Fast Multi-Pattern Matching Algorithm on Compressed Network Traffic

Hao Peng, Jianxin Li, Bo Li, M. Hassan Arif

School of Computer Science and Engineering, Beihang University, Beijing, 100191, China

Abstracts: Pattern matching is a fundamental approach to detect malicious behaviors and information over Internet, which has been gradually used in high-speed network traffic analysis. However, there is a performance bottleneck for multi-pattern matching on online compressed network traffic(CNT), this is because malicious and intrusion codes are often embedded into compressed network traffic. In this paper, we propose an online fast and multi-pattern matching algorithm on compressed network traffic (FMMCN). FMMCN employs two types of jumping, i.e. jumping during sliding window and a string jump scanning strategy to skip unnecessary compressed bytes. Moreover, FMMCN has the ability to efficiently process multiple large volume of networks such as HTTP traffic, vehicles traffic, and other Internet-based services. The experimental results show that FMMCN can ignore more than 89.5% of bytes, and its maximum speed reaches 176.470MB/s in a midrange switches device, which is faster than the current fastest algorithm ACCH by almost 73.15 MB/s.

Keywords: compressed network traffic; network security; multiple pattern matching; skip scanning; depth of boundary.

I. INTRODUCTION

Security of compressed network traffic is a very serious problem. The typical case includes stolen financial center data and runaway high-speed vehicles, etc. Nowadays, compression method has been extensively used in many popular Web service and APP, such as Google+, social networking services APP, Amap navigation, Motor Vehicle Auto Driving System and Big data storage systems. For example, the vehicles themselves form a temporary network, and the participating vehicles also serve as a wireless router [1]. No matter the vehicle at a high speed or stop state, automotive automation control systems are likely to be attacked by hackers over the wireless network and remote control. This makes Internet of Vehicles (IOV) a prominent vertical for the emerging Internet of Things (IOT) market [2]. With the rise of the Internet+ economy, it's the identical threat for HTTP compression services. However the core technology, for preventing network traffic being attacked, is a secure and reliable network communication protocol and a security filtering system on Network Intrusion Detection System (NIDS) or Web Application Firewall (WAF). Generally, multi-pattern matching and regular expression matching are key technologies of deep packet inspection in networks analysis. As far as we know, real compression data includes plaintext, compressed and encrypted data. In this paper, we focus on a faster and safer multi-pattern matching on compressed networks including HTTP traffic and vehicular traffic.

Multi-pattern matching on compressed traffic is a basic NIDS/WAF technique on HTTP and vehicular networks to detect malicious behaviors in cyberspace. The performance of online network security tools are dominated by the speed of multi-pattern matching algorithms on compressed traffic that detects malicious activities. Nowadays traffic compression uses gzip compression technology[3] and is extensively used in many popular vehicular network devices, Internet+applications and distributed storage data system. In actual compressed traffic packet, average compression rate of gzip reaches 75% in the network transmission.

The average compression rate for multi-coding network services will be slightly lower and it also varies with different character sets. However, Multi-pattern matching on CNT faces three time-consuming stages, including network packets capturing, packet decompression and multi-pattern matching. In an actual online network environment, the size of Ethernet packets range from 64 bytes to 1518 bytes. Packet header stores some information about whether the current packet uses gzip compression and the serial number to reassemble, but it is useless for the content matching. Gzip compression consists of LZ77 [4] and Huffman, which are unsupported by compressed pattern matching. LZ77 decompression consumes more time and space than Huffman decoding. After Huffman and LZ77 decompression, the time consumption of excellent multi-pattern matching algorithms are linear with the length of characters, such as Multiple Shift-And [5], Ad-AC [6], Wu-Manber [7] and, SBOM [8]. The Wu-Manber and SBOM have more excellent matching performance than AC in the field of multiple byte matching. A faster online multi-pattern matching method on compressed HTTP traffic will improve performance and bandwidth for vehicular network communication, such as 3G、4G、Wi-Fi telematics communications as in reference [9].

In this paper, we present a novel fast multi-pattern matching algorithm on compressed networks (FMMCN). The algorithm is based on the facts that parts of scanned bytes don't need to be scanned again, so we designed a strategy to accelerate matching on the LZ77 dataset. We reference the principle of the Wu-Manber algorithm, and design FMMCN to adapt to LZ77 mixed datasets (*literal*, *distance* and *length*) for security filtering in vehicular network. Our key insight lies in only scanning a limited depth of boundary bytes replaced by LZ77 pointer.

In FMMCN, we design astack structure that stores every hit pattern's information for every current session. We can reproduce location of hits and corresponding patterns replaced by LZ77 pointer, with only a lightweight computation over the corresponding stack. In online vehicular network traffic, up to 89.5% of bytes can be ignored by skipping inner data that is replaced by an LZ77 pointer. The average speed of FMMCN algorithm reaches 176.470MB/S, with thousands of Snort signatures and random malicious datasets, in Mid-Range switches devices.It is remarkably faster than Wu-Manber and Ad-AC matching on decompression multi-byte data and even faster than ACCH by about 70MB/S. The FMMCN algorithm uses the backtracking of LZ77 pointers and records stack structures for multi-pattern matching on compressed vehicular networks.

