Abstract—Multi-pattern matching is a key technique for implementing network security applications such as Network Intrusion Detection/Protection Systems (NIDS/NIPSes) where every packet is inspected against predefined attack signatures written in regular expressions (regexes). To this end, Deterministic Finite Automaton (DFA) is widely used for multi-regex matching, but existing DFA-based researches have claimed high throughput at the expenses of extremely high memory cost.

In this paper, we propose a parallel architecture of DFA called Parallel DFA (PDFA), using multiple flow aggregations to increase the throughput with nearly no extra memory cost. The basic idea is to selectively store the DFA in multiple memory modules which can be accessed in parallel and to explore the potential parallelism. The memory cost of our system in both the average cases and the worst cases is analyzed, optimized and evaluated by numerical results. The evaluation shows that we obtain an average speedup of 0.5$k$ to 0.7$k$ where $k$ is the number of parallel memory modules under our synthetic trace and compressed real trace in a statistical average case, compared with the traditional DFA-based matching approaches.

Index Terms—Deterministic Finite Automata (DFA), Deep Packet Inspection (DPI), Regular Expression

I. INTRODUCTION

Nowadays, Network Intrusion Detection/Protection Systems (NIDS/NIPSes) are deployed to safe-guard the network operations. As most of the known attacks can be represented by regular expressions (regexes), multi-regex matching becomes one of the key components in NIDS/NIPSes. Meanwhile, more and more security detection or application identification tend to be performed at devices where the available memory resource is quite limited such as routers, so designing a multi-regex matching scheme with high throughput and quite low memory cost leads to a great challenge both in algorithm design and in hardware implementation. Traditionally, the byte-at-a-time processing manner of Deterministic Finite Automaton (DFA) strictly limits the throughput due to the memory access frequency. To raise throughput, exploiting parallelism, which checks multiple characters per clock cycle, is a promising solution. Nevertheless, existing parallel approaches still suffer from the memory explosion or steep performance under certain rule sets.

Many algorithms [1] [2] [3] [4] are currently developed to exploit the parallelism of matching one flow aggregation by sending multiple characters to the matching engine(s) per cycle. However, to ensure that a matched pattern starts at any position of the flow may lead to a huge extra consumption in memory (severe transition explosion) and hardware logic, and therefore compromises the scalability. Alternatively, processing characters from multiple flow aggregations might be a more practical strategy. Following this way, current approaches [5] [6] [7] [8] [9] [10] [11] mainly focus on compressing single non-accelerated matching engine and simple duplication with little compression ratio. Therefore given the memory size, their speedup is strictly bounded by the compression rate and thus they fail to fully exploit the parallelism.

In this paper we propose a parallel architecture of DFA called parallel DFA (PDFA) which selectively stores one standard DFA in multiple memory devices that can be accessed in parallel with little redundancy. We investigate partitioning DFA in a hierarchical manner: DFA is first divided into state groups and further to transitions subsets. Concretely, two schemes, states assignment (SA) scheme and transitions assignment (TA) scheme, are separately proposed. Sharing the same idea of storing transition in distinct memories, the two schemes can be applied together. It shall be remarked that PDFA makes little change on DFA structure and can be thereby combined with existing compression techniques. Multiple packets can be processed simultaneously once their memory accesses do not conflict. In practice, proper construction of PDFA can greatly eliminate such conflicts and therefore obtains large acceleration. Specifically, our contributions are listed as follows:

1) We propose a parallel representation of DFA which exploits the parallelism among multiple flow aggregations with little memory duplication.

2) We propose a fast and effective construction algorithm for PDFA. Experimental evaluation shows that an average speedup is about 0.5$k$ to 0.7$k$, where $k$ is the number of parallel memory modules under our synthetic trace and compressed real trace in a statistical average case.

3) We devise a preliminary hardware circuit design for PDFA which guarantees the latency fairness among multiple flows while achieving speedup.

The remainder of this paper is organized as follows: Sec-
In Section II presents our approach and gives the problem statement. In Section III, the PDFA’s speedup is mathematically analyzed and in Section IV an effective construction algorithm is further proposed to increase the throughput. Section V provides the hardware architecture design containing the scheduling policy and shows the experimental results. Finally Section VII concludes the paper.

II. EXPLOITING DFA PARALLELISM

We begin with motivation examples of SA and TA schemes, and further give the problem statement of system design.

