Fast Pattern Matching in Compressed Data Packages

Michael S. Berger and Brian B. Mortensen, *Member, IEEE*

Abstract— **This paper targets deep packet classification (Layer 4-7) for applications within Firewalls and Intrusion Detection/prevention systems (IDS/IPS). Specifically targeting string matching applications, this paper will focus on processing of web-content, HTTP, at high speed (>10Gigabit). The main contribution of this work is in the field of combined payload search and HTTP decompression. In general, decompression is difficult to perform at wire-speed due to the additional data-amounts generated, which means that a decompressed 10 Gbit/s link could contain e.g. 30-40 Gigabit data. This paper presents a method which makes it is possible to find a match in the data packages of the data stream without the need for searching in the decompressed data stream, thereby avoiding the possibility of "data explosions".**

Index Terms—**Pattern Match, Compressed data, HTTP, Security, Firewall, Intrusion detection.**

I. Introduction

During the last 10 years, the Internet has become an all encompassing means for connecting computers globally. The ever increasing use of the Internet has resulted in more and more services being offered, and the increasing number of computers connected to the network has caused a veritable explosion in the amount of data transferred.

Another factor in the use of the public Internet is the presence of a great variety of malicious programs like viruses and worms or the like, which spread across the Internet. Detection and blocking of these programs is extremely important, not only for the private user, but even more for corporations, which rely almost entirely on a fully functional network, both internally for their users and externally with customers and partners. An infection of a corporate network with malicious software can be very costly and can render a corporate data network unusable for days and invaluable data may be lost. This can have a major financial impact for the corporation.

Also the providers of data networks like Internet Service Providers (ISPs) face the task of protecting their networks from infection. ISPs provide Internet connections to a large number of customers and can carry a significant amount of internet traffic on their backbone. Backbone links with capacity of 40 Gbps or more are now possible. The ISPs must employ means for protecting their infrastructure from interruption, thus causing downtime for the customers. They can also in some cases be contractually obliged to ensure that no malicious software penetrate from their network into their customer corporate network.

For corporate users and ISPs alike, it should be possible to apply protection to parts of the infrastructure, wherein data traffic from a large number of users flow. Historically these kinds of protection means have been implemented on the lower levels of the OSI model. Protection was provided by blocking or allowing specific ranges of IP addresses or TCP/UDP ports, i.e. layer 3 and layer 4 of the OSI model. However, this approach has turned out to be much too crude to provide sufficient protection.

A major challenge for deep packet inspection is related to inspection of HTTP traffic. This is due to the fact that compression is commonly used to reduce latency and increase bandwidth efficiency. Currently HTTP supports compression using the Deflate [1] algorithm or Gzip [2] format, which is also based on Deflate. Deflate compression utilises a combination of Huffman coding [3] and LZ77 [4] . Huffmann uses Prefix coding that represents symbols from a known alphabet by bit sequences (codes), e.g. A:00, B:1, C:011, D:010. Note in this example that no codeword is a prefix of another code word. LZ77 on the other hand is also a lossless compression technique that uses references to earlier occurrences of data; by an <offset,length> instruction, the decoder must go *offset* positions back and copy *length* number of data bytes to the actual decode buffer. Thus, this type of coding is very efficient if data contains repetitions of long sentences.

Pattern match methods typically de-compress the data before the actual search is carried out. This has the undesired effect that the necessary bandwidth may have to be many times the bandwidth of the incoming data. Therefore, searching in such de-compressed data is difficult or impossible to perform at line speed. To overcome this limitation, a new method is described in the following that enables inspection of compressed data without a full search in decompressed data stream. In the following sections of the paper, the approach is described in detail and current state-of-the art is reviewed.

Manuscript received March 15, 2010. Patent pending.

M S. Berger is with Technical University of Denmark, DTU Fotonik, Oersteds Plads, Bldg. 343, DK-2800 Lyngby, Denmark , e-mail: msbe@fotonik.dtu.dk).

