Journal of Network and Computer Applications 54 (2015) 78–87

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Speculative parallel pattern matching using stride-*k* DFA for deep packet inspection

Maleeha Najam [a,*], Usman Younis [a], Raihan ur Rasool [a,b]

[a] School of Electrical Engineering and Computer Science, National University of Sciences and Technology, H-12, Islamabad 44000, Pakistan
[b] Department of Computer Science, King Faisal University, Al-Hofuf, Saudi Arabia

ARTICLE INFO

Article history:
Received 22 August 2014
Received in revised form
28 November 2014
Accepted 29 April 2015
Available online 11 May 2015

Keywords:
DFA
Regular expressions
Multi-byte matching
Alphabet compression
Deep packet inspection

ABSTRACT

Modern deep packet inspection (DPI) systems match network traffic against a large set of patterns which are defined using regular expressions. Deterministic finite automata (DFA) is generally preferred to parse these regular expressions. However, packets are mostly scanned one byte at a time which becomes a bottleneck for the DPI systems as they are unable to cope up with higher line rates. In this paper, we present an approach which allows a packet to be split up into two chunks. Furthermore, we scan the bytes of each chunk in parallel using speculation and multi-stride (stride-*k*) DFA. Stride-*k* DFAs results in fast processing of bytes but leads to high memory usage. Therefore, we propose a transition compression algorithm using alphabet compression table to limit the memory usage of multi-stride DFA. Experimental results show that the speculative parallel pattern matching using stride-*k* DFA leads to improvement in terms of speedup and latency over traditional DFA matching.

1. Introduction

Deep packet inspection (DPI) has become an attractive choice for inspecting network packets. This technology is being widely adopted by network intrusion detection systems (IDS), intrusion prevention systems (IPS), firewalls, and many other traffic monitoring applications (Roesch, http://www.snort.org; http://l7-filter.clearfoundation.com/start; Ahmad and Younis, 2014). Earlier, packets were used to be classified on the basis of their header fields, where as DPI based network monitoring systems allow inspection on the basis of packets' payload. It searches for predefined patterns inside payload which are either expressed as simple strings or in the form of regular expressions. Regular expressions are highly preferred over string patterns as they provide much more flexibility and expressiveness as compared to fixed strings (Wang et al., 2012).

Modern IDSs, such as, Snort, Bro, and L7 filter, make significant use of regular expressions in their matching engines. Regular expressions are expressed using finite automata as it is considered to be the most efficient mechanism to detect patterns. The two most widely used finite automata representations of regular expression are deterministic finite automata (DFA), and non-deterministic finite automata (NFA). In both NFA and DFA, each state is defined for all the elements present in the alphabet. In NFA, each state triggers one or more than one state transition for each unique element in the alphabet. Unlike NFA, a state in a DFA triggers exactly single transition for each element in the alphabet. Therefore, DFAs are considered to be processing efficient with the worst case processing complexity of $O(1)$ for each input character, as compared to NFA which has $O(n^2)$ worst case complexity – n being the length of the regular expression (Yu et al., 2006). On the other hand, NFAs are considered to be more space efficient with the worst case storage cost of $O(n)$, as compared to DFA which has $O(2^n)$ worst case storage cost.

Mostly, DFAs are used in pattern matching engines due to their ability to do high speed matching in fast-pace networks. Therefore, researchers are putting their efforts to improve the DFA based matching engines by focusing on reducing the memory footprint as it is the major drawback of DFAs; whereas less work is being done to speed up the look-up process in DFA based matching engines. Some hardware based approaches, reported in Lunteren et al. (2012), Brodie et al. (2006) and Becchi and Crowley (2007), have been proposed for pattern matching which either make use of field-programmable gate arrays (FPGA) or application-specific integrated circuits (ASIC) to achieve high throughputs in DFA based matching engines. However, they do not provide the type of flexibility which the software based approaches can provide when it comes to update the regular expressions.

In IDS/IPS, the incoming packets are mostly scanned one byte at a time which limits the throughput and latency of the system and creates a bottleneck for the network traffic which is having speeds

* Corresponding author.
E-mail addresses: 12mseemnajam@seecs.edu.pk (M. Najam), usman.younis@seecs.edu.pk (U. Younis), raihan.rasool@seecs.edu.pk, rrasool@kfu.edu.sa (R.u. Rasool).

http://dx.doi.org/10.1016/j.jnca.2015.04.013
1084-8045/© 2015 Elsevier Ltd. All rights reserved.

of tens of gigabits per second (Luchaup et al., 2009). However, some new approaches make use of multi-stride DFAs to speed up the packet matching engines - where stride refers to the number of bytes processed per state transition. However, memory requirements increase when stride level is increased. For instance, with the alphabet size of 256, stride-1 DFA has 256 transitions per state. Its corresponding stride-2 DFA, which can process two bytes at a time, will have 256^2 transitions per state. Speculative parallel pattern (SPPM) matching is one approach which speeds up the evaluation of regular expressions against packets without increasing the number of strides. The core idea behind SPPM is to divide the incoming packets stream into two chunks and scan one byte from each chunk at a time.

The contributions of this paper can be summarized as follows: (1) we have designed a transition compression algorithm for the multi-stride DFA, and (2) we have proposed an algorithm which makes use of SPPM on multi-stride DFA structures. The performance of our proposed algorithm has been evaluated using the real and artificial network traces for regular expressions extracted from Snort and L7 filter. The remainder of this paper is organized as follows. In Section 2, we present the related work. In Section 4, we propose the transition compression algorithm for multi-stride-DFAs. In Section 5, we discuss the algorithm which makes use of SPPM on multi-stride DFAs. In Section 6, results of performance evaluation of the algorithms are discussed. Finally, the paper is concluded along with future work in Section 7.