A. State Assignment (SA) Scheme

Figure 1(a) depicts a standard DFA recognizing pattern set \{aA+b+, [aA]+c+, b+[aA]+, b+[Cc], d+d+\} together with its transitions. Transition assignment (TA) scheme stores transitions of a given DFA in distinct memory modules. In essence, SA scheme is a special case of TA scheme since any state itself is a set of transitions in storage. TA scheme is based on a classical technique called alphabet set classification. To present the intuition, we consider that typical DFA (e.g., Figure 1(a)) has 256 transitions for each state in ASCII alphabet set. But in practice, DFA generator like JFlex [12] never holds 256 entries for each state; instead, they perform classification for transitions: If two characters starting from any state always lead to the same next state, their transitions are grouped into one class. For instance, ‘A’ and ‘a’ in the plots have the same next hop for any states, thus they are classified into one class. Readers may refer to [13] for detailed treatment on the subject.

Delighted by the transition classification, we first classify the transition set of each state into classes and further group the classes into class-sets. Figure 2 illustrates how the TA scheme works in the same standard DFA of Figure 1(a). For instance, state 1 has six transitions being preserved in 3 class-sets. Formally, for any class-set \(i\), the TA scheme stores all its transitions into the same memory module. Since the contents of memory modules do not overlap, no extra cost in memory is brought in by the TA scheme. When receiving one character the matching system dispatches it to the memory module according to the class-set it belongs to. As a result, multiple flows can be matched simultaneously once the incoming characters do not conflict with each others (i.e., belong to distinct class-sets).

---

1 Here “\(x+x\)” refers to one or more than one regex \(x\).
2 We use flow and aggregated flows interchangeable in Section II and Section III for simplicity.
C. Problem Statement

We refer to a DFA stored in TA scheme or SA scheme as a Parallel DFA (PDFA). In our system, PDFA requires that all transitions belonging to the same state group and the same class-set should get stored in one memory module. An active module is any memory module that receives at least one character. We need to maximize the speedup while keeping the fairness amid multiple arrival flows. Albeit scheduling among flow aggregations might influence speedup, the received bytes of each memory module form a new flow, indicating that the acceleration is still independent with flow scheduling statistically. Hence, our first objective is to properly group transitions so that a considerable active modules exist in each cycle. In addition, handling the access conflicts and preventing of flow starvation make it indispensable to design a proper scheduling.

III. ANALYZING SPEEDUP

It shall be seen that the impact of SA and TA scheme is mutually exclusive, indicating that they can be addressed independently. In Subsection III-A we first study the features of input characters in flows, which are the basis of further discussion on TA and SA scheme in III-B.

A. Flow Study and Brief Markovian Analysis of DFA

For modeling the sequences of characters in various real-world flows, we chose to consider “language-independent” statistical modeling of data flows best exemplified by the well-known 1-gram analysis, which suffices to model the character distribution in most flow environments. For detailed treatment of this subject, the reader can refer to [14].

Given a DFA \((Q, \Sigma, \delta, q_0, F)\), where \(Q = q_0, \ldots, q_N, \Sigma, \delta, q_0\) and \(F\) refers to the state set, the input character set, the transition function, the initial state, and the accept state set respectively. Of a certain input flow \(T = t_0, t_1, \ldots\), we denote the current state sequence with \(X = x_0, x_1, \ldots\), where \(x_0 = q_0\) and \(x_{n+1} = \delta(x_n, t_n), n = 0, 1, \ldots\). So the possibility of current state transferring from \(q_i\) to \(q_j\) in the \(n\)th step is \(P_{ij}(n) = \Pr(x_{n+1} = q_j|x_n = q_i) = \sum_{c = \delta(q_i, c)} \Pr(t_n = c)\). Based on the flow modeling, we assume the stationary distribution of \(D(c) = \Pr(t_n = c)\). Then \(P_{ij}(n) = \sum_{c = \delta(q_i, c)} D(c) = P_{ij}\). So, if we consider the DFA as a Markov chain over a finite state machine, the transition possibility \(P_{ij}(n)\) between \(q_i\) and \(q_j\) is independent with \(n\) which means that the DFA model here is a time-homogeneous Markov chain. Actually, large DFA can be seen as a non-decomposable Markov chain, whose states are all recurrent states, so a unique limiting probability distribution vector (stationary distribution) \(\pi = (P(q_1), \ldots, P(q_N))\) exists, where \(P(q_i)\) refers to the probability of state \(q_i\) being the current state and \(\pi\) satisfies the equation \(\pi = \pi M\). For minute discussion on periodicity and decomposability of Markov chain in large finite state machine, the reader can refer to [15].

