. T. Liu, N. F. Huang, C. H. Chen, and C. N. Kao, "A fast string-match algorithm for network processor-based network intrusion detection system," *ACM Trans. Embedded Comput. Syst.*, vol. 3, pp. 614–633, 2004.

[10] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, "Implementation of a content-scanning module for an Internet firewall," in *Proc. 11th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach.*, Napa, CA, Apr. 2003, pp. 31–38.

[11] I. Sourdis and D. Pnevmatikatos, "Fast, large-scale string match for a 10 Gbps FPGA-based network intrusion detection system," presented at the 13th Conf. Field Programmable Logic and Appl., Lisbon, Portugal, 2003.

[12] C. R. Clark and D. E. Schimmel, "Scalable pattern matching for high speed networks," in *Proc. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2004, pp. 249–257.

[13] Z. K. Baker and V. K. Prasanna, "A methodology for synthesis of efficient intrusion detection systems on FPGAs," in *Proc. Field-Programmable Custom Comput. Mach. 12th Annu. IEEE Symp.*, 2004, pp. 135–144.

[14] Y. Sugawara, M. Inaba, and K. Hiraki, "Over 10 Gbps string matching mechanism for multi-stream packet scanning systems," in *Lecture Notes in Computer Science*. Heidelberg, Germany: Springer-Verlag, 2004, vol. 3203, Proc. 14th In. Conf. Field-Programmable Logic Appl., pp. 484–493.

[15] S. Yusuf and W. Luk, "Bitwise optimised CAM for network intrusion detection systems," in *Proc. 15th Conf. Field Programmable Logic Appl.*, 2005, pp. 444–449.

[16] Y. H. Cho and W. H. Mangione-Smith, "Fast reconfiguring deep packet filter for 1+ gigabit network," in *Proc. IEEE Symp. Field Programmable Custom Comput. Mach.*, Napa Valley, CA, 2005, pp. 215–224.

[17] M. E. Attig and J. Lockwood, "A framework for rule processing in reconfigurable network systems," in *Proc. 13th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach.*, Napa, CA, 2005, pp. 225–234.

[18] F. Yu, R. H. Katz, and T. V. Lakshman, "Gigabit rate packet pattern-matching using TCAM," in *Proc. Netw. Protocols, 12th IEEE Int. Conf.*, 2004, pp. 174–183.

[19] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, pp. 333–340, 1975.

[20] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic memory-efficient string matching algorithms for intrusion detection," in *Proc. INFOCOM 23rd Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 2004, pp. 2628–2639.

[21] M. Aldwairi, T. Conte, and P. Franzon, "Configurable string matching hardware for speeding up," *SIGARCH. Comput. Archit. News*, vol. 33, no. 1, pp. 99–107, 2005.

[22] L. Tan and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," in *Proc. Int. Symp. Comput. Arch.*, 2005, pp. 112–122.

[23] Y. H. Cho and W. H. Mangione-Smith, "A pattern matching coprocessor for network security," presented at the 42nd Des. Autom. Conf., Anaheim, CA, 2005.

[24] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood, "Deep packet inspection using parallel bloom filters," *IEEE Micro*, vol. 24, no. 1, pp. 52–61, 2004.

[25] S. Dharmapurikar and J. Lockwood, "Fast and scalable pattern matching for content filtering," presented at the Symp. Arch. Netw. Commun. Syst., Princeton, NJ, 2005.

[26] G. Tripp, "A finite-state machine based string matching system for intrusion detection on high-speed networks," in *Proc. EICAR Conf.*, 2005, pp. 26–40.

[27] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed. Reading, MA: Addison-Wesley, 2000.

[28] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, pp. 712–727, 2006.

[29] H. Lu, K. Zheng, B. Liu, X. Zhang, and Y. Liu, "A memory-efficient parallel string matching architecture for high speed intrusion detection," Tech. Rep., 2005. [Online]. Available: http://s-router.cs.tsinghua.edu.cn/~luhongbin/publications/TR_string_matching.pdf

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (2nd Edition)*. Cambridge, MA: Mass. Inst. Technol., 2001.

