High Performance Pattern Matching Algorithm for Network Security

Yang Wang and Hidetsune Kobayashi

Graduate School of Science and Technology, Nihon University, Tokyo, Japan

Summary
Many network security applications rely on pattern matching to extract the threat from network traffic. The increase in network speed and traffic may make existing algorithms to become a performance bottleneck. Therefore, it is very necessary to develop faster and more efficient pattern matching algorithms in order to overcome the troubles on performance. In this paper, we present a new pattern matching algorithm. The improved algorithm and its working process are described in detail. Together with a new concept of reference point, a two-dimensional array redesigned based on novel generated rules in the pre-processing phase, endorse the algorithm a better performance and more efficient. The algorithm also passed tests and is validated. Our experimental results, the average performance of English text and network traffic can be improved up to 24% ~ 31% compared to Boyer-Moore algorithm.

Key words:
pattern matching, Boyer-Moore algorithm, network security, network intrusion detection

Introduction

Network security applications such as firewall, Network Intrusion Detection Systems (NIDS), virus scan software, anti-spam software, are endeavored to detect such attempts by monitoring incoming traffic for suspicious contents. They use a set of signatures (or rules) and report offending packets to the administrators for further actions. Since firewall and quality-of-service (Qos) applications examine multiple fields in the packet header, many firewall rules only have to check roughly 128 bits within the first 40 bytes of a packet header. On the other hand, signature-based NIDSs, such as Snort [8] and Bro [9], identify threat by testing network packets against rules that specify conditions for both the packet header and content. These applications often rely on pattern matching techniques.

While the pattern matching algorithms are applied to network security, such NIDSs, the speed of pattern matching usually becomes a bottleneck. Previous research results suggest that in the performance of NIDSs, 30% of total processing time is spent on pattern matching [1], especially in the cases like Web-intensive traffic, this percentage raises up to 80% [2]. The increase in network speed from Mbps to Gbps poses new challenges to existing network security applications designed for 100 Mbps. Along with the improvement of network technology presses forward, Gigabit and 10 Gigabit Ethernet is becoming a popular network environment. Most of the existing algorithms are not suitable for new generations of network security applications. In order to protect such environment, one of approach is to improve the performance of signatures detection engine by increasing the efficiency of pattern matching algorithm.

The rest of the paper is organized as follows. Section 2 is contributed to the characterization of the place of pattern matching in network security such as NIDS and discussion relevant prior works in pattern matching algorithm. We discuss our new algorithm in Section 3. Evaluation of the results of our techniques can be found in Section 4. Our contributions are summarized in Section 5.

2. Related Works

Pattern matching is one of the most important areas which have been studied in computer science [11]. In a standard formulation of the problem, we are given a pattern and a text and it is required to find all occurrences of the pattern in the text. If more than one search strings are matched against the input string simultaneously, it is called multiple pattern matching. Otherwise, it is called single pattern matching. Only single pattern matching algorithm will be referred in this paper [3].

Since the publication of the Boyer-Moore (BM) [5] and Knuth- Morris-Pratt (KMP) [6] algorithm, several hundreds of papers have been published dealing with exact pattern matching. The obtained algorithms include several non-trivial ones, notably the KMP left-to-right pattern matching algorithm and simplified variants of the BM right-to-left pattern matching algorithm [7].

The KMP algorithm, stated in terms of the Brute-Force algorithm, reduces the number of times it compares each character in the text to a character in the pattern. If properly implemented, the KMP algorithm only looks though each character in the text once.

Manuscript received September 5, 2006.
Manuscript revised October 25, 2006.

