Robust Quick String Matching Algorithm for Network Security

Jianming Yu,[1,2] and Yibo Xue,[2,3]

1 Department of Automation, Tsinghua University, Beijing, China
2 Research Institute of Information Technology, Tsinghua University, Beijing, China
3 Tsinghua National Lab for Information Science and Technology, Beijing, China

Summary
String matching is one of the key algorithms in network security, and many areas could be benefit from a faster string matching algorithm. Based on the most efficient string matching algorithm in usual applications, the Boyer-Moore (BM) algorithm, a novel algorithm called RQS is proposed. RQS utilizes an improved bad character heuristic to achieve bigger shift value area and an enhanced good suffix heuristic to dramatically improve the worst case performance. The two heuristics combined with a novel determinant condition to switch between them enable RQS achieve a higher performance than BM both under normal and worst case situation. The experimental results reveal that RQS appears efficient than BM many times in worst case, and the longer the pattern, the bigger the performance improvement. The performance of RQS is 7.57~36.34% higher than BM in English text searching, 16.26~26.18% higher than BM in uniformly random text searching, and 9.77% higher than BM in the real world Snort pattern set searching.

Key words:
String matching; network security; algorithmic performance attack

Introduction

String matching is one of the basic and important research subjects in computer science. String matching consists in finding one, or more generally, all the occurrences of a search string, also be called as pattern, in an input string. 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. Single pattern matching algorithm will be referred only in this paper.

Single pattern matching algorithm is widely used in network security environments. (In network security realm, pattern is a string indicating network intrusion, attack, virus, Spam or dirty network information et al). For example, in Snort, the famous open sources lightweight NIDS [1, 2], the Boyer-Moore (BM) [3] algorithm is called while the number of patterns needed to be matched is less than five. Single pattern matching algorithm is also the basis to construct exclusion-based pattern matching algorithm and hybrid pattern matching engine to handle enormous network security detection patterns.

The exclusion-based string matching algorithm utilizes heuristics to identify the patterns that could not occur in the input string first, and then use single pattern matching algorithm to match the patterns could not be excluded. The ExB [4] and E^2xB [5] are typical exclusion-based algorithms. The hybrid pattern matching engine triggers different algorithms, generally combines single pattern matching and multiple pattern matching algorithms, depending on different application environments such as the number of patterns and the size of input string. Considering the fact that no single algorithm performs best in all cases, a hybrid pattern matching engine appears to be the best approach in network security applications [6, 7].

Along with the development of network attack technologies, the network security equipment itself becomes the attack objective. So does the string matching algorithm. An effective attack method to string matching algorithm is “algorithmic performance attack”: an attacker intentionally provides inputs that will overload or cause the worst case performance of an algorithm [6]. It could slow down the search and cause dropped packets, upon which the intruder could begin his attack. For example, the processing time of Snort can be raised by up to 25 times under such attack [8]. Improving the average-case and worst-case performance of string matching algorithm simultaneously would effectively defend from the algorithmic performance attack.

The BM algorithm is considered as the most efficient string-matching algorithm in usual applications, and is widely regarded as providing the best average-case performance of any known algorithm [6, 13]. In this work we concentrate on improving the average and worst case performance of BM at the same time. A novel algorithm named as Robust Quick String Matching (RQS) is proposed.

Manuscript received July 5, 2006.
Manuscript revised July 25 , 2006.

The rest of the paper is organized as follows. Section 2 gives the description of the BM algorithm. Section 3 is a formal description of RQS. In section 4, detailed evaluation and analysis of RQS is presented. We summarize our contributions in section 5, where the open issues for future investigation is outlined as well.From this section, input the body of your manuscript according to the constitution that you had. For detailed information for authors, please refer to [1].

2. Related Works: The BM Algorithm

String matching deals with a string $T=t_1t_2\ldots t_n$ (the input string) of size n and searches it for all occurrences of another, shorter string $P=p_1p_2\ldots p_m$ (the search string) of size m ($n > m$). Both strings build over a finite set of character called an alphabet denoted by Σ.

String matching algorithms scan the T with the help of the sliding window mechanism. The size of the window is generally equal to m. To a check point j $(1\leq j\leq n)$, the window is positioned on $t_jt_{j+1}\ldots t_{j+m-1}$. At each check point, the characters of the window are compared with the characters of the search string. After a whole match or after a mismatch, the window is shifted along the T according to the heuristics of each algorithm.