B. B. Mortensen, was with Technical University of Denmark, DTU Fotonik, Oersteds Plads, Bldg. 343, DK-2800 Lyngby, Denmark , e-mail: brianbachmortensen@gmail.com).

978-1-4244-8865-0/10/$26.00 ©2010 IEEE

Following that, in section IV, application scenarios are given and finally section V gives the conclusion.

II. Related Work

Several papers have described work in the area of pattern matching for deep packet inspection applications. Originally, string matching algorithms such as Aho-Corasick [5] and Boyer-Moore [6] was developed to perform bibliographic search. The methods have been improved in several ways to perform fast pattern match in security applications, examples are [7][8]. Several other approaches to fast pattern matching is also proposed, e.g hashing using bloom filters [9], methods using FPGA [10][11] or TCAM [12]. At this point none of the above takes compressed data into account. The concept of searching on compressed data, however, is not new: In [13] a fast compression and decompression technique is proposed that enables direct search in the compressed data. However the technique is not applicable to the HTTP compression method that is in consideration in this paper. Several papers have been published on pattern matching in LZW/LZ78 compressed data. [14][15]. These schemes are simpler to manage than LZ77, which is a part of the HTTP deflate algorithm, because LZW/LZ78 does not contain back-references. The basic idea in these papers is to run the search on the compressed data using compressed keywords, but this imposes some problems with the HTTP compression scheme. A more recent paper [16] discuss' pattern matching in compressed HTTP traffic, and is one of the first papers on this topic. The paper proposes the ACCH (Aho-Corasick for Compressed HTTP) algorithm that improves the search speed by utilising that data provided by a back-reference has already been scanned for pattern matching. And that this knowledge can be used to save unnecessary scans. Several available security tools has the ability to decompress and perform search in compressed data, but often with a performance penalty due to normal decompression followed by pattern matching. This is furthermore quite vulnerable to DOS (denial of Service) attacks, because synthetic data with a very high compression degree can easily be constructed. The latest development release 2.6.8 from Snort will feature the ability to perform pattern matching in compressed HTTP.

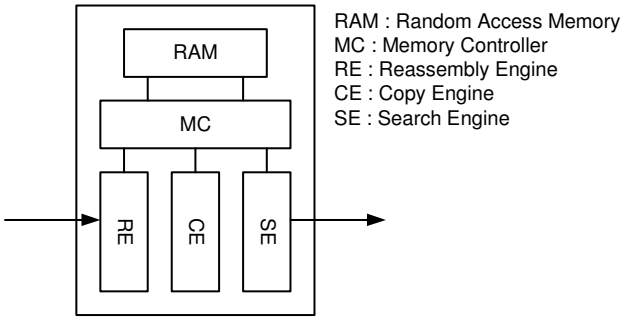

Fig. 1. High level block diagram of system to perform search in compressed data. The following figures show each component in more detail.

This paper presents a method which makes it possible to find a match in the HTTP packages of the data stream without the need for searching in the decompressed data stream, thus providing a further development than the schemes described in this section. The main contribution is a scheme to handle back-references during the compressed data search.

III. Pattern Match in Compressed data

The system on fig. 1 receives incoming <offset, length> compressed packet data and performs search for specific patterns in the packet payload. The data traffic might belong to several different flows, and the purpose of the Reassembly Engine (RE) is to collect segments from different flows to form complete information. The packet data is stored in the RAM by the Memory Controller (MC). At the same time, the Search Engine (SE) examines the data for predefined keywords <search strings>. The Copy Engine (CE) to copy a section of data in the RAM specified by the <offset, length> values in the data.

