Multiprocessing scalable string matching algorithm for network intrusion detection system

Adnan A. Hnaif*, Ali Aldahoud, Mohammad A. Alia, Issa S. Al'otoum and Duaa Nazzal

Faculty of Science and Information Technology,
Al Zaytoonah University of Jordan,
P.O. Box 130 Amman, 11733, Jordan
Email: adnan_hnaif@zuj.edu.jo
Email: Aldahoud@zuj.edu.jo
Email: dr.m.alia@zuj.edu.jo
Email: issa@zuj.edu.jo
Email: science@zuj.edu.jo
*Corresponding author

Abstract: With high increasing speed of today's computer networks which affects the performance of security issues in terms of detection speed, the traditional security tools such as firewall is insufficient to protect the networks from external threads. Intrusion detection systems (IDS) are one of the most reliable tools that can be used to monitor all the network traffic to identify unauthorised usage of computer system networks. In this paper, we have proposed a scalable string matching algorithm based on network IDS (NIDS) to enhance the speed of NIDS detection engine, which called multiprocessing scalable string matching algorithm for network intrusion detection system (MSNIDS). The MSNIDS implemented by using enhanced weighted exact matching algorithm (EWEMA) in both sequential and parallel processing. The MSNIDS based on EWEMA can be achieved more than 89% in sequential processing time compared with WEMA, and 86% in parallel processing time compared with sequential matching processing.

Keywords: string matching algorithms; distributed architecture; parallel processing; network intrusion detection system; NIDS.

Reference to this paper should be made as follows: Hnaif, A.A., Aldahoud, A., Alia, M.A., Al'otoum, I.S. and Nazzal, D. (xxxx) 'Multiprocessing scalable string matching algorithm for network intrusion detection system', *Int. J. High Performance Systems Architecture*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Adnan A. Hnaif is an Associate Professor at the Computer Science Department, Faculty of Science and Information Technology, Al Zaytoonah University of Jordan. He received his PhD in Computer Science from the University Sains Malaysia – National Advanced IPv6 Centre and Excellence (NAV6) in 2010. He received his MSc of Computer Science from the Department of Computer Science in 2003, and obtained his Bachelor of Computer Science from the Department of Computer Science in 1999/2000. His researches focus on the network security, parallel processing and computer algorithms.

Ali Aldahoud is a Full Professor at the Computer Science Department, Faculty of Science and IT, Al Zaytoonah University of Jordan, senior in IEEE, ACM, MIAENG, and SMASDF. He received his PhD in Engineering Science, National Technology University of Ukraine 1996. His researches focus on distributed system (communication), algorithms and E-systems.

Mohammad A. Alia is a Full Professor at the Computer Information Systems Department, Faculty of Computer Science and Information Technology, Al Zaytoonah University of Jordan. He received his BSc in Science from the Alzaytoonah University, Jordan in 2000. He obtained his PhD in Computer Science from the University Science of Malaysia in 2008. During 2000 until 2004, he worked at Al-Zaytoonah University of Jordan as an Instructor of Computer Sciences and Information Technology. Then, he worked as a Lecturer at Al-Quds University in Saudi Arabia from 2004 to 2005. Currently he is working as a Faculty Deputy Dean and Chair of Computer Information Systems Department at Al Zaytoonah University of Jordan. His research interests are in the field of cryptography, network security, and computer networking.

Issa S. Al'otoum is an Assistant Professor at the Computer Information Systems Department, Faculty of Computer Science and Information Technology, Al Zaytoonah University of Jordan. He received his BSc in Electrical and Communications Engineering – Peshawar University 1 Pakistan; and obtained his Master from the University of Jordan, and received his PhD in Computer Science from the University of Khartoum. His research interests are in E-Gov. and computer algorithms.

Duaa Nazzal is a Research Assistant at Al Zaytoonah University of Jordan, she obtained her BSc and Master from the Al Zaytoonah University of Jordan. Her research interests are in network security, parallel computing and computer algorithms.

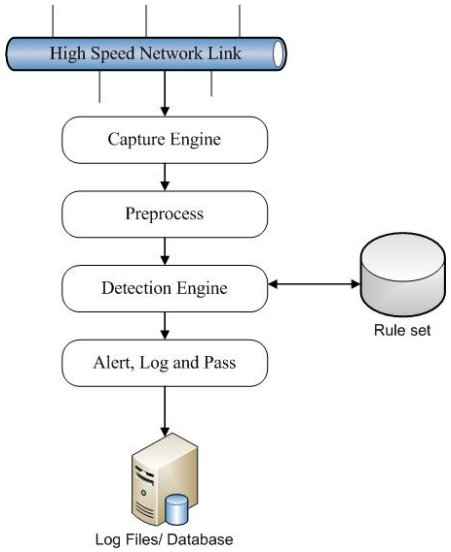
1 Introduction

With the development of computer and network communication technology, computer networks spread rapidly over the past few years ago. The internet plays an effective role for information exchange and sharing in our society. Hence, this raised the possibility of having malicious users gain illegal access to organisations for the purpose of stealing the information they are interested in, or rather, for destroying it by injecting applications (Malwares). Further, a number of network attacks are dramatically increasing by the time, ranging from the denial of services, IP spoofing eavesdropping, man in the middle attack masquerading, and Malware attacks (Snehal and Jadhav, 2010).