BM utilizes two heuristics, bad character and good suffix, to reduce the number of comparisons (compared to Brute Force string matching). The search is carried out by shifting the window from left to right. t_1 is the first check point. Within each window, the characters of the window and the P are compared from right to left. Both heuristics are triggered on a mismatch. Suppose the matching check is taken between $t_{i+1}t_{i+2}\ldots t_{i+m}$, $0\leq i\leq n-m$ and $p_1p_2\ldots p_m$ now, and $p_{j+1}p_{j+2}\ldots p_m=t_{i+j+1}t_{i+j+2}\ldots t_{i+m}$, $1 \leq j \leq m-1$, the mismatching occurs at p_j, $p_j \neq t_{i+j}$.

The bad character heuristic works this way: when a mismatch appears, the window is shifted to right so that the mismatching character in T, t_{i+j}, is aligned with p_{k1}, $k1=max\{k2|p_{k2}=t_{i+j},\ 1\leq k2<j\}$. If t_{i+j} dose not appears in $p_1p_2\ldots p_{j-1}$, the window is shifted to the position that p_1 is one position past t_{i+j}.

The good suffix heuristic, works another way: when a mismatching occurs, there is a non-empty suffix that matches. The window is shifted to the next occurrence of the suffix in $p_1p_2\ldots p_{m-1}$. If the suffix does not appear, the window is shifted to the position that p_{k3} is aligned with t_{i+m}, $k3=max\{k4|\ p_1p_2\ldots p_{k4}=t_{i+m-k4+1}\ldots t_{i+m}\}$, $1\leq k4\leq m-j-1$. If k_3 does not exist, the window is shifted to the position that p_1 is one position past t_{i+m}.

BM takes the far most shift caused by the two heuristics.

3. RQS: Robust Quick String Matching

The RQS algorithm utilizes an improved bad character heuristic and an enhanced good suffix heuristic to improve the average and worst case performance of BM simultaneously.

The improved bad character heuristic of the RQS algorithm is shown in Figure 1. The character next to the rightmost character of the current window is always used as the bad character. For example in Figure 1, at current check point marked by the broken line window, use character 's' of T as the "bad character" to calculate the shift value. Compared with the bad character heuristic of BM, this will provide a larger shift value area and easy to implement [14].

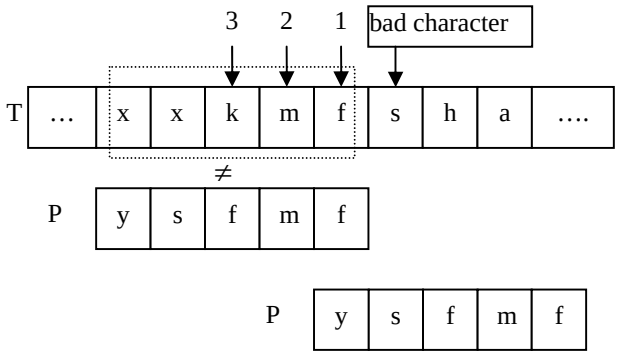

Fig 1. The improved bad character heuristic of RQS algorithm

The main drawback of BM is that after a shift it forgets the information about characters already matched. So in worst case situation, its behavior is same as Brute Force. We proposed a novel enhanced good suffix heuristic which could remember the characters already matched when needed.

The enhanced good suffix heuristic is: while the information of the characters already matched is needed to remember at a specific check point, only good suffix heuristic is used at that check point. Compared to the policy of BM (at every check point always calculate good suffix and bad character shift value and select the bigger one), we can easily remember the characters have already matched by a variable and do not compare them again at next check point. As shown in Figure 2, if the shift value, *shift*, is calculated by good suffix heuristic at current check point, so $p_0p_1\ldots p_{m-shift-1}=p_{shift}p_{shift+1}\ldots p_{m-1}$. We use variable $offset=m-shift$ to remember the starting position at next check point, then at next check point only need to compare $p_{offset}\ldots p_{m-1}$.