B. Computing Average Speedup

For one single flow, the limiting probability of group \(Q_j\) that contains at least one current state is \(P(Q_j) = \sum_{q_i \in Q_j} P(q_i)\), where \(P(q_i)\) is the stationary possibility of state \(q_i\). Given \(r\) flows, the expectation of number of current states \(n_j\) is \(E(n_j) = rP(Q_j)\). Given the number of current states within some state group, the number of virtual active modules is decided by the probability distribution of class-set that partitioned by TA scheme. As mentioned in Subsection III-A, for every character \(c\) its probability of being the current one is \(P(c)\), and we denote the probability of \(C_i\) with \(P(C_i)\). Then, when matching a single flow, the limiting probability of class-set \(C_j\) containing the current state is \(P(C_j) = \sum_{c \in C_j} P(c)\).

Now given the number of concerned flows \(n\) (the number of concurrent flows in some group) and the number of class-sets \(t\), the theorem I gives the expected number of active modules within a certain state group.

**Theorem 1:**

\[
E(S) = t \left(1 - \sum_{i=1}^{t} \frac{P(C_i)_{\text{all}} - (P(C_i) - \frac{1}{r})}{\prod_{j=1, j \neq i}^{t} (P(C_j) - P(C_i))} \right)
\]

**Proof:** Let function \(sign(x) = 1\) if \(x > 0\) and \(0\) if \(x = 0\), then \(E(S) = E(\sum_{i=1}^{t} sign(x_i)) = \sum_{i=1}^{t} E(sign(x_i))\) under the constraint \(\sum_{i=1}^{t} x_i = n\) where \(x_i\) is the number of currently input characters in class \(C_i\). Notice that \(E(sign(x_i)) = 1 - \Pr(x_i = 0)\). If \(x_i = x\), the probability of a specific solution \((x_1, \ldots, x_t)\) of \(\sum_{j=1, j \neq i}^{t} x_j = n - x\) is \(\prod_{j=1, j \neq i}^{t} P(C_j)^{x_j}\). Sum them up, we get the probability of \(x_i = x\), that is \(\Pr(x_i = x) = P(C_i)^t \sum_{x_1+\cdots+x_t=n} \prod_{j=1, j \neq i}^{t} P(C_j)^{x_j}\), which is the coefficient \(\alpha_n\) in the polynomial \(\prod_{j=1, j \neq i}^{t} (1 - P(C_j) - \frac{1}{r})\). So \(E(S) = \sum_{i=1}^{t} E(sign(x_i)) = t - \sum_{i=1}^{t} \alpha_n(i) = t - \alpha_n\) where \(\alpha_n\) is the coefficient of \(u^n\) in polynomial \(\sum_{i=1}^{t} \prod_{j=1, j \neq i}^{t} (1 - P(C_j) - \frac{1}{r})\). Learnt from Lagrange interolation, we have \(\alpha_n = \sum_{i=1}^{t} \prod_{x_i \neq \hat{i}}^{N} P(C_{\hat{i}})^{x_i}\).

we now give two properties of great significance in designing a refined SA and TA scheme.

**Theorem 2:** For two possible distribution: \(P(C_i)\) and \(P'(C_i)\) for \(i = 1, \ldots, t\) only different in two classes, i.e., \(P(C_x) < P'(C_x) < P(C_y) < P'(C_y)\), then \(E(S)\) and \(E'(S)\) be the expectation of speedup under two distributions. Then \(E(S) < E'(S)\)

**Proof:**

\[
\frac{1}{2}(E' - E) = W_0 + W_x + W_y
\]

where \(W_0 = \sum_{i=1}^{t} \frac{P(C_i)}{P'(C_i)} P(C_i) - P(C_j)\) and \(W_i = \sum_{i=1}^{t} \frac{P(C_i)}{P'(C_i)} P(C_i) - P(C_j)\), \(i = x, y\). Considering that \(P(C_x) - P'(C_x) > P'(C_y) - P(C_y) > 0\), it is trivial to have \(W_0 + W_x + W_y > 0\).

**Theorem 3:** For two possibility distribution on \(k\) groups: \(P(Q_i)\) and \(P'(Q_i)\) for \(i = 1, \ldots, k\) only different in two groups, i.e., \(P(Q_x) < P'(Q_x) < P(Q_y) < P'(Q_y)\) and \(P(Q_i) = P'(Q_i), i \neq x, y\). Let \(E(S)\) and \(E'(S)\) be expected speedup of two distributions, then \(E(S) < E'(S)\)

**Proof:** Let \(w_j = \sum_{i=1}^{t} \frac{P(C_i)}{P'(C_i)} P(C_i) - P(C_j)\) > 0, we have \(E(S) = t \sum_{j=1}^{k} w_j P(C_i)^{rP(C_j)}\). Since \(0 < P(C_i)^r < 1\) and \(\sum_{j} P(Q_j) = 1\), which infers \(P(C_i)^r\)
is convex over $x$. Then according to Jensen’s Inequality, $E(S)$ is larger if $P(Q_j)$ is more average.