In fact, detecting malicious activities occurring in computers or networks can be achieved by using the intrusion detection system (IDS), which is considered a security management system that monitors network traffic, and raises an alert when capturing malicious activities. The IDS have been widely employed in many organisations to detect attacks.

On the other hand, firewall is used to prevent an intrusion within the networks by restricting the access among them. Nevertheless, it is not used to report or to find attacks or threats inside the networks. The IDS is responsible for finding and reporting unwanted entries to the system. In practice, IDS is required for detecting malicious traffics that cannot be detected by general deployed tools as such firewalls.

Figure 1 IDS architecture (see online version for colours)



IDSs are normally formed from four components, namely the decoder, the pre-processor, the detection engine, and the alert module (Snehal and Jadhav, 2010). Figure 1 depicts the real time IDS architecture.

1 the decoder or capture engine: this engine is used to capture all the incoming packets from the network

2 pre-processor engine: used to prepare the captured packets for the detection engine

3 detection engine: used to check all the pre-processed packets against any possible intrusions

4 alert, log, and pass engine: used to generate a suitable level of alert to the network administrator.

Intrusion detection approaches are categorised into two types: the signature-based detection (SD) and the anomaly-based detection (AD).

The SD approach defines a pattern that matches a particular attack. This approach is extremely efficient in identifying common attacks or threats. Nonetheless, it is not easy to keep patterns up-to-date. Besides, this approach is considered to be ineffective in detecting unknown threats or attacks (Liaoa et al., 2012).

The AD approach is extremely efficient in finding new vulnerabilities. In particular, this approach works on the basis of defining the network behaviour (profile). Next, the defined profile is compared with monitored events and activities that can detect significant attacks. The main disadvantage of this approach is its high dependency on the profile definition, where not well-defined profiles can lead to weaken the accuracy in detecting attacks or threats (Jyothsna et al., 2011).

Many studies have shown that the string pattern matching is one of the primary performance bottlenecks that are related to these systems. The key challenge of the pattern string matching is that its performance requirement has dramatically increased (for example, from multi-Gbps to multi-10s of Gbps), and has outpaced the performance of existing solutions. In particular, if the capacity of the network intrusion detection system (NIDS) cannot match up with the speed of the network, a passive NIDS will then cause to the dropping of packets, and thus, may miss attacks, whereas a real time NIDS will create a bottleneck for the network performance. On the other hand, as the number of potential threats and their associated signatures are expected to grow, the cost of the pattern matching is likely to further increase. Consequently, the pattern matching algorithm needs to be highly efficient. The ability to detect encountered attacks in an efficient and with a high speed manner is considered to be one of the most shortcomings of the IDS.

2 Background and literature reviews

2.1 Background

This section elaborates a comprehensive literature researches, and discusses the findings from the literature that is based on the NIDS detection engine domain.

2.2 Literature reviews

Pattern matching engine (PME) is available in each modern NIDS detection engines. Basically, a set of patterns within the rule is compared by the pattern matching algorithm with the payloads of the packets. Pattern matching is intensive when it is being computed.

It is considered that the string matching can reach a percentage of 70% to 80% for the NIDS CPU cycles, and is extremely costly when it is being operated in web firewalls. This is due to the fact that it is considered to be a critical building block when payloads are being inspected by network security applications (Jamshed et al., 2011).

Many pattern matching algorithms are proposed recently in a context that is based on the network intrusion detection. Accordingly, in the following subsections, some of the most famous algorithms that are used in NIDS are discussed in detail.

2.2.1 Exact string matching algorithms

Boyer-Moore algorithm (BM)

An algorithm that searches through a pattern of a particular text for the index of the first pattern occurrence is proposed by Boyer and Moore (1977). In fact, the BM algorithm is considered to be one of the most common and practical method to be used for a particular single pattern matching.

In particular, BM is considered to be the best and the essential algorithm for single pattern matching algorithms for which the SNORT tool can make use of it. BM is based on the sliding window concept that is appropriate for pattern matching algorithms. In order to have the number of required comparisons in a text of length 'T' being reduced, the BM must be based on shifts that are longer than a single step. This can be performed based on the shifting process pertaining to the pattern 'P' of the length 'n' to the right in longer steps, particularly, of less than 'm' characters. The time and space complexity is O(m + σ) for the pre-processing phase and Omn for the searching phase.

Quick search algorithm (QS)

QS is a simplification of the BM algorithm that is considered to be the easiest and extremely fast for a short pattern and a large alphabet. The QS algorithm only uses the bad-character shift table, and pre-processes the pattern 'P' by using a modified bad shift array, which is called the qbad_shift.

The searching phase of the algorithm has a worst case time complexity of O (mn). In the case of each time, a shifting distance is maintained as one, whereas the bad character is found in the last comparison of the P [0] along to the corresponding text. Next, the algorithm starts the comparison from right to left after a single attempt where the window is positioned to the text factor y [j ... j + m-1, and the length of the shift is at least equal to one. Accordingly, the character y [j + m] is necessarily involved in the following attempt. In this algorithm, the shift is based on the text character immediately, and is followed by the current alignment instead of the last text character of the alignment (Sunday, 1990).

2.2.2 Intrusion detection based on packets content

The NIDS deploys the network information searching technique based on certain rules. The pattern matching is performed on each packet's content in order to conduct the intrusion detection (Wang and Kobayashi, 2006).