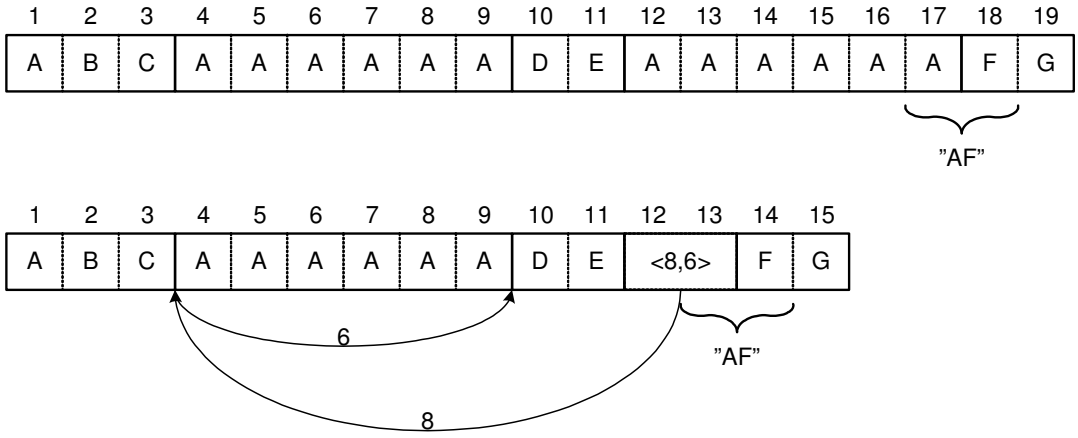

Fig. 2. Principle of <offset,length> compression. The second occurrence of the string "AAAAAA" is replaced by the pointer <8,6> indicating that 6 bytes must be copied from a position 8 bytes back.. It is important to take this backreference in consideration when e.g. searching for the pattern "AF"

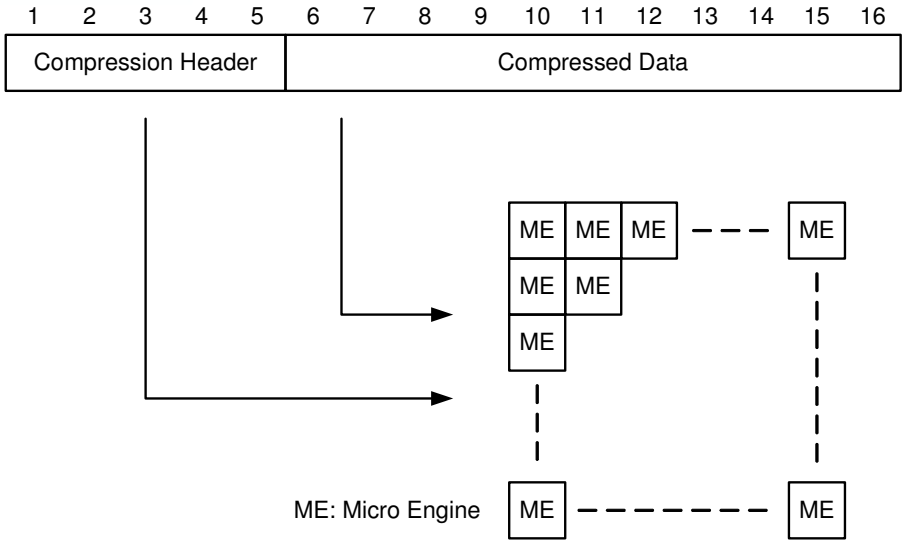

Fig.3. Shows the data package with compressed data. The packet header contains information about the compression scheme used. Micro Engines (ME) utilizes the compression header to perform compression of search strings enabling search in compressed data

The principle of <offset, length> compression in Fig. 2 illustrates an example where pattern "AAAAAA" is repeated, and the second occurrence is substituted by an <offset, length> field equal to <8, 6> pointing to the previous occurrence. The CE copies <length> bytes of data from (current position – offset) to current position. The amount of RAM storage is related to the maximum supported offset value, e.g. 32K. The search engine skips searching in already consulted text, and only the borders of the copied data is examined. This is exemplified on fig.2 with search pattern AF. Here, the A is contained in the border of the copied block.