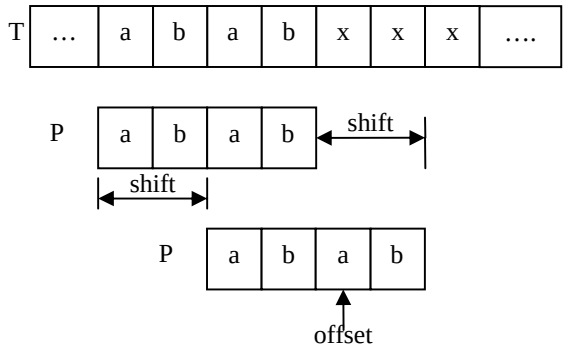

Fig 2. The enhanced good suffix heuristic of RQS

To keep the simplicity of implementation and achieve bigger shift value, a simple determinant condition is proposed to determine if the enhanced good suffix heuristic need to be called to remember the information of the characters already matched.

Under the worst case situation, the behavior of BM is same as the Brute Force. This means the shift value is always equal to one. So in the design of RQS, if the shift value calculated by the improved bad character heuristic is equal to one at current check point, the enhanced good suffix heuristic is utilized. This has two benefits. First, we could get a bigger shift value because the shift value calculated by good suffix heuristic is no less than one. Second, we could decrease the characters need to check at next check point because the enhanced good suffix heuristic remembered the characters already matched.

The search procedure of the proposed RQS algorithm is shown in Figure 3. Two tables are used to store the shift value calculated by the two heuristics: *RQSBc[]* for bad character and *RQSGs[]* for good suffix. Where *P* is the search string, *m* is its length. *T* is the input string, *n* is its length.

```
If (n<m) do exit;
j = 0; // j represents the check point
offset = 0;
while (j <= n - m) do begin
      i = m-1;
      if (RQSBc[T[j + m]]  > 1) do // using RQS bad character heuristic
            while(P[i] = T[j+i] and i>=0) do begin
                  i = i-1;
            end while
            j = j+RQSBc [T[j + m]];
      else   // using RQS good suffix heuristic
            while (P[i] = T[i + j] && i >= offset) do begin
                  i = i-1;
            end while
            if (i < offset) do    // a match found
                  j = j+RQSGs[0];
                  offset = m-shift; //remember the character already matched
            else do
                  j = j+RQSGs[i];
                  offset = 0;
end while
```

Fig 3. The search procedure of RQS

4. Experiments and Analysis

We implemented the RQS algorithm and compared the performance of it with the BM algorithm. The codes of BM in [13] are used, and it is a high quality implementation.

4.1 Experimental Environment

In the experiments we used a PC with Intel Pentium® 4 2.4 GHz processor, with L1 cache of 8KB and L2 cache of 512KB, and 512MB of RAM. The host operating system is Microsoft® Windows XP professional. The compiler used is Microsoft® Visual C++ 6.0.

In all the experiments the length of the input string *T* is 16KB. We performed experiments with three kinds of input string *T*. The first was made up of the same character 'A' to test worst case performance. The second is a piece of English text obtained from a software manual. And the third consists of uniformly random characters from 256-character alphabet.

4.2 The Worst Case Test

To the input string and search string both be made up of the same character, the shift value is always equal to one and the match number is maximum. It provides the worst case scenario to string matching algorithm. We found that there are such keywords used in practice, for example the blow detection rule of Snort.

alert tcp any any -> any any (ack:0; flags:SFU12; content:" AAAAAAAAAAAAAAAA" ; depth: 16;).

It tries to find the keyword " AAAAAAAAAAAAAAAA" in all the tcp flow.

T=AA…A and *P=AA…A* was used to test the performance of RQS and BM. The results are shown in Figure 4. The horizontal axis represents the pattern length, and the vertical axis represents processing time.

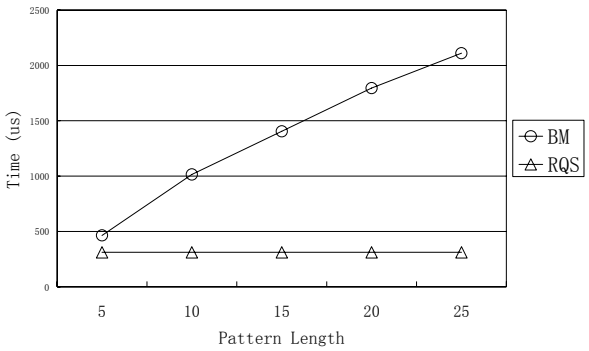

Fig 4. The performance with P=AA…A and T=AA…A

We could find that the performance of RQS is stable and higher than BM for many times under this situation. Because the information of the characters already matched is remembered, the number of character comparison is dramatically reduced. In fact, because the shift value is always equal to one under this situation, the enhanced good suffix heuristic is called at every check point. So the number of characters compared is equal to the length of input string and regardless of the length of the keyword.