The BM algorithm is a rather famous pattern matching that is quite fast in practice. It is widely known as the best average-case performance of any known algorithm. The algorithm scans the characters of the pattern from right to left beginning with the rightmost one. In case of a mismatch it uses two pre-computed heuristics to shift the window to the right. These two heuristics are called bad-character-shift and good-suffix [7]. Both heuristic are triggered on a mismatch. The first heuristic works as follows: if the mismatching character appears in the search string, the search string is shifted so that the mismatching character is aligned with the rightmost position at which the mismatching character appears in the search string. If the mismatching character does not appear in the search string, the search string is shifted so that the first character of the pattern is one position past the mismatching character in the input. The second heuristic is also triggered on a mismatch. If the mismatch occurs in the middle of the search string, then there is a non-empty suffix that matches. The heuristic then shifts the search string to the next occurrence of the suffix in the string. This algorithm strives to completely ignore as many characters in the text as possible.

ExB [12] and E2xB [13] are pattern matching algorithms designed for providing quick negatives when the search pattern does not exist in the network traffic. The main notion of E2xB is the same as ExB. Once new network traffic arrives, the traffic is reprocessed to construct an occurrence bitmap to record the occurrence of distinct characters within the traffic. They identify the patterns individually to check if any characters appear in the pattern but not in the network traffic. If such characters do exist so the pattern is skipped since matching is impossible. Otherwise, the Boyer–Moore algorithm is called to search the pattern in the given traffic.

Boyer-Moore-Horspool [4] algorithm is one simplification of the BM algorithm. As we know, the bad-character shift used in the BM algorithm is not very efficient for small alphabets, but when the alphabet is large comparing with the length of the pattern, as it is often the case with the ASCII table and ordinary searches made under a text editor, it becomes very useful. Using it alone produces a very efficient algorithm in practice. Horspool proposed to use only the bad-character heuristic and the order in which the text character comparisons are performed is irrelevant. So the BMH algorithm is more efficient implementation. Some of today's security systems such as Snort, execute a fast string search on the associated pattern using the BM algorithm [10].

About string length of Snort pattern, M. Aldwairi et al [15] suggest that the average string length is 14 bytes and the majority of the strings are shorter than 26 bytes; it is also clear that there is a non-negligible number of strings longer the 40 bytes. In this paper, we propose a new method to improve the average performance of BM algorithm. The basic idea is to utilize two consecutive characters for the reference point instead of one character as in BM and BMH algorithm, because the case of two consecutive characters in a pattern is far less possible than one character in it when the pattern is relatively small. We will discuss it in detail.

3. New Pattern Matching Algorithm

A new pattern matching algorithm is presented here. Core part of the algorithm is described briefly as followings.

Pattern Matching Algorithm

```
begin
1  for (each char1 Σ) do
2      for(each char2  Σ) do next[char1,char2] ←m+2;
3  for (each char1  Σ) do next[char1,pattern[0]] ←m+1;
4  for (i=0 to m-2) do next[pattern[i],pattern[i+1]] ←m-i;
5  j←0;
6  while(j<=n) do begin
7      i←m-1;
8     while(i>=0 and pattern[i]=text[i+j]) do i←i-1;
9      if (i<0) then output(match at location j);
10     if (text[j+m-1,j+m]=pattern[m-2,m-1]) then j←j+1;
11          else j←j+next[text[j+m],text[j+m+1]];
    end while
end
```

Fig. 1 The core part of our algorithm

Where, Σ is the character set of *text* and *pattern*. *text* is the text string of length *n*. *pattern* is the pattern string of length *m*. *NEXT* is a two-dimensional array. *i* and *j* are pointers of the pattern string and text string respectively. Our pattern matching algorithm can be divided into two parts: preprocessing phase and search phase. Step 1 to step 4 above is called as preprocessing phase. Step 5 to step 11 is called as searching phase. The task of preprocessing phase is to generate a two-dimensional array *NEXT*. Array

NEXT in our algorithm is very important, which will decide how to move to a proper position for next search. One good design of array *NEXT* will bring better performance for whole pattern matching algorithm. The position of *No.m+1* and *No.m+2* characters in text string is assigned as the reference point. Array *NEXT* can be generated according to the following rules:

$$NEXT[char1, char2] = \begin{cases} m+1 & \text{if char2} = \text{pattern}[0] \\ m-i & \text{if char1} = \text{pattern}[\text{m}-\text{s}] \text{ and} \\ & \quad \text{char2} = \text{pattern}[\text{m}-\text{s}+1] \\ m+2 & \text{otherwise} \end{cases}$$

Fig.2 The rules of *NEXT* array

After array *NEXT* is generated, the values of array *NEXT* will be invariable during searching process. Above rules are very clear and simple, so algorithm generated array *NEXT* is very simple too.