[31] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[32] H. J. Chao and J. S. Park, "Centralized contention resolution schemes for a large-capacity optical ATM switch," in *Proc. IEEE ATM Workshop*, Fairfax, VA, May 1998, pp. 11–16.

[33] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, 1970.

[34] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, *Longest Prefix Matching Using Bloom Filters*. Karlsruhe, Germany: ACM Press, 2003, Proc. ACM SIGCOMM, pp. 201–212.

[35] Analog Bits Inc., High Speed Ternary CAM Datasheet, 2004. [Online]. Available: www.analogbits.com/pdf/High_Speed_T_CAM_Datasheet.pdf

Hongbin Lu (S'03) received the B.S. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2002. He is currently working towards the Ph.D. degree in the Department of Computer Science and Technology, Tsinghua University, Beijing.

His research interests include network security, network measurement, and packet classification.

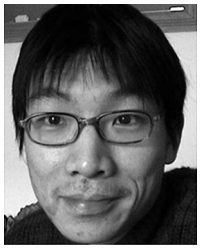
Kai Zheng (S'02) received the B.S. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2001. He is currently working towards the Ph.D. degree in the Department of Computer Science and Technology, Tsinghua University, Beijing.

His research interests include IP address lookup, packet classification, and pattern matching associated network security issues.

Bin Liu (M'03) received the M.S. and Ph.D. degrees in computer science and engineering from Northwestern Polytechnical University, Xi'an, China, in 1988 and 1993, respectively.

From 1993 to 1995, he was a Postdoctoral Research Fellow in the National Key Laboratory of SPC and Switching Technologies, Beijing University of Post and Telecommunications, Beijing, China. In 1995, he transferred to the Department of Computer Science and Technology, Tsinghua University, Beijing, as an Associate Professor, where he mainly focused on multimedia networking including ATM switching technology and Internet infrastructure. He became a Full Professor in the Department of Computer Science and Technology, Tsinghua University, in 1999, and currently is the Director of the Laboratory of Broadband Networking Technologies, Tsinghua University. He led a team to prototype a 64 k $\times$ 64 k single-board intelligent ISDN switch in 1997, which was transferred to ZTE, China, and has been used in the field till today. As a pluralistic CTO of Weifang Beida Jade Bird Huaguang Technology Company, Ltd., from 1996 to 1998, he led a team of 150 people to build a BSP-80 broadband/narrowband hybrid super-large capacity of carrier-class switch and put it into operation in Chinese telecommunication networks. From 1999 to 2001, he was a Principle Investigator of a key project from the national 863 high-tech plan, where he led the team to develop a high-performance scalable core router with capacity of 128 Gb/s, which has been transferred to Bitway Networks Company, Ltd. His current research areas include high-performance switches/routers, high-speed network security, network processors, and traffic management.

Dr. Liu has received numerous awards from China. He is also a corecipient of the 16th ICCC Best Paper Award among over 800 accepted papers. He has served the Panel Chair of HPSR 2005, TPC of INFOCOM 2005/2006, ICCCN 2005, SUTC 2006, IWCMC 2006, and Globlecom 2006. He is a member of Communications and Information Security Technical Committee (CISTC), ComSoc.



Xin Zhang received the B.S. and M.S. degrees from the Department of Automation, Tsinghua University, Beijing. China. He is currently working towards the Ph.D. degree at Carnegie Mellon University, Pittsburgh, PA.

His current researches focus on the IP route lookup, packet classification, as well as system and algorithm design.

Yunhao Liu (M'04–SM'06) received the B.S. degree in automation from Tsinghua University, Beijing, China, in 1995, the M.A. degree from the Beijing Foreign Studies University, Beijing, in 1997, and the M.S. and Ph.D. degrees in computer science and engineering from Michigan State University, East Lansing, in 2003 and 2004, respectively.

He is currently an Assistant Professor in the Department of Computer Science, Hong Kong University of Science and Technology, Kowloon. His research interests include peer-to-peer and grid computing, sensor networks, pervasive computing, network security, and high-speed networking.

Dr. Liu is a member of the IEEE Computer Society.