Cache-Based Scalable Deep Packet Inspection with Predictive Automaton

†Yi Tang, †Junchen Jiang, ‡Xiaofei Wang,†Yi Wang, †Bin Liu
†Tsinghua National Laboratory for Information Science and Technology
†Department of Computer Science and Technology, Tsinghua University, China
‡Department of Electronic Engineering, Dublin City University, Ireland

Abstract*—Regular expression (Regex) becomes the standard signature language for security and application detection. Deterministic finite automata (DFAs) are widely used to perform multiple regex matching in linear time. However, when implemented by modern memories, the matching speed turns out to be a tradeoff with the size of DFA. To improve the performance, we propose a generalized caching scheme that strike the boundaries of memory size. We define the concept of local prediction which predicts the memory accesses to the DFA and guides the cache to be replaced with proper states so that the cache hit rate is greatly raised. The idea of using predictive DFA matching specifies an entire new class of approaches. We also develop techniques to intelligently store the DFA using local prediction so that given any replacement policy, the caching scheme would produce nice performance. Evaluation shows that our storage achieves a 30% higher cache hit in comparison with the previously proposed approaches. Especially, our approach appears to be more robust when processing malicious traffics. By analyzing numeric results, the optimal configurations that achieve the ultimate performances in various traffic are provided.

***Index Terms*—Pattern Matching, Deep Packet Inspection**

I. Introduction

Deep Packet Inspection (DPI) has recently found several critical applications dealing with intrusion detection, keyword blocking and anti-virus. Deterministic Finite Automata (DFAs) have become a popular matching technique because it can match multiple signatures simultaneously with a guaranteed worst-case performance of $O(1)$ time per character. However, when implemented by modern memories, the matching speed turns out to be compromised by the prohibitively large sizes of DFAs. That is, to store a single large DFA requires a large memory in size, causing an inherently low memory access frequency.

To improve the performance, the hierarchical scheme of memory is an inevitable choice as it strikes the boundaries of memory sizes by transparently storing data in small but fast memories, called *caches* such that future requests for that data can be served faster. The caching schemes have been proven surprisingly efficient in several areas of computer science. However, to directly adopt the hierarchical scheme to DFA matching improves the performance with very limit acceleration compared with the large potentials in speedup of caches. In addition, the current caching schemes for DFAs either use fixed-content caches or use dynamic caches on untreated DFAs, both of which are proven vulnerable when processing malicious traces. From our point of view, the major obstacle of adopting caches lies in the fact that it is quite difficult to draw any pattern on the memory accesses of DFAs due to the large sizes of today's DFAs and the arbitrariness of the incoming traffic.

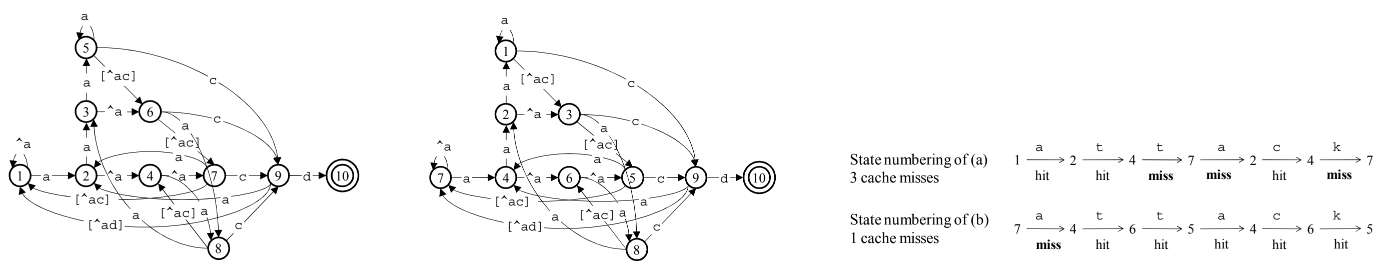
Generally speaking, the rationale of cache-based acceleration is the prediction on the requests to a large database in a limited period of time. In this paper, we propose the concept of *local prediction* which shows the potentials in predicting DFA matching. Basically, local prediction reveals the fact that at different processing moments, one transition has a distinct probability to be accessed in the near future. These messages enable us to predict the memory accesses to the DFA and to store proper states into caches so that cache hit rate is greatly raised. The idea of using predictive DFA matching specifies an entire new class of approaches. In summary, we make the following contributions:

1) We propose the concept of local prediction which enables designers to do prediction on the behavior of DFAs. Basing on the concept, we give a generalized model of caching schemes which attempts to use the local prediction to improve the performance, even in worst cases.
2) We separately study the design issues of the caching schemes from the aspects of policy and mechanism. Major considerations of caching policies are discussed on the basis of statistical results. We develop pre-computing techniques to intelligently store the DFA so that given any runtime cache replacement, the caching schemes would produce nice performance.
3) The caching scheme model is studied by both theoretical analysis and simulation results. Evaluation shows that our storage achieves a 30% higher cache hit and more robust when processing malicious traffics.

This paper is organized as follows. The related works is presented in Sec. II. Sec. III gives some background knowledge and the definition of local prediction.Sec. IV generalizes the idea of the example to have a general caching scheme. Sec. V develops the techniques to intelligently store the DFA and Sec. VI gives the numeric results of the proposed system. Finally, Sec. VII concludes the paper.

II. Related Work

In high-speed networking applications, DFA is used widely to represent regular expression. Regular expression line rate



(a) State numbering in breadth-first order (b) State numbering using local prediction (c) Trace "attack" for the matching of regex

Fig. 1. DFAs for $P = $ `.*a.{2}cd` with different state numberings and their effect.

matching has been recognized as an important issue and well be studied. A lot of FPGA based implementation [1]–[3] and general purpose processor and ASIC hardware [2], [4]–[7] have been proposed.

In [8], Tseng proposes a specialized cache automata matching circuit to accelerating network process. His approach focuses on design and implementation circuit for multi-character matching. In [9], [10], Becchi gives a evaluation benchmark includes regular expression model and traffic generator for the evaluation of different regular expression architectures. In [11],Song mentions to use cache with DFA, and proposes a scheme named "cached deterministic finite automate (CDFA)". But the cache in his paper is not used as a match accelerator. It helps to reducing the memory space based on the observation of a lot of transitions will point to the start state of DFA. The author eliminates these back pointing transitions through a cache doing the matching job from start state every cycle.

III. Technical Overview

Traditionally, caching schemes are used to strike this inherent tradeoff by transparently storing data in small but fast memories, called *caches*, such that future requests for that data can be served faster. If the requested data is contained in the cache (*cache hit*), this request can be served by simply reading the cache, which is comparably faster. Otherwise (*cache miss*), the data has to be recomputed or fetched from its original storage location, which is comparably slower.

A. Local Prediction

We use a concept of *local prediction* as the basis of further analysis and design. In later discussion, we will give examples and explain the rationale of local predictions. Here is the formal definition.

Definition 1: Given a set of elements $A = \{a_1, \dots, a_n\}$ and an integer d, the elements are requested by a sequence $s_1, s_2, \dots$ where $s_i \in A$. The local prediction of A with the *predicted diameter* k (for simplicity LP_d) maps all element pairs (a, a') to $[0, d]$, $a, a' \in A$, so that $LP_d(a|a') = E(\#\text{ of } s_j = a | s_i = a', i < j \le i + d)$, which is the expectation of the number of request for a in the d data requests after a request for a'.

Intuitively, the larger the $LP_d(a|a')$ is, the more likely that element a' will be requested within the next d requests after a is just requested, i.e., more cache hits by requests for a' may be obtained.

B. Example of Using Local Prediction

We show a simple idea of using local predictions to accelerate DFA matching in this section. The core idea is to treat all states as elements and to store the states with high local predictions as close as possible such that they tend to be appear together in the cache, reducing the cache miss rate since a high local prediction implies a high probability for them to be accessed within a short time. We assume that the cache always replace its content with a contiguous part of the secondary memory (*main memory*), because it would largely reduce the overhead for replacing the data in the cache. Notice that the cache does not necessarily update all its contents.