An online fast and multi-pattern matching algorithm on compressed network traffic is proposed in this paper.

II. Background & Related Work

The problem of multi-pattern matching on compressed traffic has received many researchers' attention. According to Anat Bremler-Barr [10] [11] [12], ACCH can gain up 70% improvement in the performance of multi-pattern matching on compressed HTTP traffic. The papers [13] [14] [15] [16] show that compressing a file once and then performing pattern matching on the compressed file, accelerate the scanning process. Wu-Manber and SBOM moving faster than Ad-AC on multiple byte plain text. But, there is no one put forward an accelerated multi-patterns matching algorithm on CNT.

2.1 Gzip

Gzip is a typical compression algorithm for application layer communication, which is composed of LZ77 and Huffman. Gzip compression applies LZ77 once to reduce information redundancy and three times Huffman mapping to reduce coding length. The process includes three kinds of compression techniques. Firstly, the sequence of source symbols is compressed by the LZ77. The output consists of *literal*, *distance* and *length* pointer, which will then be compressed by two types of Huffman mapping. Then, the output of the second compression stage is two Huffman dictionaries and two types of Huffman coding bits (*LIT* and *DIST*). Finally, Run Length encoding and Huffman coding are used to compress two Huffman dictionaries.

LZ77 has to search repetitive bytes in front of historical characters, so it introduces sliding window and minimum repeat length technology. Sliding window stores the characters that are just scanned, in order to take advantage of reproducibility and locality in language expression. Generally, the size of a sliding window is 32 KB. When the length of repeated bytes is more than minimum i.e. 3 bytes, then repeated bytes can be replaced by a short pair (*distance*, *length*).The '*distance*'is the distance of pointer from referred string, and it is between 1 and 32768(32KB), and length is a number between 3 and 258, which shows the length of the repeated byte. For example, the text: "apple fapplt", will be compressed to: "applef (6, 4) t" by LZ77.

Huffman mapping encodes *literal*, *length* and *distance* bytes, working on a byte by byte basis and transforms every byte to a corresponding codeword. Huffman mapping uses the coding of the most unbalanced tree which ensures that no codeword is a prefix for other codeword. Huffman Dictionaries store a one-to-one mapping relationship between input byte and a corresponding codeword. Compression will generate two Huffman Dictionaries and two types of coding bytes.

2.2 Multi-pattern matching

In the multi-pattern matching field, there are four excellent performance automaton algorithms, Multiple Shift-And [4], Advanced Aho-Corasick, Wu-Manber and SBOM. Multiple Shift-And algorithm uses the architecture of the bit-parallelism method. It is well suited in situations where patterns can be stored in physical words. But actually, the total number of patterns is more than thousands and includes multi-byte patterns. Advanced Aho-Corasick is the best algorithm in prefix search methods and suitable for parts of character sets, in which the length of shortest patterns is less than 8 bytes. But Advanced Aho-Corasick has high memory consumption. In multi-pattern matching, the suffix based approach is faster than the prefix based one. According to former research [17], every algorithm works well in some suitable computing conditions, as shown in figure 1.

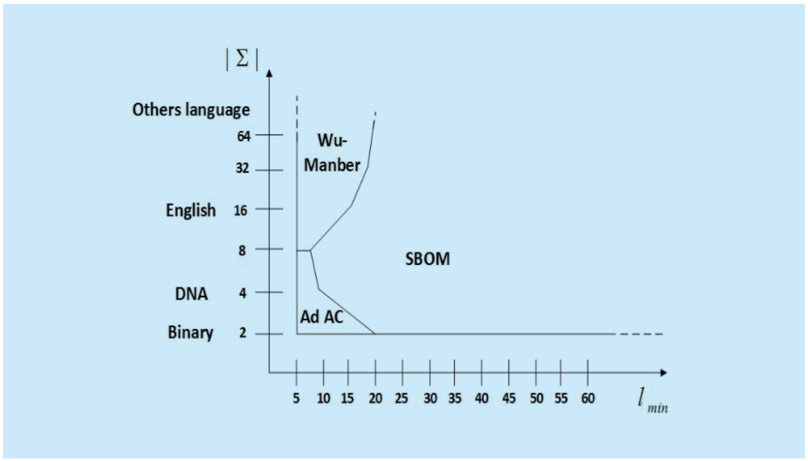

Fig.1 *Map of the most efficient algorithms when searching for 1000 strings*

III. THE CHALLENGES IN MULTI-PATTERN MATCHING METHODS

In this section, we will give an overview of the challenges in multi-pattern matching in compressed network traffic. Online pattern matching will encounter more difficulties than offline matching. The online filter should not affect the speed of network access for normal service. The reliability of online multi-pattern matching algorithm not only depends on packet capturing and matching, but also on time and space consumption. Importantly, Huffman Dictionaries are stored in the first packet. In general, different network sessions have different Huffman Dictionaries.