Before the results are being displayed, the NIDS rules analysis must be described to constrain a range of patterns lengths and payload length, which are being traced to find the pattern. Aldwairi (2006) illustrates that a percentage of 87% of the rules contain strings to match against the packet payload or against the packet content. Moreover, Munz et al. (2007) shows a percentage of 88.6% of all rules are satisfied by the first 145 bytes of the payload. In fact, this implies that most of the signatures that are related to the attacks can be found in the first 145 bytes.

In the following subsections, some of these improved algorithms, which are based on two processing techniques are highlighted and are discussed thoroughly. These two processing techniques comprise: sequential processing and parallel processing.

Comment [a1]: Author: Please note that per the Inderscience typesetting guidelines, all listed numbers/bullets are formatted as left-aligned and not justified, due to this, the listed entries are formatted as subsection. Please confirm of correct

Sequential processing

In this subsection, the related works which are based on the detection engine for the packet content by using exact matching algorithm are discussed in detail.

Zhang (2009) proposes a new feature pattern matching algorithm, which first arranges letters in the pattern string form from a low appearance probability to a high appearance probability. Next, each is matched one by one by using existing pattern matching algorithms. This algorithm first matches the rule heads, and then the option heads. Finally, it matches the payloads of data packages in order to find the intrusion.

Friedl (2006) develops an automaton algorithm, which is a mathematical model for the finite state machine. In general, there are two types of automata: the non-deterministic finite automaton (NFA) and the deterministic finite automaton (DFA). If a state q jumps to multiple different states with only one input α, or with an empty input ε existing in the transition, then this automaton considers the NFA.

In the pattern matching area, the NFA of a string is said to be easily obtained. Nonetheless, an NFA might contain multiple active states, while the DFA might only contain a single active state. Hence, a faster searching speed can be in

the DFA for pattern matching where the entire NFAs can be possibly converted into DFAs.

Hasan and Rashid (2012) suggest enhancing the BM Horspool (BMH) algorithm by adding a hashing function that is called the HBMH in order to reduce the characters comparison time.

The HBMH algorithm uses the hash function, which converts the string into numbers. The main benefit of using the hash function is to reduce the number of character comparisons that are performed by the BMH algorithm in each attempt. Thus, it reduces the required comparison time.

Parallel processing

In general, there are two major techniques can distribute a single job into different processors. These comprise the multi-processing technique, and the multi-threading technique. The multi-processing technique uses a 'fork' for creating an additional separate process manually in order to execute the entire later codes after the creation of the process, such as running a new particular program. In fact, the idea of the multi-threading technique is to divide up a single large task into multiple small tasks, which can be scheduled to be executed by the operating system.

- Multithreading technique

  The supra-linear packet processing is an achievement that is developed by the (Intel Corporation in 2006) based on Snort 2.x. The data acquisition component of the supra-linear is separated, where other components are duplicated. A packet classification hash module is added to dispatch the packets into processing threads where they make use of one and four execution cores. The data acquisition and the dispatch component is put in a single thread, and each processing component is a separated thread itself. Each processing thread executes the same code in order to go through a flow from a decoder to an output where each of these processing threads does not communicate with each other.

  The multi snort is a multi-thread Snort, which is presented by Schuff et al. (2007), and is based on the Snort 2.6. In comparison with the supra-linear packet processing, the Multi Snort only executes multiple instants of the original Snort in a parallel manner. Besides, it proposes a strategy of a memory sharing. In addition, Chen et al. (2009) designed a parallel structure for a high-performance NIDS. A new structure named the Para-snort contains a data source module, one or more load balancing module, multiple processing modules, and an output module.

- Hnaif (2015) proposes a new platform that aims to enhance the speed of the packet payload detection engine in the NIDS in sequential and in parallel modes. The proposed platform NIDS is based on the weighted exact matching algorithm (WEMA), which can detect the intruders that are trying to gain access through to the network by using the packet payload information. On the other hand, the proposed platform is able to run on a single core and multi-cores processor in order to show that the idea could cope up with the traffic arrival speed and with various bandwidth demands.

- Multiprocessing technology

  Al-Mamory (2012) uses a multi-processor technology with a LAN of computers in order to parallelise the Snort's string matching engine. A LAN of nine computers is used in the evaluation. Three different string matching algorithms are used (KMP algorithm, Karp-Rabin algorithm, and BM algorithm) in order to check the efficiency, and the behaviour of these algorithms in the distributed multiprocessing environment. These algorithms have different behaviours with respect to the length of the pattern.

  Li (2005) used a multi-processor technology to parallelise the AC algorithm. The aim from the paralleled AC algorithm is to enhance the speed of the detection engine for NIDS. Their methodology that is called NIDS PME is based on defragmenting each captured packet into chunks. The defragmentation function depends on the available number of processors as well as the packet payload length, i.e., if the number of processors = 2 and the length of the packet payload is 20 characters (text), and then this packet will be defragmented into two portions. The first portion of length ten will be sent to the processor number 1, and the second portion of length ten will be sent to processor number two, where each processor contains a copy of the same rule sets. If one of the two processors has a matching, then the final result will return to the first processor (main). But if the pattern exists between the portion number one and portion number two, then neither processor number one nor processor number two will have a matching. In this case, the amount of the false negative will increase. To avoid this problem, the authors proposed that if one portion ends with the tk (the last position in the first part), the next portion will start from tk-patternlength +2. Thus, no possible occurrence of a pattern will exist between the portions.