To sum up, Theorem 2 and 3 indicate that to achieve a high speedup, we just need to allocate the states and transitions so that $P(Q_j)$’s are as average as possible and $P(C_i)$’s are as average as possible. Together, they would be the discipline in designing the formal SA and TA scheme in next section.

IV. DESIGN OF SA AND TA SCHEME

For SA scheme, the limiting distribution $\pi = (P(q_1), \ldots, P(q_N))$ must be computed beforehand. Unfortunately, traditional method (such as Cramer’s solution) to compute $\pi$ by solving $\pi = \pi M$ costs a time complexity of $O(N^2)$, where $N$ is the number of states which is very large. Alternatively, we employ another approximating method which significantly reduces the time complexity to $O(N^2)$. Initially, $\pi(0) = (1, \ldots, 1)$, and $\pi(n+1) = \pi(n)M$. Exponentially ergodic property of states shows that $|P_{ij}(n) - P_{ij}| \leq (1 - N\delta)^n$ where $\delta = \min(P_{ij} : 1 \leq i, j \leq N)$. We define the “distance” between $\pi(n)$ and $\pi$ as $d(n) = ||\pi(n) - \pi|| = \left(\sum_{i=1}^{N} (P_{ii}(n) - \pi_i^2)\right)^{\frac{1}{2}} < N^{\frac{1}{2}}(1 - N\delta)^n$ where $1 - N\delta \ll 1$. So in finite steps, the result is sufficiently approximating to $\pi$. For TA scheme, the distribution is directly determined by allocation of character classes.

Now, we turn to the design of SA scheme, and then on TA scheme, assuming that the stationary distribution of state and classes is determined. According to Theorem 2 and 3, the aim of SA and TA scheme is to allocate the states and transitions so that $P(Q_j)$’s are as average as possible and $P(C_i)$’s are as average as possible. We introduce a “deviation” $\varepsilon$ and, for SA scheme, restrict that $|P(Q_j) - \frac{1}{N}| < \varepsilon$ and for TA scheme $|P(C_i) - \frac{1}{N}| < \varepsilon$, $1 \leq j \leq k$, $1 \leq i \leq t$. Algorithm 1 is designed to meet this purpose. The key step is to store state $q_i$ whose $P(q_i) > \varepsilon$ for multiple times. If the next state $q$ of certain transition is stored in $t$ memory modules, we randomly choose one of them to store and the probability of each state being chosen is thereby $P(q_i) < \varepsilon$ (in line 4). To bound the extra memory of state duplication, we denote the number of state $q_i$

```
Algorithm 1 Assigning $N$ states in $k$ group, $|P(Q_j) - \frac{1}{N}| < \varepsilon$
1: procedure ALLOCATION($q_1, \ldots, q_N, k, \varepsilon$)
2:    $N' \leftarrow 0$
3:    for $i = 1, \ldots, N$ do
4:        $t \leftarrow \left\lceil \frac{P(q_i)}{\varepsilon} \right\rceil$  $> P(q_i) / \varepsilon < \varepsilon$
5:        $q_{N'+1} \leftarrow q_i, \ldots, q_{N'+t} \leftarrow q_i$, $N' \leftarrow N' + t$
6:    end for
7:    for $j = 1, \ldots, k$ do
8:        $Q_j \leftarrow 0$, $P(Q_j) \leftarrow 0$
9:        while $P(Q_j) < \frac{1}{k}, i \leq N'$ do
10:           $Q_j \leftarrow Q_j \cup q_i$, $P(Q_j) \leftarrow P(Q_j) + P(q_i)$
11:              $i \leftarrow i + 1$
12:    end while
13:    end for
14:    return $Q_1, \ldots, Q_k$
15: end procedure
```

with $P(q_i)$ satisfying $j \varepsilon \geq P(q_i) > (j-1) \varepsilon$ as $a_j$, then the total number of states is $N' = \sum_{j=1}^{N} a_j = N + \sum_{j=1}^{N} (j-1) a_j < N + \frac{1}{2} \sum_{j=1}^{N} P(q_i) = N + \frac{1}{2}$. Generally, it would suffice to set $\frac{1}{k} = 10\varepsilon$, and then the upper bound of new adding states $N' - N$ is $10k(\sim 10^3)$ compared with $N(\sim 10^6)$. Hence, the extra memory is negligible even in the worst case. For TA scheme, it would not be so complex as SA scheme since the class number is much smaller than state number. We simply test all the cases which is $\leq N^3$ where $N$ is the class number.