Published excellent algorithms are based on accelerating Aho-Corasick for multi-pattern matching on compressed HTTP (ACCH) .ACCH performs LZ77 decompressing and pattern matching at the same time, and adds status information for every byte to compute the best pointers depth. It only scans a limited depth of boundary characters in LZ77 pointer‘s referred string. The main challenges in multi-pattern matching are:

3.1 Space

For each traffic session, the time and space consumption in Huffman decoding is very little, while it‘s inevitable that 32KB sliding window are costly for LZ77 backtracking straight in each session. In an actual network environment, the size for most of the Ethernet packets is 1.5KB, after removing the header. Notably, the space complexity of AC algorithm is $O(|P| \times |\sum^{|}|)$. In 1000 malicious snort patterns, space consumption of AC’s DFA [18] [19] [20] [21] requires about 10.6MB to store 10174 states. There will be large memory consumption for high concurrent sessions.

3.2 Time

The speed of existing pattern matching algorithms is not up to mark in networks. In the search phase, the time complexity of ACCH algorithm is determined by the length of the input string and the depth of the pointer’s back tracking. The size of the backtracking length should not exceed the total size of text characters that are scanned. For mixed datasets (*literal*, *length* and *distance*), ACCH is a good method of matching which uses skipping internal data. When the reference characters are scanned, it combines status of scanned characters with the DFA depth information. Then it needs to scan a finite depth boundary characters. The ACCH algorithm can speed up matching by skipping about 60%-70%, but the value is affected by the actual situation of the LZ77 compression ratio and the pointers distribution. The jumping affect will be increased if the compression ratio is high. In compressed traffic networks, the ACCH’s speed reaches about 103.707MB/S in matching LZ77 mixed datasets, based on 1000 malicious snort patterns. But this speed is not enough for online services networks.

3.3 Distribution of pointers

CNT services like Motor Vehicle Auto Driving System, Amap navigation, SNS App, and Google App have more numbers than characters. On average, a complete network request contains more than 60 segments (60+ packets) and the total size of compressed traffic is 76.2 KB. The average compression ratio of network traffic is 70.4%, and in commonly used web

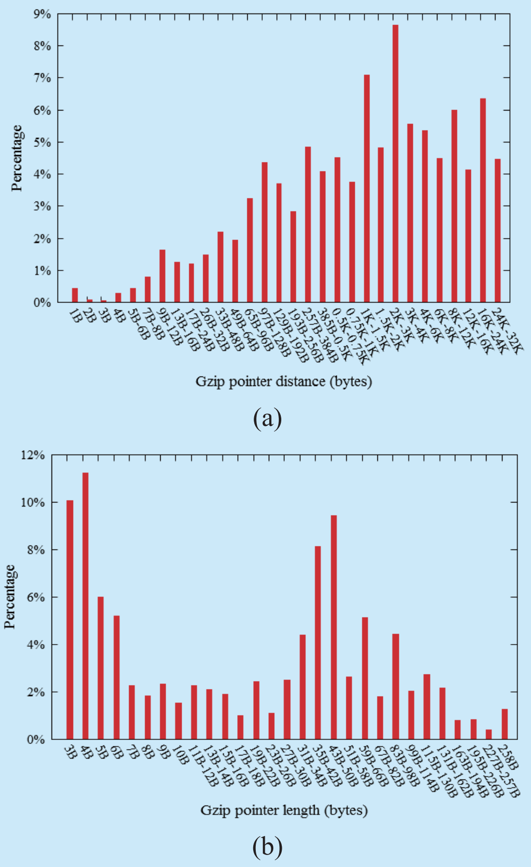

Fig.2 *Distribution of the pointer characteristics on CNT(a) distance of a pointer (b) length of a pointer*

pages is 78.4% as shown in figure 2.CNThas smaller number of longer length pointers as compared to web pages. Generally, the speed of matching is directly proportional to the *length* of pointers.

IV. Fast Multi-Pattern Matching Algorithm on Compressed Network Traffic

In this paper, we will focus on multi-service compressed network traffic, especially on performance matching on multiple navigation and surfing services that were compressed by gzip. Wu-Manber (figure3), a suffix matching algorithm, is an extended version of Horspool in multi-pattern matching. There is a special jumping sliding window, whose length is same as the length of minimum pattern. In preprocessing, we constructed a *Shift* table and *Hash* table with malicious characters and block sizes. *Hash* table stores the block size of pattern's end characters and pattern's index. *Shift* table stores step information for sliding windows. According to Wu-Manber's research, if the size of block is:

$$B = \log_{|\Sigma|}(2 \times l_{min} \times r)$$

It will perform the best sliding effect. In our scenario, one byte's max capacity is 256 different characters, so $\Sigma = 256$, $l_{\min} = 9$ and $r = 1000$. The optimal block size$B$ is therefore 2. Then the algorithm reads the last 2 bytes in the sliding window and selects *step* information from the *Shift* table. When the *step* is 0, the characters in the window may be a part of malicious pattern, so we need an accurate checking of the calculations. The space complexity for *Shift* table is:

$$O\left(\sum_{i=0}^{B-1} |\Sigma^{|} \times (2^{bitshift})^{i}\right)$$

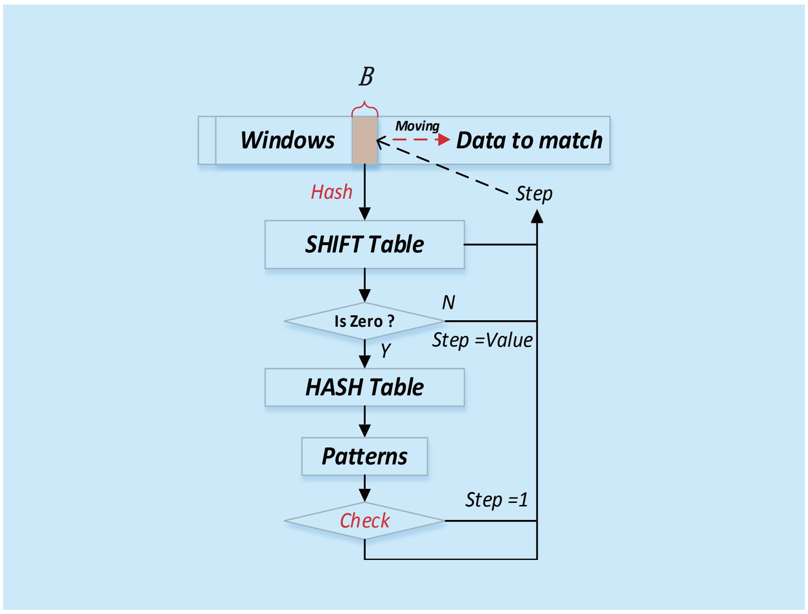

Fig.3 *Illustration of Wu-Manber Algorithm*

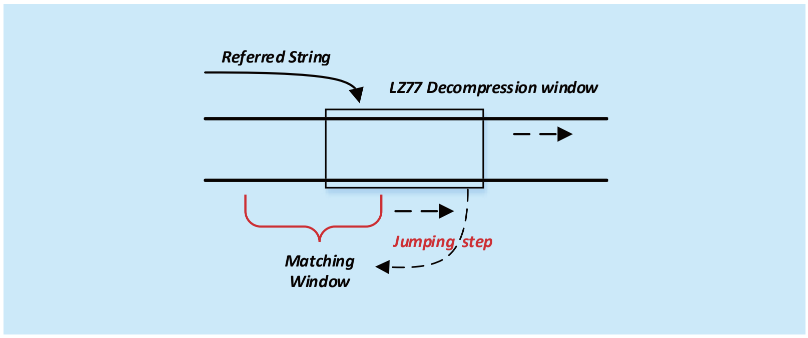

Fig.4 *Illustration of decompressingLZ77 and pattern matching*

4.1 Designing of FMMCN

In this section, we will present a novel FMMCN algorithm on the LZ77 mixed datasets [22]. In a 32KB sliding window, there are no more than three duplication bytes. In FMMCN algorithm, decompressingLZ77 and pattern matching are sequentially performed, as shown in figure4. The optimization distance between the LZ77 decompression window and matching window is$l_{\min} - 1$. The position of the LZ77 decompression window changes with the length of pointers.

4.1.1 Jump

Actually, referred string was matched, so we believe that parts of bytes in mixed datasets can be skipped in patterns matching. With the same input bytes, table must return same *Shift* stepand hash value. We designed a *HitStack* to store the location of hits and corresponding pattern information for every traffic session.The size of *HitStack* increases with the increase in number of hits. We can't skip all bytes in referred string, such as the boundary of bytes. In our design, the depth of boundary is $l_{\min} - 1$.Left side of boundary takes up one byte in front of referred string, and jumps to right side of boundary, so does the right boundary side in the back of pointer bytes. Obviously, only if the *Length* is more than $2(l_{\min} - 1)$ than Internal Area bytes can be skipped. Short pointer also needs to perform accurate pattern matching.

4.1.2 HitStack

It is an excellent strategy, to skip Internal Area, in which the result of multi-pattern matching is determined. When the Length is more than $2(l_{\min} - 1)$, LZ77 decompression window will calculate the *jumpingstep* for matching window and traverse the *HitStack* to deduce a new probable location of a hit as shown infigure5. Therefore, there are two ways to get a location of hit, which includes accurate pattern matching and deducing result (see detailed pseudo code in Algorithm 1). Actually, in one session, the size of *HitStack* is very small or sometimes even empty. *HitStack* stored index of every successful pattern matching and position for every session. It not only stores the final result set, but also has an important role in determining whether or not current pointer's bytes are replaced by referred string. In the search process, the direction of the *Hitstack*'s traversal is from top to bottom, and the time consumption of the traversal is negligible.