3 The proposed (MSNIDS) and methodology

There are many techniques or algorithms of the literature reviewed that were discussed with the improvements of the NIDS detection engine, in both sequential and parallel processing. However, with the increase in the network speeds, it is important not to only cope-up with a new technique(s) that performs particular tasks, but to cope-up with a technique(s) that outperforms the entire previous methods.

Moreover, the work load of the pattern matching is considered to almost represent half of the workload of the SNORT-NIDS detection engine such as Aho-Corasick (AC) algorithm based on the single thread single processing algorithm. Therefore, the speed of execution can be

considerably increased if the detection engine process is accelerated through the parallelisation process.

In this section, the design of the improvement methodology is presented in order to be applied to exact string matching algorithm that is used in the SNORT-NIDS in order to produce speed reductions. The proposed methodology (MSNIDS) is based on hybrid parallel techniques using shared and distributed memory architecture. In particular, different components of the proposed hybrid parallel techniques are described with their relevance and components in order to be efficiently used while building those components. In the following subsections, the sequential processing is at first discussed. Besides, how rule sets are generated and applied is described. Furthermore, the parallel processing with the number of the proposed improvements are elaborated in detail.

3.1 Sequential matching processing

The main objective of this scenario is to have the proposed methodology (MSNIDS) run efficiently, to have it work with one process as a sequential match, and to have it create a basic prototype for the enhance WEMA (EWEMA). In fact, EWEMA composes from two logical sub-phases, pre-processing and matching sub-phase.

First, the pre-processing sub phase must be conducted prior to the start of the matching sub-phase. This sub-phase will run only once (in both sequential and parallel scenarios), as long as there are no updates available in the rule sets. Basically, it will create an alphabetical index matrix of weight 'W' of the payload rule set, which defines about 3,000 rules based on the SNORT-NIDS rule sets. Each character has its own position in the text 'T' (indices). Once the rule sets are created, it is stored in the L1 cache of the main core for performing further matching. Finally, the incoming packets will be stored in the L2 cache in order to obtain higher speeds for the proposed system.

Second, the matching sub-phase, since the EWEMA is chosen to be used as a string matching algorithm, the number of steps must then be conducted in matching sub-phase, so that it could be possible to find the exact matching between the incoming packet payload 'P', and the payload rule set 'R.S' as follows:

a Create an array list 'L' for each incoming packet payload.

b Determine the minimum character weight of the pattern 'P', which refers to the minimum number of occurrences of each character in the matrix 'M'. If the minimum character weight is equal to zero, then stop the matching process since the pattern 'P' does not exist in the text 'T'. Otherwise, proceed to step D.

c Determine the index of the first character of the pattern 'P' to start the matching from the index of the minimum character weight that is greater than the index of the first character. Thus, the number of comparison will be decreased.

d Create the attempt matching process of the array list 'L' by adding the minimum weight character, which is selected in Step B under the corresponding character position of the array list 'L' (index [i]).

e Compare the next and the previous characters of the pattern 'P', which its indices are: Index [i + 1] and index [i – 1], with the corresponding characters of the matrix 'M'. If both exist, then continue matching with index [i + 2] and index [i – 2] until the end of the pattern 'P' is reached, or until an exact match is obtained.

f If a mismatch occurs, and then read the next occurrence of the minimum character weight of the pattern 'P', and repeats from step D.

Nonetheless, saving the processing time is considered to be one of its main objectives. Thus, in order to provide an optimisation for the overall matching process (in both sequential and parallel scenarios), the best way for saving time will then not be based on creating an index matrix weight every time the matching phase starts against every single row of the rule set, but will rather be based on creating an index matrix weight for the entire rows in the rule set once starting up with the pre-processing sub-phase since no updates are available.

The indexed table of the rule set must only be copied once to the CPU for the duration of the program. Despite the fact that the input text is changed and the string searching kernel runs again, the pointers are only passed in such a way the pattern indices are stored into the CPU memory, and not into the actual character. The last step that is required to be performed prior to the launching of the CPU kernel is to specify the memory space where the kernel stores the output of the algorithm that will be running. This memory space is reserved.

In particular, the data has to be retrieved where it is now placed in the reserved memory space of the CPU function that started the kernel in the first place. In order to save the memory transfer bandwidth, the output of the algorithm is purely the ID of the pattern that is found, and the location of the input text that is located. The final operations of the output are processed by the CPU. In a normal setting, this data can be processed into a database.

3.2 Parallel matching processing

This section only focuses on the number of parallelism techniques that are used in the proposed methodology (MSNIDS). MSNIDS implemented in parallel processing by using hybrid parallel techniques. The hybrid parallel architecture was implemented as follows.

Generally, a job can be parallelised by the function parallelisation, the data parallelisation or the pipelining. If there are no dependencies held for different jobs, they can be then processed in parallel manners by the function parallelisation. A large amount of data, which has the same processing steps can be processed in a parallel manner by the data parallelisation. Nevertheless, if a single job is

divided up into multiple sections, and the work loads of each section are similar, then the pipelining can be used for the parallelisation process.

For the string matching algorithms that are previously mentioned, it is easier to see that there are no many independent functions. Hence, the function parallelisation is by then considered not to be a good option.

On other hand, the type of parallelisation which is a result of identical operations being concurrently applied on different data items is called data parallelism. Since all tasks perform similar to each other in terms of the computations, the decomposition of the problem into tasks is usually based on the data partitioning. In MSNIDS, to distribute the incoming packets in a multicore environment, the data parallel model has been used. Mainly, the work performed in cores, and the packets that is operates in different cores are different. A simple idea for the parallelisation of the packets is to divide a set of packets into multiple subsets of packets in order to be distributed over available cores.