4.3 Experiments with an English Text

The text is obtained from a software manual. The patterns are randomly selected from the text. For every pattern length, we selected 250 different patterns, and the 250 patterns are compared with the text one by one.

The results are shown in Figure 5. The horizontal axis represents the pattern length, and the vertical axis represents processing time. The processing time is the average searching time of 250 patterns.

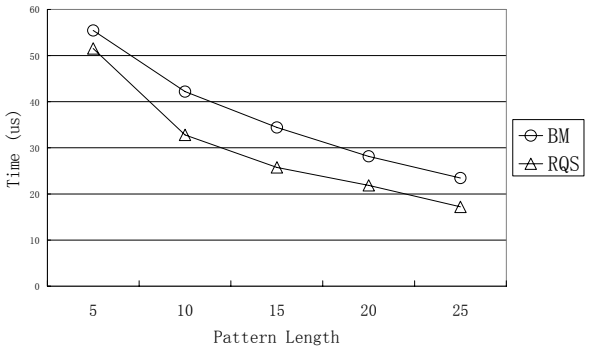

Fig 5. The performance with an English text searching

We can find that the performance of RQS is 7.57~36.34% higher than BM in the English text searching.

4.4 Experiments with Uniformly Random Text

One text which consists of uniformly random characters from 256-character alphabet is generated. The patterns are randomly selected from the text. For every pattern length, we selected 250 different patterns, and the 250 patterns are compared with the text one by one.

The results are shown in Figure 6. The horizontal axis represents the pattern length, and the vertical axis represents processing time. The processing time is the average searching time of 250 patterns.

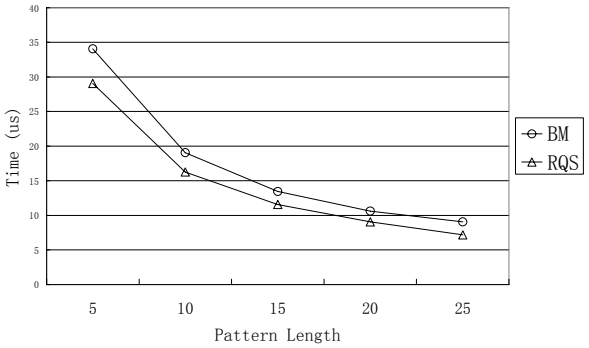

Fig. 6. The performance with uniformly random text

We can find that the performance of RQS is 16.26~26.18% higher than BM in the uniformly random text searching.

4.5 Experiments with Snort Pattern Set

The real world pattern set of Snort rules is extracted. The latest Snort rules distributed in Jun. 15, 2006 are used. There are totally 2410 different patterns. The pattern length arranges from 1 to 122. The text used is uniformly random text with 256-character alphabet. The 2410 patterns are compared with the text one by one.

The results are shown in Figure 7. The horizontal axis represents different algorithm, and the vertical axis represents processing time. The processing time in Figure 6 is the total searching time of the 2410 patterns.

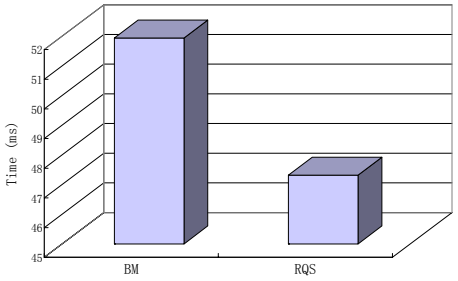

Fig. 7. The performance with Snort pattern set

We can find that the performance of RQS is 9.77% higher than BM in the real world Snort pattern set searching.