2. Related work

The string based pattern matching has been the most simple network packet inspection technique which mostly make use Aho-Corasik or Boyre Moore algorithms (Aho and Corasick, 1975; Boyer and Moore, 1977). Though, string based matching is still used during pre-processing stages of packet inspection, as it speeds up the process, but in the past few years a widespread use of regular expressions has been observed in many network monitoring applications. Regular expressions are often implemented either using DFAs or NFAs. DFAs give rise to large memory usage but need just one memory access per character, whereas NFAs offer low memory cost but have high memory bandwidth, i.e., large number of memory accesses per character. Therefore, as discussed earlier, DFAs are mostly preferred and research is still going on to reduce the memory footprint of DFAs and speed up the look up process to make the implementation of DFAs feasible in real-time systems.

A number of FPGA and ASIC approaches have been proposed which can provide higher throughput due to the parallelism offered by the hardware (Lunteren et al., 2012; Brodie et al., 2006; Becchi and Crowley, 2007). These higher throughputs are difficult to be achieved using software based techniques which make use of memory to store automaton. However, FPGAs have a drawback when it comes to update the regular expressions circuitry to support the detection of new viruses and threats. Moreover, some attempts are also made using Graphics Processing Unit (GPU) for the same purpose as they are cost effective and provide very high memory bandwidth (Yu and Becchi, 2013; Vasiliadis et al., 2008, 2009; Smith et al., 2009; Lin C-H et al., 2011).

In memory based approaches, alphabet reduction techniques are often employed for reducing DFA storage requirements (Brodie et al., 2006; Kong et al., 2008; Becchi and Crowley, 2007). Alphabet reduction is often based upon the observation that the distinct symbols of alphabets, over which the DFA is defined, often share identical behavior among different states of the DFA. This allows forming equivalent classes or groups of symbols which share the same behavior. Afterwards, each state is represented by the classes of the alphabets instead of their symbols. Moreover, authors in Becchi and Crowley (2007) have proposed a hybrid scheme which makes use of both NFA and DFA for the construction of automaton. This approach also leads to a suitable memory requirement and an average memory bandwidth.

Exponential growth in the number of DFA states is often seen when multiple complex regular expressions are compiled to generate a composite DFA. Yu et al. have proposed a grouping algorithm for regular expressions and some rule re-writing techniques to mitigate this memory blow-up problem in Yu et al. (2006). Similarly, state merging technique has been reported in Becchi and Cadambi (2007), which results in significant memory reduction by allowing non-equivalent states in a DFA to be merged. Transition compression techniques for DFAs have been used to reduce the memory footprint in Becchi and Cadambi (2007), Liu T et al. (2011), Becchi and Crowley (2007) and Kumar et al. (2006)). These techniques also achieve a better pattern matching performance, i.e., if DFA's size is reasonable then a large number of DFAs can fit into processor's cache which results in small number of main memory accesses and a reduced latency.

In general purpose processor, the throughput of any pattern matching application is limited by memory bandwidth. Therefore, some work has also been reported in the area of multi-byte matching to increase the throughput of the system by consuming multiple bytes per state transition (Brodie et al., 2006; Becchi and Crowley, 2008; Wang et al., 2011). Multi-byte approaches sometimes result in exponential increase in the alphabet size which leads to an increase in the transition table size. Therefore, alphabet encoding, run-length coding, and many other transition compression techniques are used to mitigate these effects of memory blow-up. Unlike multi-byte approaches, sampling technique has been introduced in Ficara et al. (2010), which skips large portions of text while scanning input. This results in the processing of lesser bytes, however a confirmation stage is required as this technique may lead to false alarms.

Another approach which focuses on improving throughput and reducing latency is SPPM (Luchaup et al., 2009, 2011). This approach achieves significant speed-up by dividing the input stream into two chunks, and then scanning one byte each from both the chunks in parallel. Different SPPM based algorithms for single-threaded software running on commodity processors as well as for parallel hardware are proposed in Luchaup et al. (2009, 2011).

3. Problem formulation

A multi-stride DFA requires more than one byte at a time to trigger a state transition which leads to high throughput in the evaluation of regular expressions. However, it is often observed that DFAs show a trade-off between memory and run-time execution. This trade-off is a major bottleneck for multi-stride DFAs, as the memory requirements for multi-stride DFAs are much greater than that for the single-stride DFAs. For example, a stride-1 DFA defined over 256 ASCII characters would require 256 transitions per state, whereas its corresponding stride-2 DFA will require 256^2 transitions per state. Now, suppose that we have a DFA for a complex regular expression set which comprises of 1000 states. In this case, total transitions for the stride-1 DFA and stride-2 DFA will be 256,000 and 65,536,000, respectively. Similarly, if we keep on increasing the stride level, the number of transitions will also keep on increasing, which results in increased memory usage. Therefore, it is important to overcome this issue as memory is a critical resource in software based approaches, also known as memory centric approaches. Several DFA compression algorithms have been proposed earlier, but sometimes a compression algorithm may increase the number of memory accesses resulting in slow-down of the matching engine (Kong et al., 2008). To cater all these problems, we propose a solution for the multi-stride DFAs