The master-worker model (Buyya, 1999) was used, as it was confirmed in practice that it is the most appropriate for string matching on message-passing systems (Cringean et al., 1988). A node of the computer cluster was designated as the master while the rest of the nodes were set as workers. The data set was initially stored in the local storage of the master and was subsequently exported to the worker nodes. In order to decrease the execution time, the available packets must be decomposed and mapped to the available processes to minimise the overhead which resulting from the time the processes stay idle due to the uneven distribution of the load, while the second level conducted by multicore shared memory architecture. At the first level, the nodes loaded the packet set to the host memory. At the second level, the pre-processing sub-phase of EWEMA is conducted by the master node.

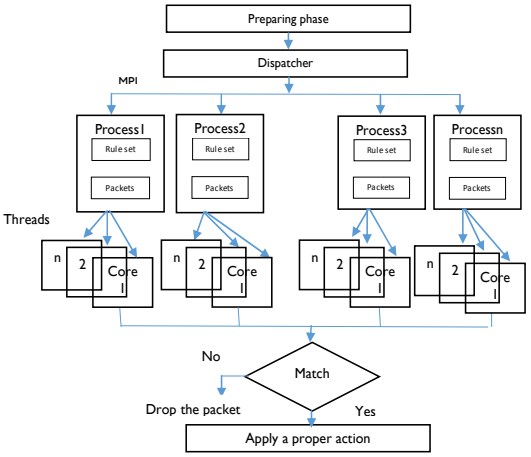
At the third level, the resulted pre-processing array was copied to the global shared memory of each worker node and was then bound to the texture cache of the device. At the fourth level, each node calculated the size of the input packets chunk based on the size of the cluster. The input packets chunk was then retrieved to a pinned memory area of the host and was subsequently set for copying to the global memory of the device. In practice though, when the specific functions were used to load data exported from TCP, a significant performance penalty due to their line-buffering behaviour.

The worker processes execute in a very simple cycle where it obtains the message with the packet, processes the packet, and sends the result to the master worker. Occasionally, the communication takes place only between the master and the workers while only it is rarely that the workers perform communicates with each other.

In addition, the distribution of packets is entirely performed at the beginning of the computation, which in turn allows the master to participate in the computation after each worker is allocated a fraction of the work. The allocation of packets is performed in a cyclic way.

According to the analysis of the sequential EWEMA in previous section, the most expensive section of a string matching algorithm is to examine if the character of the pattern matches with the character of the rule sets. To avoid this cost, the matching sub-phase, which contains the matching process between the characters of the packet and the rule sets, it will be parallelised using multicore shared memory technique inside each worker processor which conducted at the fifth level. The matching phase in the EWEMA string matching algorithm is carried out using multicore environments with shared memory architecture. Figure 2 illustrates the overall system architecture of the hybrid parallel implementation.

Figure 2 The EWEMA hybrid parallel matching phase architecture (see online version for colours)



As shown by Figure 2, the MSNIDS executed the program by divided the entire packets into subdivided parts through fork and join operations, the master thread distributed the works to the worker threads. The parallel EWEMA start execution the program in sequential processing conducted by the master thread until the algorithm reach the matching sub-phase function, at this moment worker threads generated for matching sub-phase function. The worker threads executing the matching sub-phase functions and return the partial result to the master thread, the master thread will assemble all the results with the help of join operation and show the output, this operation performed in sequential processing, the worker threads will terminate itself automatically after send the results to the master thread.

At the sixth and final level, the number of matches that each node computed was gathered by the master node and its CPU was used to determine the total amount of matches of the packet, and the total execution time.

Finally, a hybrid parallelisation technique that is proposed based on the combination of the advantages of the shared and the distributed memory parallelisation's of a clustering system, which consists of multiple interconnected multicore computers that are entirely based on a hierarchical model.

4 Analysis and experimental results

The rule sets of this test are formed by using the SNORT Rule sets. In practice, it consists of 3,000 unique content rules, which consists of lengths ranging from 1 to 80 bytes. For each individual test, a group of trace files are compiled from the real network traffic that contains various packet lengths and different packet sizes ranging from 50 packets to 2,500 packets.

The trace files contain malicious and non-malicious data that can match on the NIDS rule sets. Based on the two scenarios, the same trace files are used, so that the results are comparable.

4.1 Analysis scenarios

This section will describes two scenarios, first scenario presents the analysis of sequential matching process, and the second scenario presents the analysis of parallel matching process consequently.

4.1.1 Analysis of sequential matching process

In the proposed methodology (MSNIDS), the rule contents are used from the entire rule sets in a rule group in order to construct one index matrix table called the weighted matrix 'M', where 'M' is implemented by using a hash table. Based on this implementation, the rule sets are represented in the form of a two dimensional array. In this array, each column represents the corresponding indices of the rule sets character, where the number of rows is equal to the number of English language characters. Each cell included in this table forms a data structure, which contains a number of integers that represent the indices of that character as shown in Table 1.

Table 1 The weighted matrix 'M'

Alpha. char.	*Indices of the rule set characters*					
	1	*2*	*3*	*4*	…	*n*
a(14)	1	2	13	20		44
b(9)	3	4	8	18		45
c(3)	29	30	33			
d(8)	9	16	21	38		43

Table 1 shows a simple example on how the index matrix table would look like. For instance, the indices value of the character 'c' are 29, 30, and 33, which identifies where the character 'c' would be found in the rule sets.