Data compression using e.g. dynamic Huffman codes requires that the coding information is transmitted together with the compressed data. The Compression Header in Fig. 3 contains information on the compression strategy applied to the compressed data. The Compression Header is used by each Micro Engine (ME) to compress the search patterns. Each ME contains a specific search pattern which is compressed and compared to the compressed data contained in the packet. The ME's work in parallel and independently allowing high speed operation.

Fig. 4 shows the internal organization of each Micro Engine. The ME holds its specific search key in a register, and the content of this register is only modified in case the search pattern is changed. The Compression Engine (CE) performs the actual compression of the search key based on the compression information contained in the packet compression header. Typically, the compression algorithm is well defined, but the settings, e.g. Huffman tree, will be provided by the packet header. A comparator compares the incoming compressed packet data with the compressed search key, and in case of a match, a signal indication is given to the control/processor block. All MEs are configured by a shared control bus. Configuration includes search key settings and configuration of the CE.

Fig. 5 shows a detailed diagram of the memory structure and related logical components. We will refer to this diagram as the memory engine in the following. Compressed input data is stored in an input FIFO while the write pointer decision engine keeps track of this process. The write pointer decision engine controls the writing into the memory structure by asserting the "rd_ctrl" and "wr_addr" signals. If an <offset,length> command is found in the incoming data it is checked for validity and copy command is issued to the copy engine. The write pointer decision engine can continue writing data into the memory structure on the other side of the copy block. This feature enables the system to perform at line speed. If for some reason the write pointer decision engine needs to stall the incoming data flow, it can be accommodated due to the FIFO at the input.

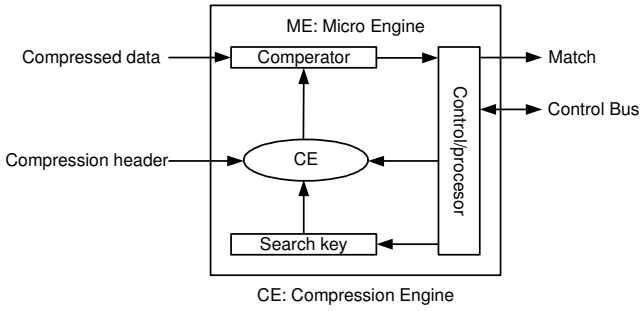

Fig. 4: The internal architecture of a Micro Engine ME

The write, read and copy directions are all progressing in the same direction, but not necessarily at the same speed. The

