Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 110 (2017) 250–257

Procedia Computer Science

www.elsevier.com/locate/procedia

The 12th International Conference on Future Networks and Communications (FNC 2017)

A Memory Efficient Pattern Matching Scheme for Regular Expressions

Yeim-Kuan Chang* and Ching-Hsuan Shih

Department of Computer Science and Information Engineering National Cheng Kung University Tainan, 701, Taiwan

Abstract

Many malicious packets are common and spread over the Internet in recent years when the scale of Internet traffic grows at a rapid speed. Thus, the regular expression matching in Network Intrusion Detection System (NIDS) that supervises network activities needs to be very fast and consume small memory. In this paper, we propose a memory efficient regular expression matching algorithm called Failureless Segmented Finite Automata (FSFA) with an acceptable searching speed. In FSFA, We eliminate Kleene closures by using additional data structures to reduce a large amount of states. We further reduce the transitions by using default state compression technique. Our performance results implemented on a PC software environment show that our scheme only needs 1% of states needed in DFA and 2 to 22% of states needed in JFA.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: Regular expression; DFA; State explosion; Network Intrusion Detection System

1. Introduction

Since the Internet becomes extremely popular as well as universal and the traffic of the Internet is growing very fast, more and more malicious attacks and viruses sprawl over the Internet every day. Therefore, network security systems such as firewall, Network Intrusion Detection System (NIDS) and antivirus software are developed to avoid our computers from being attacked over the Internet. Antivirus software and NIDS such as Snort[1] and Bro[2] examine the payloads of Internet packets, known as Deep Packet Inspection (DPI), with predefined rules.

* Corresponding author. Tel.: +886-6-275-7575 ; fax: +886-6-274-7076 .
E-mail address: ykchang@mail.ncku.edu.tw

1877-0509
Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2017.06.092

Pattern matching is the act of checking a given sequence of tokens for the presence of some patterns in the rule sets. Because of its expressive power, regular expression has been used commonly in current rule sets. Deterministic Finite Automata (DFA) is a method that implements regular expression matching. Using DFA for regular expression matching only needs to visit one state when processing a single input character and achieves very fast matching speed but suffers from the well-known state explosion problem due to Kleene closures such as ".*" and "[^\n]*" appearing in the complex rules. Therefore, recently, there are many schemes which are devoted to address the state explosion problem and reserve acceptable searching speeds comparing to DFA.

In this paper, we focus on processing complicated regular expressions that contain Kleene closures and propose a new memory efficient scheme to address the state explosion problem. And, we compress the transition table using default state compression technique.

The rest of this paper is organized as follows. In section 2, we introduce some related works and background about regular expression matching. In section 3, we present our proposed scheme in details and show how we improve memory consumption of DFA. We will illustrate the data structures and the searching procedure of our scheme. Section 4 shows the experimental results. Finally, we conclude this paper in the last section.

2. Related Work

There are many regular expression search engines[6,7,8,17,18,19] proposed in the literature. Current hardware-based methods are usually implemented on FPGA[9,10,11], TCAM[12,13,14], ASIC and so on. The most recent two schemes, XFA[3,4] and JFA[5], are devoted to solving the problem of excessive memory usage in DFA.

Extended Finite Automata (XFA)[3,4] addresses the state explosion problem by erasing most Kleene closures (such as ".*" and "[^\n]*") and repetitions (such as "a{20}") in the rule sets. XFA divides each rule into several sub-rules using Kleene closures as delimiters and builds one DFA for all sub-rules from all the original rules. A rule is matched if its corresponding sub-rules are accepted in the right order. Thus, for each sub-rule, it has its own history bit to record whether the sub-rule has been already matched. And, for the case of repetitions, XFA use one counter for each repetition to record the number of times that the repeated character appears. If the value of the counter is satisfied for the repetition, the case of the repetition is accepted.

The following examples explain the regular expression matching of XFA. Fig. 1 shows how XFA recognizes rule "ab.*cd". Consider text "abzcd" as the input string. When the second input character 'b' arrives, state 3 is reached and the bit associated to sub-rule "ab" is set. Then, when the input character 'd' is sent to XFA, final state 4 is reached on condition that the bit corresponding to sub-rule "ab" is set. Thus, the acceptance of rule "ab.*cd" is declared.

JFA[5] uses state variables to handle Kleene closures (such as ".*" and "[^\n]*") and address the problem of state explosion. JFA[5] is able to solve the overlapping character problem that XFA can solve. As motivated by the overlapping character problem and the forbidden character problem detailed below, we will proposed the a novel scheme to solve these two problem along with a memory efficient transition table compression scheme.

XFA[3,4] expands DFA with a small amount of additional memory to remember various types of information for dealing with the state explosion problem caused by Kleene closures. However, XFA still has some limitations that make the reduction of states inefficient.

Let p (prefix) and s (suffix) be the portion of a rule that precedes and follows a Kleene closure, respectively, as in pattern "p.*s". In XFA, when a prefix is matched, the corresponding bit is set to record the match of the prefix. When a suffix's final state is arrived, the acceptance is declared only if the bit associated to the corresponding prefix is set. However, this scheme does not support two cases: (i) the prefix and the suffix overlap, e.g. rule "ab.*bc", and (ii) forbidden characters of the string in Kleene closure also exist in the suffix, e.g., rules of the form "p[^c]*s", where p is a prefix, s is a suffix, and forbidden character 'c' appears in s, like "ab[^c]*cd". In order to understand the reason why XFA can't support the two cases, we consider XFA for rule "ab.*bc" as shown in Fig. 2(a). Input string "abc" will wrongly match rule "ab.*bc" because prefix "ab" and suffix "bc" overlap in character 'b'. For the case that forbidden characters appear in the suffix, see XFA for rule "ab[^c]*cd" as illustrated in Fig. 2(b). Input string "abcd" will erroneously miss the match if forbidden character 'c' resets the bit.