The next step after conducting the index table matrix in a two-dimensional array is to copy the array to the CPU global memory. An additional array of offsets is constructed in order to retrieve the correct table for comparison purposes when a set of packets is received. After that, the packets are now ready to be transferred to the CPU starting with the matching processing by using the EWEMA matching algorithm.

Table 2 Reading the pattern 'P = gcagagag' and creating the array list 'L'

Position	*1*	*2*	*3*	*4*	…	*n*
Array list 'L'	g	c	a	g	…	g

Once the captured packets enter into the system, the matching phase can then start. First of all, the minimum character weight of a pattern 'P' is determined, then the minimum weight character is added into an array list 'L' under the corresponding character position of the array list 'L' (index [I]) in order to start the first attempt as shown by Table 2. In order to decrease the number of comparisons being made for the minimum weight character, the index of the first character in the pattern 'P' is stored in a variable to start the attempts from the index of the minimum weight character that is greater than the index of the first character of the pattern 'P'. Furthermore, the next and the previous characters are compared in the pattern 'P', where its indices comprise: index [i + 1] and index [i – 1] if both exist, then continue matching with index [i + 2], and with index [i – 2] until reaching the end of the pattern 'P', or until obtaining an exact matching. If a match occurs, then store the index of the matching patterns, where the pattern is matched in the rule set. In case of a mismatch, read the next occurrence of the minimum character weight of the pattern 'P'.

4.1.2 Analysis of the parallel matching processing

The parallel processing approaches are mainly divided into two categories

- Auto-parallelisation: sequential programs are automatically parallelised by using the instruction level parallelism (ILP) or the parallel enabled compilers.
- The parallel programming approach comprises: the splitting mechanism of a problem into a set of tasks, and the development of a distributive mechanism that maps those tasks into the processors in an efficient manner. Therefore, it requires more attention from the programmer, making it more difficult to be coded when compared with the auto-parallelisation approach. Consequently, it achieves a higher execution performance.

The hybrid parallel architecture is implemented as follows: the master-worker model is used where it is considered the most appropriate model to be used for the pattern matching in the distributed memory systems. A node of the computer cluster is designated as the master, while the rest of the nodes are set as workers. The data set is initially stored in the local storage of the master, and is subsequently exported to the worker nodes.

In the first level, the pre-processing phase of the EWEMA is sequentially executed by the master node, then the master distributes the rule set index matrix table to the host memory of the worker nodes. In the second level, the resulted pre-processing arrays are copied to the global memory of the CPUs, and are by then, bounded to the texture cache of the device. In the third level, the necessary

data structures are sequentially computed from the CPU for each node in the form of arrays. The reason behind this computation is to ensure that the CPU receives a set of packets from the master node. In the early stages of the implementation, each node uses the standard I/O of the system. In the fourth level, the searching phase of the EWEMAs is executed in parallel by the number of threads. In the fifth and the final levels, the number of matches for which each node is computed is gathered by the master node based on the use of the MPI Reduce() function of the MPI, and its CPU that is used to determine the total amount of matches of the pattern, or the pattern set in the input string.

In each worker, the array of packets is collectively copied by the entire threads of each thread block into to the shared memory of the device in order to reduce the pressure over the texture caches. In addition, the threads collectively make an access into the input string characters, and store them to the shared memory of the device in order to work around the coalescing requirements of the global memory.

However, a hybrid parallel program is modelled by using the data parallel model. Basically, it is considered as a set 'd' of 'n' data that is denoted by {d1, ..., dn}. Each data di, i ∈ [1, n] is potentially a parallel data that is composed of a set of 'ni' segments. Each parallel segment di, j, i ∈ [1, n], j ∈ [1, ni] may further be composed of bi, j of potential parallel code blocks, which is denoted by bi, j, k, i ∈ [1, n], j ∈ [1, ni], k ∈ [1, bi, j].

There is a performance penalty that is involved when applications that introduce data dependencies are executed in parallel. A significant latency is introduced during the launching of the CPU cores, and also when copying the data between the host and the device memory due to the bandwidth of the system bus. In order to address some of these deficiencies, this section presents a hybrid parallel architecture that combines the distributed memory and the shared memory that can implement the EWEMA on a homogeneous cluster of CPU nodes.

The hybrid parallel architecture is implemented as follows: the master-worker model is used where it is considered the most appropriate model to be used for the pattern matching in the distributed memory architecture. A node of the computer cluster is designated as the master, while the rest of the nodes are set as workers. The data set is initially stored in the local storage of the master, and is subsequently exported to the worker nodes. The space and time complexity of the EWEMA algorithm are shown in Table 3.