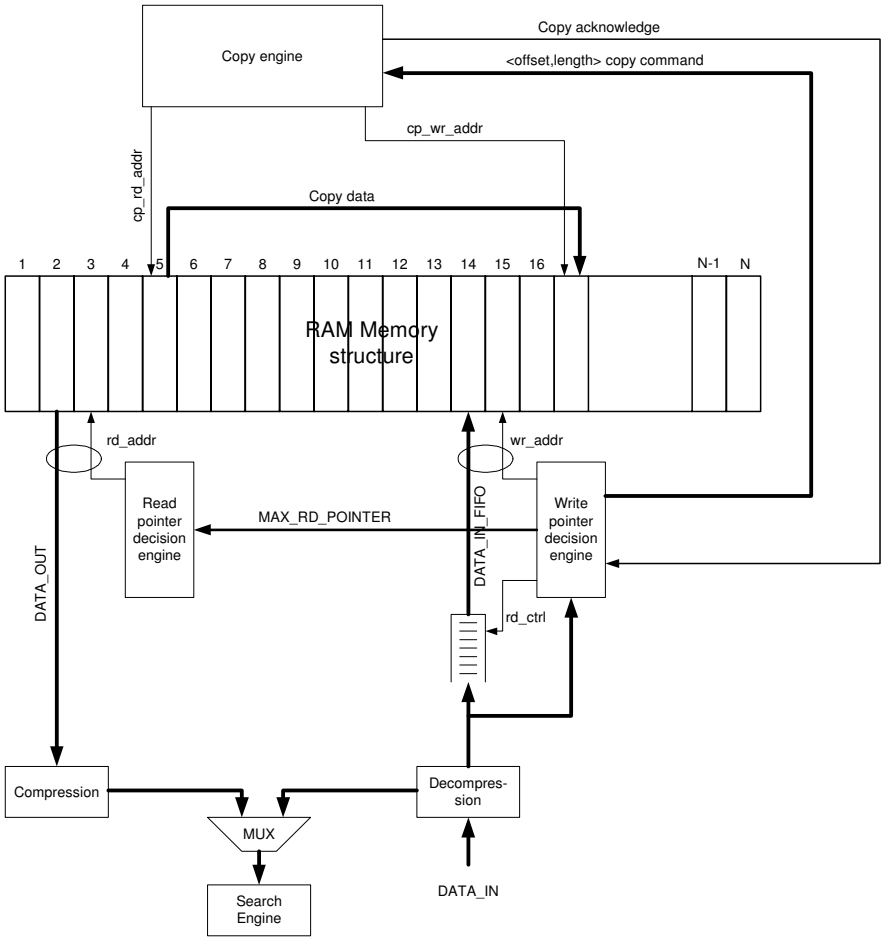

Fig. 5: Memory structure and related functional blocks. The search engine works in general on the received compressed data, but in order to handle <offset,length> back-references, a memory structure is provided that can provide a fast copy over a wide data bus. Only required data at the border of the back-reference is compressed again for further processing.

copy command forwarded to the copy engine contains the absolute start address of the block to be copied (abs_src_start_cp_addr), the absolute start address of the new location (abs_dst_start_cp_addr), and the length of the block to be copied (abs_length). More parameters may of course be provided. The absolute copy start address and the absolute start address of the new locations are calculated within the write pointer decision engine. The copy engine controls the cp_rd_address and the cp_wr_addr in addition to rd_enable and wr_enable of the respective ports on the multi port memory structure. Once a copy operation is complete an acknowledge signal is asserted (cp_ack), which indicates to the write pointer decision engine that the copy operation has completed. If multiple copy commands have been scheduled a copy identifier (cp_id) can be used for book-keeping purposes. Multiple copy commands may be pending in the copy engine, which keeps the commands stored internally in a FIFO structure. This feature is useful if the multi-port memory structure for one or another reason needs a small time period without any writing activity. This might be the case, for instance, when using DRAM (but not limited to), to build the multi-port memory structure. The read pointer decision engine can read data from the multi-port memory structure using the signal rd_addr and other dedicated control signals. The read decision engine is partly controlled by the write pointer decision engine, in the sense that a maximum read pointer is applied. This pointer must ensure that the read pointer does not progress be-yond a point of valid data. The read pointer engine writes the data from the multi-port memory component into the data_out FIFO. Reading the data from this FIFO is done by an external block, based on the FIFO control signals.

The copy engine notifies the write pointer decision engine whenever a copy command is executed successfully. This allows the write pointer engine to keep accurate track of the latest position which can be searched for patterns. This information is indicated through the MAX_RD_POINTER going to the read pointer decision engine. This makes it possible to design a simple FIFO like read interface. The read interface consist of a data bus and a selection of simple control signals indicating if unread data exist and are ready for transmission. Fig. 5 shows a total of 4 read and write ports to

the memory, but the concept does not limit the actual number of read/write ports on the memory.

IV. Applications

The above described method can be used in a number of important application areas. Firewall applications where viruses, malware or other malicious content can be hidden in the compressed web traffic present such an area. Finding links within the compressed web traffic which are not allowed due to legal or policy constraints due to explicit content is another important application area. Searching for such information with traditional methods is not fast and the presented method indicated that very improved performance can be made. If un-allowed content is found in the traffic the stream can be blocked or the individual packet can be dropped, forbidding the HTTP session to restart or retransmit. The content or keywords which are searched for can be related to parental control or terrorism related words. When finding such illegal information in web traffic the source and destination addresses of the corresponding flows can be logged and sent to appropriate authorities based on local laws and regulation.