During the searching phase, the comparison is performed from right to left and when the position on the text factor *text[j .. j+m-1]* and a mismatch occurs between *pattern[i]* and *text[i+j]* while *pattern[i..m −1]= text[i+j..j+m-1]* the *NEXT* is performed for text characters *text[j+m]* and *text[j+m+1]*.

When the pattern is relatively small, the case of two consecutive characters in a pattern is far less possible than one character in it, so the shift value can be much greater than that of in BM algorithm. For example, shift value (*NEXT* array) of pattern *gcag* in BM algorithm in Table 1, and in our algorithm in Table 2.

Table 1: Shift value in BM algorithm

char	*a*	*c*	*g*
Shift value	1	2	3

Table 2: Shift value our algorithm

char1 / char2	*a*	*c*	*g*
a	6	6	2
c	3	6	6
g	4	5	5

The example of improved array of new algorithm is shown in Table 2. At each check point, if a mismatching occurs, then the two consecutive characters next to the last character of current comparing window will be used to execute next matching.

Fig.3 is an example of pattern matching for the new algorithm. In preprocessing phase, a shift table will be generated as Table 2. In the first attempt, mismatching character in substring is found during search phase. Then character '*a*' and '*c*' are treated as a reference point. According to the Table 2, the shift value of characters '*ac*' is 6, which means the next matching will happen at right shift 6 characters. We can see in the next attempt, the matching will be finished successfully.

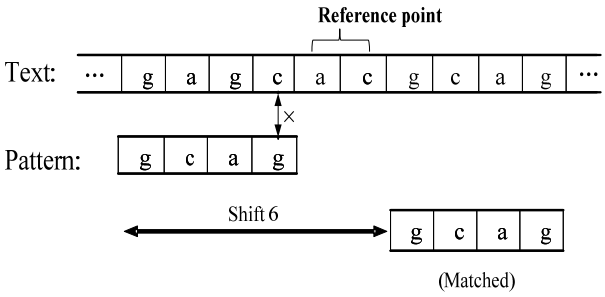


Fig.3 The example for the new algorithm

4. Performance Test and Evaluation

We evaluated the performance algorithm using various text strings and pattern strings. The results are presented as followings.

4.1 Experiments in Worst Case

Text string *gcatcgcagagagtatacagtacg* and *gcagagag*, a pair of well-known and commonly used samples for comparing and evaluating different pattern matching algorithms, are adopted in the performance test [7]. Since they great relativity between above text and pattern samples, the test results may be have the worst. We tested the attempt numbers of BM and our algorithm using the strings *g*, *gc*, *gca*, *gcag*, *gcaga*, *gcagag*, *gcagaga*, *gcagagag*. Corresponding character numbers of these pattern strings are from 1 to 8 respectively. Fig.4 shows that in some cases, the attempt numbers of the new algorithm are greater than that of BM algorithm, although they are 10% less in a general average. In such case, the new algorithm can not show its advantage.