V. Experimental Results

In our experiments, we need two types of datasets, one of the online CNT and the other of malicious signatures, including open-source snort [23] and random malicious text. All of theCNTwas filtered from 10 devices. It was continuously captured and matched for more than 1000 hours. In these experiments 4different algorithms are implemented, namely AC, WM, SBOM, and ACCH, and their results are compared with FMMCN. These are excellent multi-pattern matching methods forum compressed data or LZ77 mixed datasets. The number of input patterns includes

Algorithm 1: **Deduce Hit Position with LZ77 Pointer**

```
FunctionDeduce (position, distance, length)
  For each i =0 to size of HitStack do
  Leftposition = HitStack.search (i, left)
    Rightposition = HitStack.search (i, right)
    Pattern_id= HitStack.search (i)
    If ((position- distance<= Leftposition) && (position-distance+ length=>
      Rightposition))
       HitStack.push (Leftposition + distance, Pattern_id)
    EndIf
  End for
```

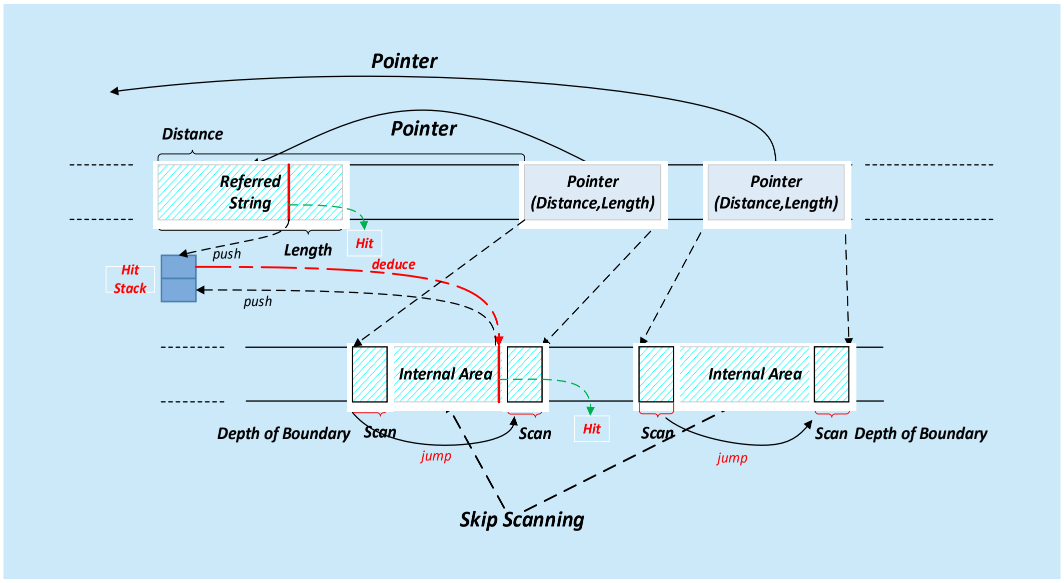

Fig.5 *Illustration of pointer and referred string in scanning. Internal area can be skipped*

Algorithm 2: Fast multi-pattern matching algorithm on CNT

```
FMMCN.Preprocessing is function of constructing the table of SHIFT and HASH.
For invariant patterns and network sessions, it's only need to initialize once.
HitStack stores successful matching.
    Function Preprocessing ($l_{\min}$, B, d)
        $B = \log_{|\Sigma|}(2 \times l_{\min} \times r)$ ⊿ Computation of B
        Initialize all elements of SHIFT to $l_{\min}$- B+1
    For i=0; i<d; i=i+1 do
        For q=m; $q \geqslant B$; q=q-1
            $h = h_1(t_{l_{\min}-B+1} \cdots t_{l_{\min}})$
            $shiftlen = m - q$
            $SHIFT[h] = MIN(SHIFT[h], shiftlen)$
            IF $shiftlen = 0$ Then
                $h' = h_2(t_{l_{\min}-B+1} \cdots t_{l_{\min}})$
                $HASH[h] = HASH[h] \cup \{i\}$
                $PREFIX[i] = h'$
            End IF
        End for
    End for
    Procedure FMMCN($Trf = t_1t_2t_3 \cdots t_n$, Plain_Size)
    Preprocessing($l_{\min}$, B, d)
    Construction of the tables of SHIFT and HASH.
    Searching
        position= $l_{\min}$
    While $position \leqslant Plain\_Size$ Do
        IF flex< position- blockMaxIndex+ mBlock-1 Then
            For p=0; p<$l_{\min}$; p++
                If thereexistsdistanceor length then
                    Memcpy ($Trf_{plain} + flex$,$Trf_{plain} + flex - distance$,length) ▹
                    copy distance and length's literal data
                    If length>2 ($l_{min} - 1$)
                        jump_step = length -2 ($l_{min} - 1$)
                        Deduce (position, distance, length)
                    Else
                        jump_step =0
                    End if
                    flex += length
                Else
                    Memcpy ($Trf_{plain} + flex$,$Trf_{plain} + flex$-distance,1)
                    flex++
                End if
            Endfor
        End If
        blockHash = HashCode($Trf_{plain}$+index- blockMaxIndex, mBlock)
        ⊿ compute Hash value
        blockHash = blockHash % mTableSize
        shift = mShiftTable[blockHash]
        shift =$l_{\min}$
    If position> 0
        position += shift+jump_step
    Else
        Check allcorresponding pattern in HASH table with $l_{\max}$ length of plain text
        If Matching
            HitStack.push (position,pattern.id)
        End if
            position++
    End If
End of while
```

1000,2000,3000,5000 and 10000. The algorithms are written in C++ and are evaluated on 10devices with Linux server, Windows server and android-based car terminal open system. For every environment, traffic contains more than 1596854 CNT packets whose size is 122957MB in compressed form. In FMMCN and ACCH when Huffman decoding is applied, it is changed in more than 314770MB in the LZ77 mixed datasets approximately. The average compression ratio of the vehicular network is 70.4% and the average length of a back-reference pointer is 48.8.