Furthermore, searching or indexing keywords related to advertising is another important area where the proposed method can speed up current solutions significantly. The solution can be used to speed up the process of indexing web servers for specific content. Web traffic that is parsing a central point in a network may also be used to generate customer relevant advertisement. The gathered information may be combined in combination with geographical user information in order to target the advertisement even better.

Enabling mobile operators to prioritize traffic flows based on deep packet inspection are another application where the above presented method can be used, enabling better performance metrics based on the actual content (http://www.ccpu.com/articles/2009/bringing-deep-packet-inspection-dpi-into-wireless-networks/)

V. Conclusion

This paper has shown a method to overcome the limitations of pattern matching in compressed data packages for HTTP traffic. The GZIP/deflate compression algorithms behind HTTP compression are more complex because they use a combination of Huffman coding and LZ77 (back-references).

Therefore it is difficult to completely avoid decompression in some way, but with the proposed scheme, it has been avoided to search in the decompressed data. Decompression is only used to handle back-references and by only compressing the necessary parts on the boundaries of the back-references, a huge benefit is foreseen. While the hardware based solution will have a significant performance gain, future work will include a proof-of-concept demonstration in real hardware to measure the size and cost of the proposed scheme.

References

[1] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[2] Deutsch, P., " GZIP file format specification version 4.3", RFC 1952, May 1996.

[3] Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers, September 1952, Volume 40, Number 9, pp. 1098-1101. W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[4] Ziv J., Lempel A., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.

[5] Aho, A.; Corasick, M. "Efficient string matching: An aid to bibliographic search". Communications of the ACM 18 (June 1975): pp 333–340.

[6] R. S. Boyer; J. S. Moore , "A fast string searching algorithm". Communications of the ACM 20 (1977):pp. 762–772

[7] T. Song, W. Zhang, D. Wang , Y. Xue, Memory Efficient Multiple Pattern Matching Architecture for Network Security, Proc. of the 27th IEEE INFOCOM, 2008.

[8] J. van Lunteran, "High-performance pattern-matching for intrusion detection", IEEE INFOCOM, 2006.

[9] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, J. Lockwood, "Deep packet inspection using parallel bloom filters", IEEE. Micro, vol. 24, no. 1, pp. 52-61, Jan. 2004

[10] Singaraju, J. Bu, L. Chandy, J.A. "A signature match processor architecture for network intrusion detection", 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2005. FCCM 2005.

[11] H, Song, J. W. Lockwood "Efficient packet classification for network intrusion detection using FPGA", Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays, pp. 238 - 245

[12] F. Yu, R. Katz, and T. V. Lakshman. "Gigabit Rate. Packet Pattern-matching Using TCAM", in Proc. of. IEEE ICNP, 2004.

[13] E. de Moura, G. Navarro, N. Ziviani, R. Baeza-Yates. "Fast and Flexible Word Searching on Compressed Text", ACM Transactions on Information Systems (TOIS) 18(2):113-139, 2000

[14] Kida T., Matsumoto T., Shibata Y., Takeda M., Shinohara A., Arikawa S., "Collage system: a unifying framework for compressed pattern matching", Theoretical Computer Science, Volume 298, Number 1, 4 April 2003 , pp. 253-272(20)

[15] G. Navarro and J. Tarhio. Boyer-Moore string matching over Ziv-Lempel compressed text. In Proc. CPM'2000, LNCS, 2000

[16] Bremler-Barr, A. and Koral, Y., "Accelerating Multi-patterns Matching on Compressed HTTP Traffic," INFOCOM, 2009.

[17] www.snort.org