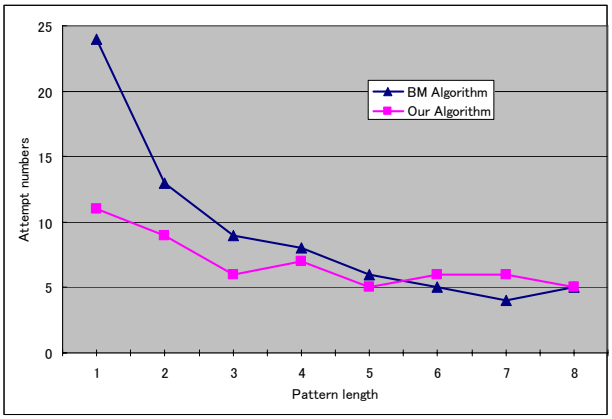


Fig.4 Attempt numbers for worst case

4.2 Experiments in English Text

We ran experiments on an English text *"Snort is an open source network intrusion prevention and detection system utilizing a rule-driven language, which combines the benefits of signature, protocol and anomaly based inspection methods."*. We searched for short to long patterns in this text and compared the attempt numbers of BM and our algorithm. Test results are shown in Fig.5. The attempt numbers of our algorithm are decreased by 13%～50%(31% as average) comparing to those of BM algorithm in this case.

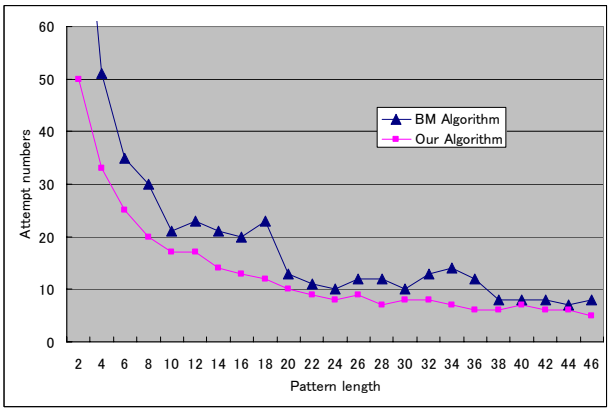


Fig.5 Attempt numbers for English text

4.3 Experiments in Network Traffic

The network traffic from the 2000 DARPA Intrusion Detection Data Sets [14], the inside Tcpdump file of Scenario Two which has a size of 66,110,361 byte, are selected. We randomly selected the pattern strings length arranged from 1 to 122 from Snort rules distributed in Aug. 9, 2006. The results are shown in Fig.6.

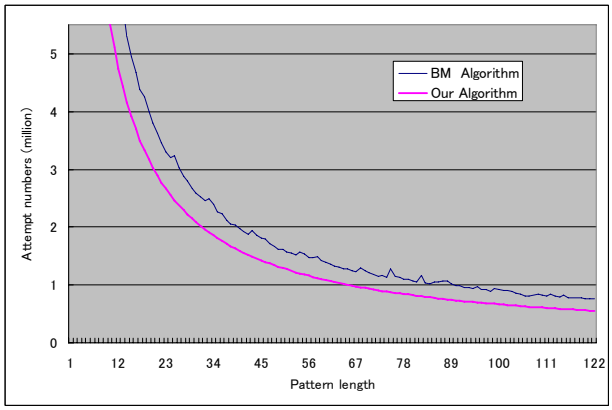


Fig.6 Attempt numbers for network traffic

The improvement in real traffic is shown in Fig.7, comparing to BM algorithm we can see a 24% average improvement of our algorithm.

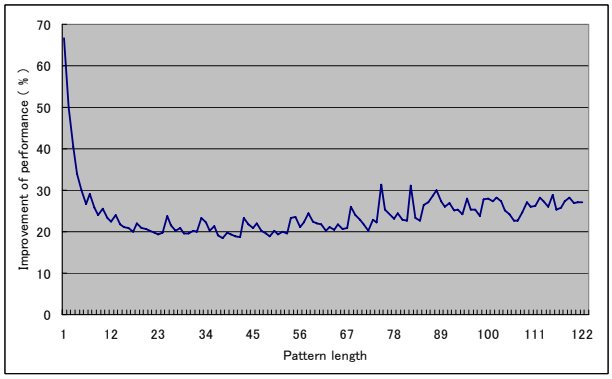


Fig.7 Improvement of our algorithm

5. Conclusions

We presented a novel pattern matching algorithm and evaluated its performance by using diverse text strings and various set of pattern strings. The testing results of English text and network traffic show an improvement of 24%～31% in average comparing to the case of BM algorithm . The new algorithm is a variant of BM algorithm and has been greatly improved. A two-dimensional array in the preprocessing phase is redesigned. The concept of reference point, makes the algorithm to have better performance and more efficient. It provides another option for network security applications, not only for Intrusion

Detection System, but also for other security applications such as virus scanning, firewalls, and layer seven switches.