5. Conclusion and Future Work

We have examined the problem of string matching for network security, especially the anti-algorithmic performance attack problem, and presented the design of an improved BM algorithm called RQS. RQS utilizes an improved bad character heuristic to achieve bigger shift value area and an enhanced good suffix heuristic to remember the information of characters already matched when needed. A novel determinant condition to determine if the enhanced good suffix heuristic need to be called is proposed. So the worst case performance of RQS is dramatically improved compared with BM and the normal performance is also improved at the same time..

We have evaluated RQS against BM using different texts and patterns. The experimental results reveal that RQS appears efficient than BM many times in worst case, and the longer the pattern, the bigger the performance improvement. RQS is also efficient than BM in normal situation. The performance of RQS is 7.57~36.34% higher than BM in the English text searching, 16.26~26.18% higher than BM in the uniformly random text searching, and 9.77% higher than BM in the real world Snort pattern set searching.

Future work would include designing exclusion-based algorithm and hybrid pattern matching engine for network security applications based on RQS. The design idea of this paper, improving and balancing the normal and worst case performance of string matching algorithms for network security, will be applied to multiple pattern matching algorithms.

Acknowledgments

We would like to thank the anonymous reviewers for providing useful comments on this paper. We would thank our colleagues at the Network Security Laboratory, Research Institute of Information Technology, Tsinghua University for their comments and enlightening discussions on draft of the paper. And this work is supported by Intel IXA University Program.

References

[1] M. Norton, and D. Roelker, “The new Snort”, Computer security journal, vol. 19, no. 3, pp. 37-47, 2003.
[2] M. Roesch, “Snort: lightweight intrusion detection for networks”, Proc. 13th System Administration Conference and Exhibition (LISA’1999), pp. 229-238, 1999.
[3] R. Boyer, and J. Moore, “A fast string searching algorithm”, Communications of the ACM, vol. 20, no. 10, pp. 762-772, 1977.
[4] E. P. Markatos, S. Antonatos, M. Polychronakis, and K. G. Anagnostakis, “EXB: Exclusion-based signature matching for intrusion detection”, Proc. The CCN’02, 2002.
[5] K. G. Anagnostakis, E. P. Markatos, S. Antonatos, and M. Polychronakis, “E2XB: A domain-specific string matching algorithm for intrusion detection”, Proc. 18th IFIP International Information Security Conference (SEC2003), 2003.
[6] M. Fisk, and G. Varghese, “Fast content-based packet handling for intrusion detection”, UCSD Technical Report CS2001-0670, May 2001.
[7] S. Antonatos, K. G. Anagnostakis, E. P. Markatos, and M. Polychronakis, “Performance analysis of content matching intrusion detection system”, Proc. 2004 International Symposium on Applications and the Internet (SAINT’04), 2004.
[8] S. Antonatos, K. G. Anagnostakis, and E. P. Markatos, “Generating realistic workloads for network intrusion detection systems”, Software engineering notes, vol. 29, no. 1, pp. 207-215, 2004.
[9] R. N. Horspool, “Practical fast searching in strings”, Software practice and experience, vol. 10, no. 6, pp. 501-506, 1980.
[10] R. M. Karp, and M. O. Rabin, “Efficient randomized pattern-matching algorithms”, IBM J. Res. Dev., vol. 31, no. 2, pp. 249-260, 1987.
[11] D. Knuth, J. Morris, and V. Pratt, “Fast pattern matching in strings”, SIAM journal on computing, vol. 6, no. 2, pp. 323-350, 1977.
[12] M. Crochemore, M. C. Hancart, Pattern Matching in Algorithms and Theory of Computation Handbook. CRC Press Inc., Boca aton, FL, 1999.
[13] C. Charras, and T. Lecroq, “Exact string matching algorithms”, http://www-igm.univ-mlv.fr/~lecroq/string/, 1997.
[14] D. M. Sunday, “A very fast substring search algorithm”, Communications of the ACM, vol. 33, no. 8, pp. 132-142, 1990.

Jianming Yu received the B.S. degree in Electrical Power System and Its Automation from Zhengzhou University of Technology in 1999 and the M.S. degree in Control Theory and Control Engineering from Harbin Institute of Technology in 2003. From 2003 to now, he stayed in Network Security Laboratory, Research Institute of Information Technology, Tsinghua University as a PhD candidate to study network intrusion detection, string matching for network security, and network equipments design based on network processor.