As an example, consider the DFA recognizing the regex $P = $ `.*a.{2}cd` [1]shown in Fig. 1. Suppose we have one cache which contains at most five states, and once a request for a transition of the state i causes a cache miss, the cache will replace its content with the five states around state i, i.e., all state j such that $|j - i| \le 2$. In Fig. 1-(c), we illustrate how different state numbering methods impact the performance of caching schemes. The states of the DFA are numbered by two methods: breadth-first manner in (a),[2] and based on knowledge of local predictions in (b). To see how local predictions are considered in the state numbering of (b), we notice that each state has one or two transitions which are labeled only one character while the rest one is labeled with all other characters (e.g., in (a), state 4 has one transition of `a` to state 8 and another one of $\wedge$`a` (not `a`) to state 7). Such asymmetry exactly reflects the local predictions with predicted diameter 1. For example, in Fig. 1-(a), if the current state is state 4, the next state is most likely to be state 7, not state 8, since the probability of receiving an `a` as the next character is relatively small. And by local prediction, this simple observation is interpreted by $LP_1(7|4) \gg LP_1(8|4)$, which is exactly why the state numbering in (b) gives two close numbers to state 4 and 7, i.e., the state 6 and state 5 in (b). For other state pairs, we use the similar idea to give close numbers if their local predictions are large, which on the surface means that the transition between them is labeled with a large character set (e.g., $\wedge$`[ac]`, $\wedge$`c`).

[1]We use JFlex syntax for regex in this paper, i.e., "." and "*" means the wildcard and any number of matchings respectively. In default, we use the 8bit input character set.

[2]In fact, the breath-first numbering of states is the direct result of using the classical subset construction method to build a DFA

Fig. 1-(c) shows a trace of the matching the input $I =$ `attack` against the DFA under the two state numberings. In our cases, the numbering of Fig. 1-(a) results in three cache misses while that of Fig. 1-(b) results in one cache miss.

IV. A Generalized Caching Scheme with Local Prediction

In this section, the example of Subsec. III-B will be generalized to get a basic model of the caching scheme. We will explain the rationale of designing caching scheme using local predictions and clarify the influence of various parameters by studying statistical evidences. Fig 2 depicts the structure of the proposed caching scheme which will intensively studied in the discussion that follows.

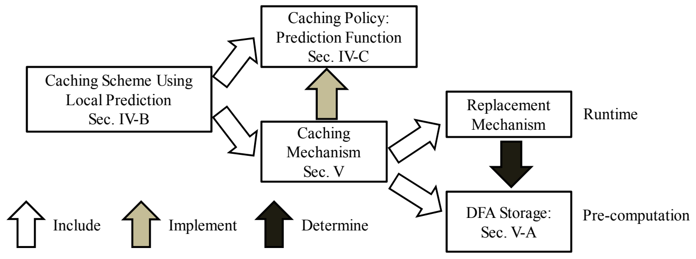


Fig. 2. The generalized structure of caching scheme.

A. A Basic Model

Using local prediction to design caching schemes for DFA matching is a general method. Depending on the hardware platform, the storage of DFA, the replacement policy, or other parameters, one can have a wide variety of caching schemes using local prediction. We design the caching scheme using a separation of *caching policy* and *caching mechanism*. The caching policy, that gives the objective of mechanism designing, concerns what should be stored in cache so that the performance can be optimized. The caching mechanism involves the implementation of the caching policies during both runtime stage and pre-computing stage. The runtime processing decides which part of cache should be preserved and which should be replaced, in the case of cache miss. To better fit a given replacement mechanism, in pre-computing stage, the DFA are stored such that the contents to be stored in cache by runtime replacement do raise the cache hit rate.

Fig. 3 shows the generalized model of caching scheme that we use in this paper. We split the cache into two parts, *local segment* and *global segment*. For each access to transition table, the request is served by both the cache and the main memory. In case of cache miss, the contents in local segment are replaced while those in global segment are preserved.

B. Local prediction vs. global prediction

There are two cases where the gains of the caching scheme might be compromised: (i) the elements are requested in complete randomness; (ii) each datum is requested with a fixed probability which varies between different datum. The two cases essentially indicate another extreme to the local prediction. In the two cases, we see a stable probability of each element to be requested, so a fixed cache containing elements with highest probability is the best. Similar to the local prediction, we define the *global prediction* as the probability of each element to be requested *at any time*. We denote the global prediction of element a by $GP(a)$. If all $GP(a)$ are the same, it is the case (i), otherwise, the case (ii). Basically, global prediction specifies a class of element a of which the $LP_d(a|a')$ is insensitive to a', since $LP_d(a|a') = \sum_{j=1}^{d} GP(a)^j$. We call the elements that hold the above property, *global elements*, and otherwise, *local elements*.