The Snort data-set have 3400 signatures, but most of them are binary and unfit for HTML searching. After removing no effect signatures, a subset of about 1,200 textual patterns is left. Snort datasets are not designed to be performed on vehicular or HTTP network. Thus, we mixed a proportion of random meaningful malicious text as part of patterns, including multi-byte and single-byte patterns. However, many researches refer Snort dataset as signatures, and since it makes us to introduce parts of effective dataset to evaluate FMMCN and ACCH. In experiments, we randomly extracted meaningful malicious pattern text. The length of all the patterns is 9 byte or more for different experiments.

5.1 Matching speed

AC, WM and SBOM all need to decompress packages into plaintext so their performance is low for multi-pattern matching on compressed data. ACCH and FMMCN can directly perform multi-pattern matching on a LZ77 mixed datasets. On average, Huffman decodes 34.4KB compressed traffic to 90KB as does LZ77 mixed datasets, which can be extracted to 290KB plaintext approximately in the end. Time consumption of Huffman decoding is negligible, though the time complexity is $O(\log(n))$.The speed can be calculated as:

$$speed = \frac{Bytes_{gzip}}{Time_{LZ77} + Time_{Matching}}$$

The best performance of FMMCN algorithm is 176.47MB/S for 1000 patterns, even faster than ACCH by about 70MB/S. FMMCN can ignore more than 89.5% of bytes during scanning, because it uses two types of jumps, i.e. *jump during sliding window and jump during scanning*. Table 1 and table 2 shows the comparison of time and speed for above mentioned algorithms respectively. The performance of FMMCN is better than all publicly available algorithms for CNT, as shown in Figure6.

5.2 Memory usage

The proposed algorithm FMMCN consumes the same memory as Wu-Manber. The memory usage of FMMCN depends on the size of the *SHIFT* table and the *HASH* table. In experiments we use 2 bytes as block size. FMMCN also supports efficient hash search to select *SHIFT* and *HASH* tables, after reading a block of characters. To reduce the probability of a hash collision, we choose to map the theoretical space onto a larger fixed space in the same order of magnitude. In most cases, FMMCN can show distinct advantages of memory space, as shown in figure7.

5.3 Preprocessing time

Preprocessing time is of great significance in real-time traffic filtering. The shorter preprocessing time makes our application work more effectively. In fact, the preprocessing time of FMMCH is similar with Wu-Manber on the support of matching. In 1,000 patterns and 10,000 patterns, preprocessing time of FMMCNis barely half of the time used by AC and SBOM, as shown in figure8.