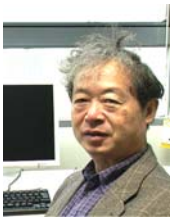
Acknowledgments

The authors thank the anonymous reviewers for providing constructive comments and suggestions on this paper.

References

[1] M. Fisk and G. Varghese: An analysis of fast string matching applied to content-based forwarding and intrusion detection. Technical Report CS2001-0670 (updated version), 2002.
[2] S. Antonatos, K. G. Anagnostakis, and E. P. Markatos: Generating realistic workloads for network intrusion detection systems. In ACM Workloads on software and Performance, 2004.
[3] M. Crochemore, C. Hancart: Pattern Matching in Algorithms and Theory of Computation Handbook. CRC Press Inc., Bocaaton, FL. 1999.
[4] R. N. Horspool: Practical fast searching in strings. Software – Practice & Experience, 10(6):501-506, 1980.
[5] R. S. Boyer and J. S. Moore: A fast string searching algorithm. Communications of the ACM, vol. 20, no. 10, pp.762-772, 1977.
[6] Beate Commentz, Walter: A string matching algorithm fast on the average. Proc. 6th International Colloquium on Automata, Languages and Programming, vol. 71 of Lecture Notes in Computer Science, pp. 118-132, 1979.
[7] C.Charras, T.Lecroq: Exact String Matching Algrothms. http://www-igm.univ-mlv.fr/~lecroq/string.
[8] Snort, http://www.snort.org
[9] V. Paxson: Bro: A System for Detecting Network Intruders in Real-Time. Proc. of the 7th USENIX Security Symposium, 1998.
[10] Stephen Gossin, et al: Pattern Matching in Snort. http://www.sporksoft.com/~njones/notes/CSE202/project.pdf , 2002.
[11] W. F. Smyth: Computing Pattern in Strings. Pearson Addison Wesley, 2003.
[12] E.P. Markatos, S. Antonatos, M. Polychronakis, and K.G. Anagnostakis: ExB: Exclusion Based signature Matching for Intrusion Detection. In Proc. of Communications and Computer Networks, MIT, USA, 2002.
[13] K. G. Anagnostakis, S. Antonatos, E. P. Markatos, M. Polychronakis: E^2XB:A domain-specific string matching algorithm for intrusion detection. Proc. of the 18th IFIP International Information Security, 2003.
[14] Stolfo SJ, Fan W, Lee W, Prodromidis A, Chan PK: Cost-Based modeling for fraud and intrusion detection: Results from the JAM project. In: Proc. of the 2000 DARPA Information Survivability Conf. and Exposition. 2000.
[15] Monther Aldwairi, Thomas Conte, Paul D. Franzon: Configurable string matching hardware for speeding up intrusion detection. SIGARCH Computer Architecture News 33(1): 99-107. 2005.

Yang Wang received the M.Sc. degree in Computer Science from Graduate School of Science and Technology, Nihon University in 2004. He has been received the Ph.D. course in Computer Science at the Graduate School of Science and Technology, Nihon University, Tokyo, Japan from 2004. His main research interests are in the areas of intrusion detection, network security, and wireless sensor network.



Hidetsune Kobayashi received the M.Sc. degree in Mathematics from Kyoto University. He also received Doctor from Nihon University. He is a professor of Nihon University. His research interest is computer mathematics.