Table 3 Space and time complexity of the EWEMA algorithm

Scenario	*Pre-process phase*	*Searching phase*	
		Best case	*Worst case*
Sequential	O (mn)	Ω(c)	O (LV)
Parallel		Ω(c)	O (L/Z) or O (LV/ZV)

where c: constant, L: payload length, V: minimum number of occurrences and Z number of processors

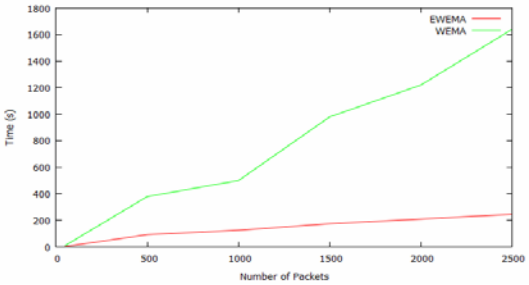
4.2 Experimental results

The proposed methodology (MSNIDS) is based on the EWEMA that is tested in a sequential processing, and in a parallel processing. The results are explained in detail as follows:

4.2.1 Sequential matching processing

In this section, the actual results are obtained according to the comparisons that are performed between the implementations of the WEMA and the EWEMA, which are presented in order to measure the enhancements that are performed for the WEMA technique. A range of 50 packets to 2,500 packets are read from the file. The payload rule sets have a size of 3,000 rules. In order to evaluate the effectiveness of the proposed NIDS that is based on the EWEMA, a comparison is performed with the WEMA.

Figure 3 The comparison result between WEMA and EWEMA in sequential matching processing (see online version for colours)

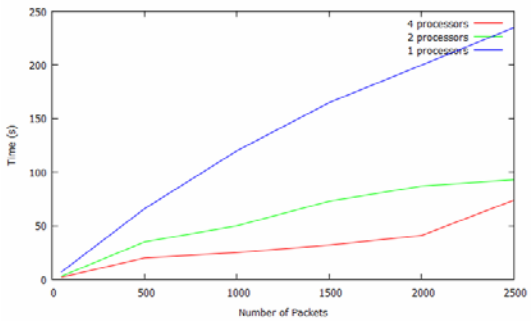


The comparison results are presented in Figure 3. It can be seen that the performance of the proposed NIDS based on EWEMA and WEMA, which are very close to the first 200 packets. Thus, the enhancement in some cases is not implemented (i.e., when the minimum weighted character has a first index, which is greater than the index of the first character of the processed packet), which makes the implementation of both algorithms seem be closer. After this point, it is obvious that the proposed NIDS, which is based on the EWEMA, has an advantage in comparison with the WEMA. In addition, it is clear that the proposed NIDS based on EWEMA is faster after 900 packets than the WEMA. The achieved improvement reached a proximate percentage of 89%.

4.2.2 Parallel matching processing

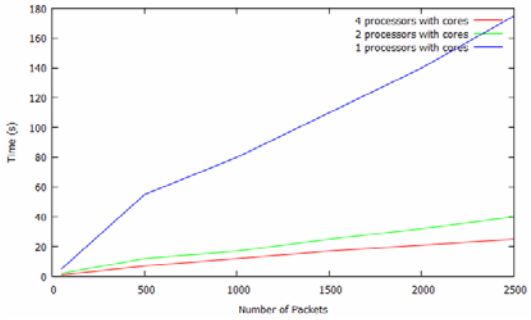
This section discusses the efficiency of the presented hybrid parallel processing, which is used for the implementation of the EWEMA.

Figure 4 EWEMA running on one processor (sequential processing) vs. parallel processing using two and four processors (see online version for colours)



Mainly, the pre-processing phase of the algorithm is sequentially executed by the CPU of the master node. As depicted from Figure 4 and Figure 5 the searching time of EWEMA decreases the proportion to the size of the packet file.

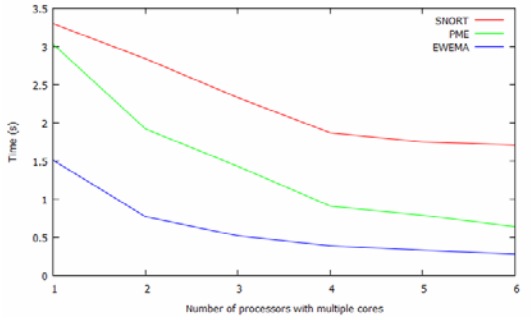
Figure 5 EWEMA running on one processor (sequential processing) vs. parallel processing using two and four processors with three cores (see online version for colours)



As presented in Figure 4, and Figure 5, the searching time different according to the size of packets. The number of processors are ranged from 1 to 4 processes with one and 3 cores in each processor. The minimum consumed time can be conducted by using four processes which decreased the searching time of 2,500 packets from 232 seconds in one process to 31 seconds. From the results we can concluded that, by using parallel processing, our methodology MSNIDS can achieved more enhancements in term of speed of NIDS detection engine.

We repeat the same experiment by applying a rule set of 1 MB in size and 3,000 input data into current SNORT, and PME technique and the proposed EWEMA. The comparison result between the current SNORT and the result of the PME technique and the result of the proposed EWEMA is depicts in Figure 6.

Figure 6 Snort-NIDS vs. PME vs. EWEMA on 3,000 packets input data (see online version for colours)



As shown in Figure 6, we can note that the EWEMA platform has obtain the best results between all techniques, then the PME technique, and the current SNORT obtained the worst results.

In fact, the aim of using the parallel programming is to use 'p' processors in order to execute a program several times faster than it being executed on a single processor. The ratio of the sequential execution time to the parallel execution time is called the speedup, which measures the increase in running time due to parallelism. The speedup can be defined as follows:

$$\text{speed up} = \frac{T_s}{T_p} \tag{1}$$

where T_s denotes to the sequential execution time, and T_p denotes to the parallel execution time.

The second performance measurement is the efficiency. The efficiency of the parallel computation measures the fraction of time for which a processor is usefully utilised, which is also called the processor utilisation. The efficiency is the speedup that is divided up by the number of processors:

$$\text{efficiency} = \frac{\text{speed up}}{P} \tag{2}$$

where P denotes to the number of processors.