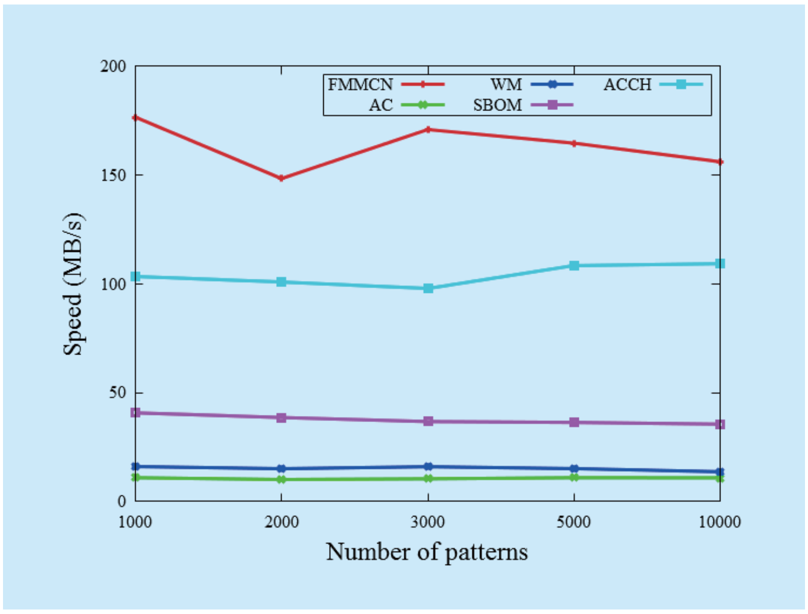

Fig.6 *Matching speed of AC, WM, SBOM, ACCH and FMMCN on real CNT*

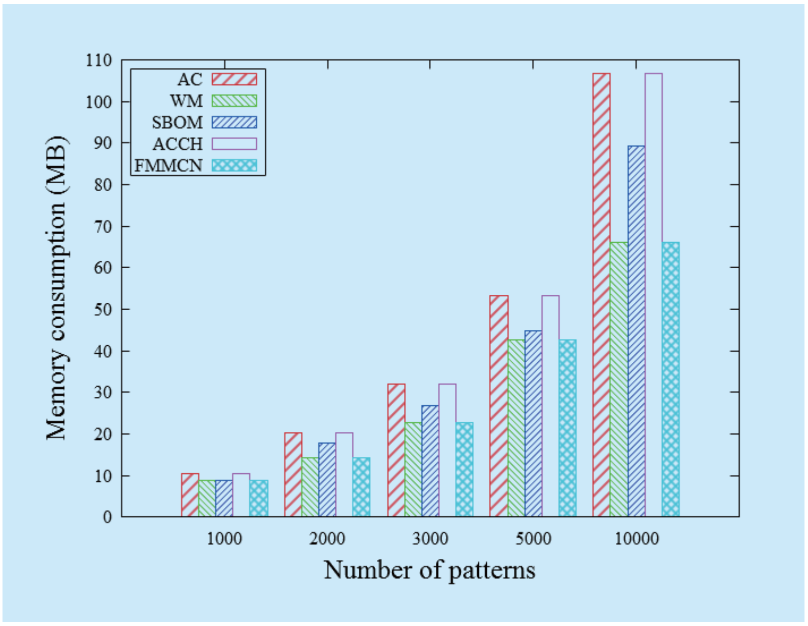

Fig.7 *Memory consumption of AC, WU-Manber, SBOM, ACCH and FMMCN*

Table II *time consumption(S) of decompressing and matching for 34.4KB CNT*

		AC	WM	SBOM	ACCH	FMMCN
	Decompressing LZ77 Time	0.00043	0.00043	0.00043	0	0
Matching time	1000patterns(500 Snort + 500 Random)	0.002723	0.001712	0.000415	0.000871	0.00051
	2000patterns(1000 Snort + 1000 Random)	0.002977	0.001815	0.000462	0.000893	0.000607
	3000patterns(1000 Snort + 2000 Random)	0.002876	0.001721	0.000507	0.000920	0.000530
	5000patterns(1000 Snort + 4000 Random)	0.002724	0.001849	0.000517	0.000830	0.000550
	10000patterns(1000 Snort + 9000 Random)	0.002702	0.002057	0.000527	0.000824	0.000580

Table II *Patterns matching speed (MB/S) of AC, WM, SBOM, ACCH and FMMCN on real life CNT of multi-byte web page.FMMCN is our contribution*

	AC	WM	SBOM	ACCH	FMMCN
1000patterns	10.91	16.059	40.71	103.329	176.47
2000patterns	10.1	15.08	38.57	100.772	148.27
3000patterns	10.41	15.99	36.71	97.792	170.778
5000patterns	10.91	15.09	36.33	108.327	164.534
10000patterns	10.856	13.671	35.528	109.209	155.979

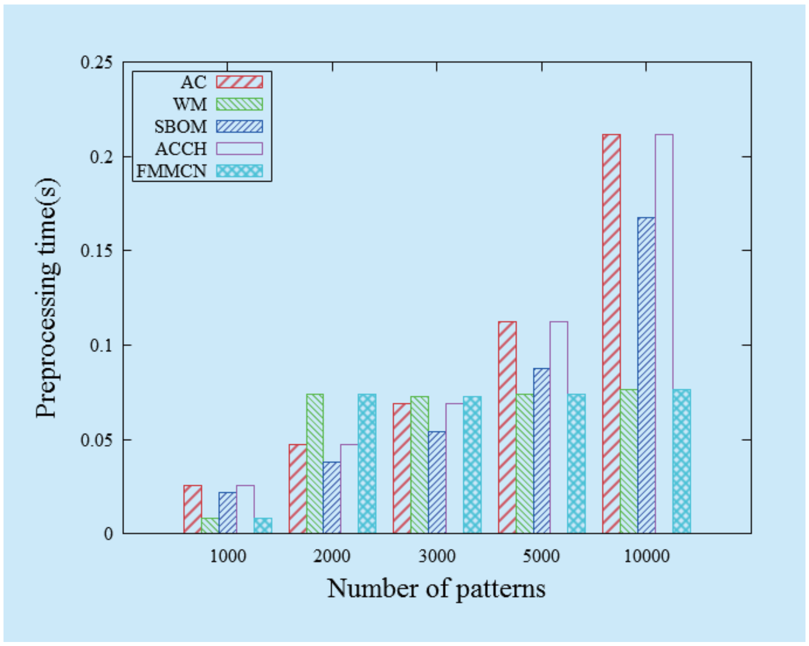

Fig.8 *Preprocessing time of AC, WU-Manber, SBOM, ACCH and FMMCN*

VI. Conclusion

In modern network security tools, multi-pattern matching algorithm is very important. The Gzip compression has become a widely used application standard in fast network transmission including vehicular network and HTTP service, etc. In compressed multi-pattern algorithms, FMMCN has achieved the best performance on CNT. The algorithm proposed in this paper can ignore more than 89.5% of bytes due to jumps at two different stages. The maximum speed reaches 176.470MB/S in 1000 patterns, even faster than ACCH by about 70MB/S. During performing pattern matching on compressed data, FMMCN is faster than AC, too. FMMCN is not intrusive for the WU-Manber algorithm. Therefore all the methods to improve Wu-Manber are also applicable to FMMCN. The proposed FMMCN algorithm is also suitable for various HTTP traffic. In future we are planning to implement FMMCN for NetFPGA platform to security matching of compressed traffic.