In DFA matching, one receives a sequence of transitions. We treat them as elements and accordingly define *global transitions* and *local transitions*. By studying the statistical evidence of DFA matching in DPI, we find that the common sense is neither the local prediction nor the global prediction, but a hybrid one. After splitting the transition set into global ones and local ones, we have three observations:

1) Global transitions make up of a very small part of the whole transition set.
2) The local predictions of most local transitions differ greatly, i.e., the distribution of $LP_d(a|a')$ over all a's are quite different between various a's.
3) Under randomly generated traces or normal traces, the global transitions are requested with high probabilities, while in malicious traffics, such probabilities are greatly reduced.

For instance, we construct a DFA with three Snort rules, and plot in Fig. 4(a) the statistical support for Observation 1 and 3 where the upper curve shows that about 10% of states amount for more than 70% memory access under normal traces while the lower curve indicates that under malicious traces, the accesses of states become more diverse. In Fig. 4(b) which is conducted on the same DFA, we confirm the Observation 2 by using three example states. The Fig. 4(b) gives the sum of local predictions of different part of DFAs which is split into 10 parts with 200 states in each. It is shown that during the next 6 requests after each of them is requested, each part of the DFA are accessed with very different probabilities.

C. Basic Principles and Objective for Caching Policies

Observation 1 leads to a cache containing those global transitions as a part. Since signature matching for DPI is possibly facing various attacks including ones aiming to slowdown the

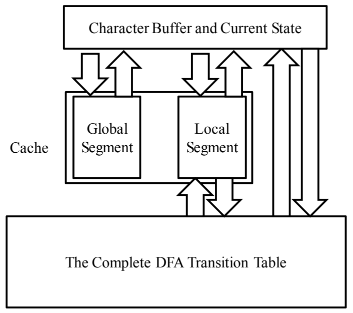
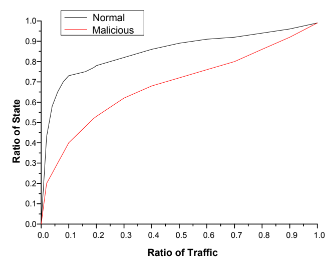
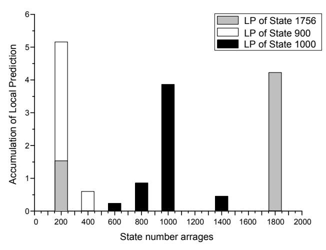

Character Buffer and Current State
Cache
Global Segment
Local Segment
The Complete DFA Transition Table

Fig. 3. The generalized caching scheme where the cache consists of the local segment and the global segment.

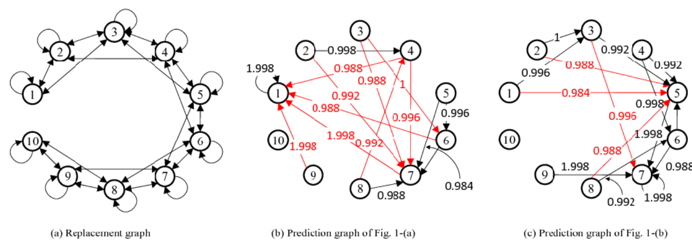


(a) Probability of access on each state under normal and malicious traces.



(b) The sum of local prediction in distinct parts of DFA.

Fig. 4. Statistical support for the observations.



(a) Replacement graph
(b) Prediction graph of Fig. 1-(a)
(c) Prediction graph of Fig. 1-(b)

Fig. 5. The graphic representations of the DFA storage with the replacement graphs and the prediction graphs. (b) and (c) depict the prediction graphs of the state numberings in Fig 1-(a),(b) respectively. Numbers on the arrows show the weights (weights less than 0.1 are not ignored, for simplicity). The numbering using prediction yields a larger predictive function value compared with breadth-first order.

matching throughput, we shall greatly focus on Observation 3 under malicious traffics, inferring a capability of cache to track local transitions, even when the input forces the DFA to not travel global transitions. Observation 2 shows the importance of using local segments, since it shows that it is hard to tracking local transition with fixed cache like that of global transitions.