Finally, the overhead which is the factor that make the parallel code run slower than the expected, when compared to the serial code can be identified as:

$$To = (P * Tp) - Ts \tag{3}$$

where P denotes to the number of processors, Tp denotes to the parallel execution time, and Ts denotes to the sequential execution time.

As a result, the overhead that can be achieved in parallel processing of MSNIDS is a negative value. This implying that the speedup on 'p' processors could exceed 'p' and the reason for this is the super liner speedup. In this case, the effective computation speed of EWEMA is slower on a serial processor than on a parallel computer using similar processors.

5 Conclusions

Several improvements have been created in this paper to increase the speed of an NIDS detection engine, these improvements including two scenarios, in sequential and parallel matching processing.

In sequential scenario, our methodology improved the WEMA in both preparing and searching phase. The results showed that EWEMA is faster compared to Boyer-Moore algorithm at rate between 30%–40%.

On other hand, hybrid parallel techniques were discussed and used in parallel scenario including multi core shared memory and multi process distributed memory techniques.

Evaluation results demonstrate the performance potential of our hybrid parallel techniques using different number of packets, and different of packets length. The best speed up can be achieved by using two processes with three cores to reach 2.9 speed up. The efficiency and the overhead also tested and achieved 22 and 30 respectively.

The performance improvement made to the existing systems which rely on hybrid parallel techniques is roughly 65%, compared with the sequential scenario.

Acknowledgements

We would like to thank Al-Zytoonah University of Jordan – Faculty of Science and Information Technology for its support thatenabled us to complete this work.

References

Aldwairi, M. (2006) *Hardware-Efficient Pattern Matching Algorithm and Architectures for Fast Intrusion Detection*, dissertation, Computer Engineering Dept., North Carolina State University.

Al-Mamory, S.O. (2012) 'Speed enhancement of snort network intrusion detection system', *Journal of Babylon University, Pure and Applied Science*, Vol. 20, No. 1, pp.10–19.

Buyya, R. (1999) *High Performance Cluster Computing: Programming and Applications*, p.2, Pretice Hall PTR, New Jersey.

Chen, X., Wu, Y., Xu, L., Xue, Y. and Li, J. (2009) 'Para-snort: a multi-thread snort on multi-core IA platform', *Proceedings of Parallel and Distributed Computing and Systems (PDCS)*.

Cringean, J.K., Manson, A., Wilett, P. and Wilson, G.A. (1988) 'Efficiency of text scanning in bibliographic databases using microprocessor-based, multiprocessor networks', *Journal of Information Science*, Vol. 14, No. 6, pp.335–345.

Friedl, J.E.F. (2006) *Mastering Regular Expressions*, 3rd ed., O'Reilly, Media, Inc. Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Hasan, A.A. and Rashid, N.A. (2012) 'Hash-Boyer-Moore-Horspool string matching algorithm for intrusion detection system', in *Proc. International Conference on Computer Networks and Communication Systems (CNCS)*, Vol. 35, pp.20–25.

Hnaif, A.A. (2015) 'A new platform NIDS based on WEMA' *International Journal of Information Technology and Computer Science (IJITCS)*, Vol. 7, No. 6, p.52.

Jamshed, M.A., Lee, J., Moon, S., Yun, I., Kim, D., Lee, S., Yi, Y. and Kargus, K.P. (2011) 'A highly-scalable software-based intrusion detection system', in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, ACM, pp.317–328.

Jyothsna, V., Prasad, V. and Prasad, K. (2011) 'A review of anomaly-based intrusion detection systems', *International Journal of Computer Applications*, Vol. 28, No. 7, pp.26–35.

Li, J.Y. (2005) 'A parallel NIDS pattern matching engine and its implementation on network processor', *Proceeding of the 2005 International Conference on Security and Management*, pp.375–381, CSREA Press, Las Vegas, USA.

Liaoa, H-J., Lina, C-H.R., Lina, Y-C. and Tunga, K-Y. (2012) 'Intrusion detection system: a comprehensive review', *Journal of Network and Computer Applications*, Vol. 36, No. 1, pp.16–24.

Munz, G., Weber, N. and Carle, G. (2007) 'Signature detection in sampled packets', The *2nd Workshop on Monitoring, Attack Detection and Mitigation*, Toulouse, Toulouse, France.

Schuff, D.L., RynChoe, Y. and Pai, V.S. (2007) 'Conservative vs. optimistic parallelization of stateful network intrusion detection', *International Symposium on Performance Analysis of Systems and software (ISPAS)*, IEEE International Symposium, Austin, pp.32–43.

Snehal, B. and Jadhav, P. (2010) 'Wireless intrusion detection system', *International Journal of Computer Applications*, Vol. 5, No. 8, pp.9–13.

Sunday, D.M. (1990) 'A very fast substring search algorithm', *Communications of the ACM*, Vol. 33, No. 8, pp.132–142 [online] http://dx.doi.org/10.1145/79173.79184.

Wang, Y. and Kobayashi, H. (2006) 'High performance pattern matching algorithm for network security', *International Journal of Computer Science and Network Security*, Vol. 6, No. 10, pp.83–87.

Zhang, H. (2009) 'Design of intrusion detection system based on a new pattern matching algorithm', *International Conference on Computer Engineering and Technology (ICCET)*, IEEE, Vol. 1, pp.545–548.