Acknowledgements

This work was supported by China MOST project (No.2012BAH46B04).

References

[1] Parul T, Deepak D, "Investigating the Security Threads in Vehicular ad hoc Networks (VANETs): Towards Security Engineering for Safer on-road Transportation,"2014 International Conference on Advances in Computing (ICACCI), pp 2084-2090, Sep,2014.

[2] Abdelmajid K, David S,"On the Suitability of Device-to-Device Communications for Road Traffic Safety,"2014 IEEE World Forum on Internet of Things(WF-IoT),pp 224-229, Mar,2014.

[3] PDeutsch, "Gzip file format specification," May, 1996, http://www.ietf.org/rfc/rfc1952.txt.

[4] JZiv and ALempel, "A universal algorithm for sequential data compression,"1977 IEEE Transactions on Information Theory, May, pp337– 343, 1977.

[5] R Prasad, SAgarwal, "Parameterized shift-and string matching algorithm using superalphabet,"2009 International Conference on Computer and Communication Engineering (ICCCE 2008), Sep, pp. 8-13,2008.

[6] AAho and MCorasick, "Efficient string matching: an aid to bibliographic search,"Communications of the ACM, pp 333–340, 1975.

[7] SWu and UManber, "A fast algorithm for multi-pattern searching," Department of Computer Science, University of Arizona, 1994.

[8] CharalamposS, KonstantinosG,"Exact online two-dimensional pattern matching using multiple pattern matching algorithms," ACM Journal of Experimental Algorithmic,Vol. 18, No. 2,2013.

[8] PDeutsch, "Deflate compressed data format specification," 1996, http://www.ietf.org/rfc/rfc1951.txt.

[9] Bassem M, Mohamed A, "Survey on Security Issues in Vehicular Ad Hoc Networks,"Alexandria Engineering Journal, pp 1115–1126, 2015.

[10] Anat B, Yaron K, "Accelerating Multi-pattern Matching on Compressed HTTP Traffic,"INFOCOM 2009, pp 397 – 405, 2009.

[11] Anat B, Yaron K, "Accelerating Multipattern Matching on Compressed HTTP Traffic," IEEE/ACM TRANSACTIONS ON NETWORK-

ING,Vol.20,pp 970-983, 2012.

[12] YAfek, AB, Y. Koral, "Space efficient deep packet inspection of compressed web traffic,"Computer Communications, Vol. 18, No. 2, pp 810-819, 2012.

[13] UManber, "A text compression scheme that allows fast searching directly in the compressed file," ACM Transactions on Information Systems (TOIS),Vol.807, pp 124 – 136, 1997.

[14] MTakeda, YShibata,"Speeding up string pattern matching by text compression: the dawn of a new era,"Transactions of Information Processing Society of Japan,Vol.43, pp 370–384, 2001.

[15] NZiviani, E. de Moura, "Compression: A key for next-generation text retrieval systems," Computer,Vol33, pp 37-44, 2000.

[16] Dana S, Ajay D, "Adapting the Knuth–Morris–Pratt algorithm for pattern matching in Huffman encoded texts,"Data Compression Conference(DCC 2004), March, 2004.

[17] Navarro G, Raffinot M, "Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences," Cambridge University Press, pp41-74, 2002.

[18] N. Tuck, T. Sherwood, "Deterministic memory efficient string matching algorithms for intrusion detection,"INFOCOM 2004,March 7-11, Vol.4, pp 2628-2639, 2004.

[19] T. Song, W. Zhang, "A memory efficient multiple pattern matching architecture for network security," INFOCOM2008, April,pp 166 – 170,2008.

[20] J. van Lunteren, "High-performance pattern-matching for intrusion detection,"INFOCOM2006, pp 1–13, 2006.

[21] Y. Liu, Q. Liu,"Speeding up pattern matching by optimal partial string extraction,"INFOCOM2011, pp 810–819, April, 2011.

[22] A. Amir, G. Benson, and M. Farach, "Let sleeping files lie: Pattern matching in z-compressed files," Journal of Computer and System Sciences, Vol.52, pp 299–307, 1996.

[23] Cisco, "Snort Dataset",https://www.snort.org.

Biographies

Hao Peng, is currently a Ph.D. candidate at the School of Computer Science and Engineering in Beihang University (BUAA), China. His research interests include network security, big datamining, cloud computing. Email: penghao@act.buaa.edu.cn.

Jianxin Li, associate professor at the School of Computer Science and Engineering, Beihang University. He received his PhD degree from Beihang University in 2008. He was a visiting scholar in machine learning department of CMU in 2015, and a visiting researcher of MSRA in 2011. His current research interests include data analysis and processing, distributed systems, and system virtualization.The corresponding author.Email: lijx@act.buaa.edu.cn.

Bo Li, assistant professor at the School of Computer Science and Engineering, Beihang University. His current research interests include information security, cloud computing, big data mining and analyticsand trusted computing. Email: libo@act.buaa.edu.cn.

M. Hassan Arif, is currently a Ph.D. candidate at the School of Computer Science and Engineering in Beihang University (BUAA), China. His research interests include datamining and cloud computing. Email: mhassanarif@act.buaa.edu.cn