V. CHIP DESIGN AND EVALUATION

We present a preliminary chip design of our matching system in Figure 3. The overall structure follows the formal DFA matching engine manner. Aggregated flows sent to Processing Module is first buffered in a certain Char FIFO in the Active Flow Buffer (AFB). The DFA is stored in $kt$ memory modules which process contents buffered in $r$ FIFOs in parallel. It should be noticed that the characters in active flows are first converted to class ID’s before being sent to FIFO. We deploy $r$ registers for each memory module of which the $i$th one always receives (class, state) pair from the $i$th FIFO (buffering the $i$th flow). In each cycle, every active module chooses one current state by round robin from its $r$ registers to process. After that, the results(next states) are fed back to be combined with the corresponding next character. The memory modules producing no next state send default signs back and all FIFOs without feedback are set “hold”. The round robin processing guarantees that all current states are processed within at most $r$ rounds. The upper bound of cycles that one flow aggregation is delayed in some module is the number of current states in the module. The round robin selection is accomplished during the memory access, so the overhead of each round comes only from the selectors. In details, under clock frequency of 166MHz, the memory access delay is 8 cycles\(^3\) and, at the same time, round robin algorithm selects the next input from the $r$ registers in 2 cycles. In addition, the selectors cost 2 cycles. Therefore, our system achieves the speedup at the cost of 2 cycles for flow and module selection.

\(^3\)Delay of memory access differs between various devices, here suppose it equals 8 clock cycles.
We evaluate the performance of our approach on the latest rule set version from Snort’s web pcre rule set [16] and L7-filter [17]. The patterns are first compiled into standard DFAs using the set splitting techniques proposed by [8] which split the pattern set into multiple subsets so that each subset creates a quite small DFA. For the DFA of each subset, we apply SA and TA scheme to assign transitions into kt memory modules where we let t = 2. We plot the result under various number of modules kt and ε = 1/20 in Table I. The table indicates that when k increases from 5 to 20, the total numbers of states of PDFA increase with a quite limited number and even the theoretical upper bound of increments is negligible (<3%).

The runtime test is conducted on 10 GB live traffic traces from campus gateway of Tsinghua University and randomly generated traces created according to 1-gram characteristic mentioned in Subsection III-A. The increase of speedup under given number of modules is plotted in Figure 4(a) and Figure 4(b), respectively. It is evident from the plots that as we increase the number of memory modules, the overall speedup scales up approximately linearly. For the purpose of comparison, we display the points labeling PDFA under various module number (see Table I) and other compression techniques can be employed. The points representing D2FA are obtained by using one instance so that no speedup is achieved. Meanwhile, the points labeling PDFA under various module number kt (k from 5 to 13 and t = 2) delineate a nearly uncompromising large speedup with little memory cost. The plots also reveal that PDFA achieves higher throughput compared with D2FA while their memory size is very similar.

<table>
<thead>
<tr>
<th>L7-filter (modules)</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DFA</td>
<td>51274</td>
<td>51274</td>
<td>51274</td>
<td>51274</td>
</tr>
<tr>
<td>PDFA</td>
<td>51290</td>
<td>51315</td>
<td>51377</td>
<td>51412</td>
</tr>
<tr>
<td>Upper bound</td>
<td>51324</td>
<td>51374</td>
<td>51424</td>
<td>51474</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Snort (modules)</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original DFA</td>
<td>720398</td>
<td>720398</td>
<td>720398</td>
<td>720398</td>
</tr>
<tr>
<td>PDFA</td>
<td>720431</td>
<td>720454</td>
<td>720499</td>
<td>721528</td>
</tr>
<tr>
<td>Upper bound</td>
<td>720448</td>
<td>720498</td>
<td>720538</td>
<td>720588</td>
</tr>
</tbody>
</table>

**TABLE I**

The number of states in PDFA using DFA allocation algorithm on rule sets of L7-filter and Snort.

VI. ACKNOWLEDGEMENT

This work is supported by NSFC (60625201, 60873250, 60903182), 973 project (2007CB310701), 863 high-tech project (2007AA01Z216) and Tsinghua University Initiative Scientific Research Program.

VII. CONCLUSION

In this paper, we propose a parallel architecture of DFA called PDFA which utilize the hardware parallelism to achieve an increased throughput. The speedup is obtained with little memory redundancy compared with the existing methods. Theoretical analysis and experimental evaluation both shows a good performance of PDFA on average situation. Even under worst cases, as analyzed in Section V, the increment of memory size is negligible.

**REFERENCES**