Conclusively, given the size of the cache, caching policy for DFA matching needs to handle the following balances:

1) The balance of preserving more global transitions with high $GP(a)$ in cache on one hand and updating with more contents to track more local transitions in prevention of attacks on the other;
2) The balance of speculating for more steps (i.e., using $LP_k(a|a')$ with larger k) to avoid cache misses in longer period of time on one hand and speculating for less steps but with raised predictive accuracy such that cache hits in near future are increased on the other.

V. Local Predictive DFA

We describe the problem using the language of graph theory and then equivalently transform it into a classical graph matching problem. We first construct two graphs $G_i = (V_i, E_i), i = 1, 2$. The vertices sets V_1, V_2 refer to the set of the addresses and the transitions of DFA respectively, so V_1, V_2 have the same number of elements. G_1, called *replacement graph*, is used to express the replacement mechanism over a given addressing space, and G_2, called *prediction graph*, is a weighted graph with weight function W and is used to express the local predictions.

In G_1, we add a directed edge from vertex v to v' if and only if contents in address v' would be written in the cache when a request for address v turns out to be a cache miss. In G_2, for any vertices pair v, v', we add a directed edge (v, v') with a weight $W(v, v') = p(v)LP_k(v'|v)$ where $p(v)$ is the prior probability of transition v. The DFA storage problem is now translated into the problem of finding the subgraph of G_2 with the largest weight that is isomorphism to G_1. Since G_1 and G_2 have same number of vertices, we need only to give a mapping R from E_2 to E_1 such that the predictive function

$$\max P = \sum_{v,v' \in V_2} p(v)LP_k(v'|v)e(R(v), R(v'))$$

where $e(R(v), R(v')) = 1$ if $e(R(v), R(v')) \in E_2$, otherwise, $e(R(v), R(v')) = 0$.

For illustrating the idea, we use the example in Subsec.III-B, where in cache miss, the caching policy allows the cache to replace all its contents with all transitions of at most 5 contiguous states around the next state and the predicted diameter is 2. We assume that the local predictions are calculated on the base of all character has an equal probability appear and the prior probability of each transition is the same. Fig. 5-(a) shows the replacement graph, and Fig. 5-(b)(c) give the prediction graphs where only edges with weight larger than 0.1 is presented. We separately compute the weights of induced subgraph under the two state numberings in Fig. 1-(a),(b). The predictive function of numbering by breadth-first order is about 7, while for numbering using local prediction, the predictive function is much larger, about 13.

VI. Evaluation

We display the numerical results on the performance of our approach. The cache hit rate is evaluated under various combinations of parameters, including the proportion of local segment in the cache and the predicted diameter. For showing the robustness, we will compare the cache hit rate of our approach with the results of using cache on untreated DFAs.

The storage of DFA is based on a transition-block scheme where all transitions of one state are stored in one contiguous block of memory called *transition block* (*TB*). Accordingly, the cache store and replace the contents by using TB as the unit. In our experiments, the cache always replace the local segment with TBs belonging to contiguous addresses.

A. Experiment Setup

We perform experiments on our prototype system under common NIDS data-base. The rulesets are extracted from Snort [12] HTTP ruleset(Snort ruleset from V2.8.4: 17 June 2009). The DFA comes from the combination of 30 regex patterns(randomly selected from the ruleset) and contains 60644 states with 3517352 transitions. We use the real trace captured at the gateway of the graduate student dormitory buildings of Tsinghua University in China with the size of 596MB. We also simulate the environment of malicious traffic by generated traces where more than 20% sessions match some signatures

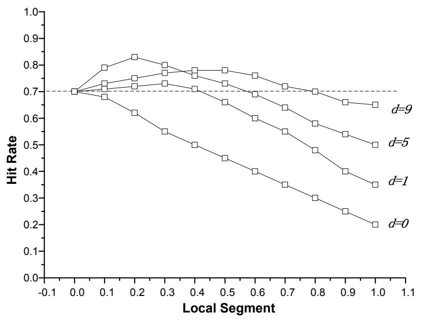


Fig. 6. The cache hit rates for different diameter in normal trace.

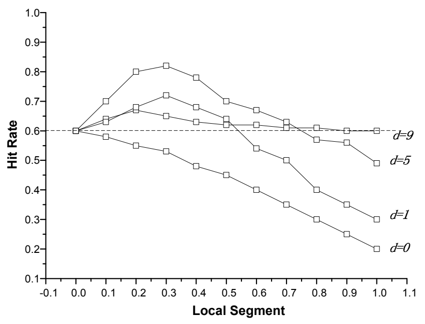


Fig. 7. The cache hit rate for different diameter in malicious trace.

in the aforementioned rulesets. The caching schemes are implemented by software and we use a cache with 10% size of the whole DFA in all experiments.

B. Cache Hit Rate

The cache hit rate is investigated under various configurations of predicted diameters and proportions of local segments. Fig. 6 shows the different trends of cache hit rate with the increasing proportion of local segments, when predicted diameter varies from 1 to 9. For each curve, when the proportion of local segments becomes close to 0, it turns out to be a fixed content cache, and when the ratio increase to 1, the cache decides its contents by completely using local predictions. The plots clearly curve the downtrends near the two extremes and we observe that the optimal proportion of local segment is around $30\% \sim 50\%$.

For the curves of different predicted diameters, the cases become more complicated. First of all, we observe that the optimal results come from the curve of predicted diameter 5, whose ultimate hit rates are improved about 10% compared with that of traditional fixed contents caches. For diameters less than 5, they result in sharp decreases when more cache space is local segments. A possible explanation is that with low predicted diameters, the cache always replaces its local segment with shortsighted predictions, which may lead to more cache misses in large time scale. For diameters larger than 5, we observe that the gains of local prediction are weaken; for example, the curve of $d = 9$ makes little improvement to the fixed cache scheme, because the local prediction will certainly prefer global transitions which are already stored in cache. Finally, we use $d = 0$ to show the naive solution of breadth-first numbering with no prediction. Evidently, its performance is at most the same good as that of fixed cache scheme.

C. Robustness

In essence, the local segments of caches endow the fast but small memories with the flexibility of dynamically tracking and serving the part of transitions that are most likely to be requested. We test the robustness of our scheme under simulated malicious traces by replaying all the test configuration in Fig 7. It is clearly shown that our approaches have more enhanced robustness since the optimal performance of our scheme decreases very little in comparison with the sharp reductions of fixed cache schemes and non-predictive numbering.

VII. Conclusion

In this paper, we propose the concept of local prediction which enables us to do prediction on the behavior of DFAs. Basing on this, we give a generalized model of caching schemes which attempts to use the local prediction to yield the high performance. We develop techniques to intelligently store the DFA using local prediction so that given any replacement policy, the caching scheme would produce nice performance.

Acknowledgement

This work was supported in part by NSFC (60625201, 60873250, 60903182); 973 project(2007CB310701); 863 high-tech project (2007AA01Z216) and Tsinghua University Initiative Scientific Research Program; the National Natural Science Foundation of China Grant Nos.(60604033,60553001).

References

[1] J. Moscola and et al., "Implementation of a content-scanning module for an internet firewall," in *Proc. of FCCM*, 2003.
[2] B. C. Brodie and et al., "A scalable architecture for high-throughput regular-expression pattern matching," in *Proc. of ISCA*, 2006.
[3] C. Clark and D.Schimmel, "Efficient reconfigurable logic circuit for matching complex network intrusion detection patterns," in *Proc. of FLP*, 2003.
[4] M. Becchi and P. Crowley, "An improved algorithm to accelerate regular expression evaluation," in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communication(ANCS)*, 2007.
[5] L. Tan and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," in *Proc. of ISCA*, 2005.
[6] S. Dharmapurikar and J. W. Lockwood, "Fast and scalable pattern matching for network intrusion detection engines," *IEEE JSAC*, vol. 24, no. 10, 2006.
[7] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," in *Proc. of ANCS*, 2006.
[8] K. Tseng, C. Wang, and Y. Liao, "A specialized cache of automata matching for content filtering," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2009.
[9] M.Becchi, M.Franklin, and P.Crowley, "A workload for evaluating deep packet inspection architectures," in *Proceedings of the 11th IEEE International Symposium on Workload Characterization*, 2008.
[10] M. Becchi, C.Wiseman, and P. Crowley, "Evaluating regular expression matching engines on network and general purpose porcessors," in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communication(ANCS)*, 2008.
[11] T.Song, W. Zhang, D. Wang, and Y. Xue, "A memory efficient multiple pattern matching architecture for netowrk security," in *Proceedings of the IEEE INFOCOM*, 2008.
[12] Snort, http://www.snort.org/.