In-Depth Packet Inspection Using a Hierarchical Pattern Matching Algorithm

Tzu-Fang Sheu, *Member*, *IEEE*, Nen-Fu Huang, *Member*, *IEEE*, and Hsiao-Ping Lee

Abstract—Detection engines capable of inspecting packet payloads for application-layer network information are urgently required. The most important technology for fast payload inspection is an efficient multipattern matching algorithm, which performs exact string matching between packets and a large set of predefined patterns. This paper proposes a novel Enhanced Hierarchical Multipattern Matching Algorithm (EHMA) for packet inspection. Based on the occurrence frequency of grams, a small set of the most frequent grams is discovered and used in the EHMA. EHMA is a two-tier and cluster-wise matching algorithm, which significantly reduces the amount of external memory accesses and the capacity of memory. Using a skippable scan strategy, EHMA speeds up the scanning process. Furthermore, independent of parallel and special functions, EHMA is very simple and therefore practical for both software and hardware implementations. Simulation results reveal that EHMA significantly improves the matching performance. The speed of EHMA is about 0.89-1,161 times faster than that of current matching algorithms. Even under real-life intense attack, EHMA still performs well.

Index Terms—Network-level security and protection, network security, intrusion detection, pattern matching, content inspection.

1 Introduction

Network services are extremely important since many companies provide services over the Internet. A variety of Internet-based applications have created a strong demand for content-aware services, network policy, and security management. Furthermore, increasing amounts of important information exist in packet payloads. Therefore, low-layer network equipment is inadequate for checking the information, since it only checks specified fields of the packet *headers*. High-layer network equipment providing in-depth packet inspection, such as intrusion detection systems (IDSs), application firewalls, antivirus appliances, and layer-7 switches, is a prerequisite in a network. Such equipment typically contains a policy or rule database applied to finding certain packets over the network. Every rule in the database consists of several patterns (also called signatures) and a matching action (or a series of actions). These patterns describe the fingerprints of packets.

The network equipment applies the predefined patterns to identify and manage the monitored packets over the network. Different network equipment may have different pattern databases applied, respectively, to attack detection, bandwidth management, load balancing, and virus blocking over the network. However, they have similar features in terms of patterns and matching procedures. The number of patterns is typically a few thousands, and the lengths of the patterns are varied. The patterns may appear *anywhere* in any packet *payload*. Consequently, the emerging high-layer network equipment needs a *pattern detection engine* capable of in-depth packet inspection, which searches the entire packet *headers* and *payloads* for pattern matching. Network equipment then employs the detection results to manage network systems intelligently. For instance, Snort is an open-source network-based intrusion detection system (NIDS) and is adopted for detecting anomalous intruder behavior with a set of patterns and generating logs and alerts from predefined actions [1]. One of the patterns of Nimda worm is described as "GET/scripts/root.exe?/c+dir." When the detection engine of Snort finds this pattern existing in a packet, the corresponding alert is generated to warn network administrators. The pattern matching is considered as the most resource-intensive task in the Snort detection engine [2]. Hence, this study focuses on the nascent issues of the payload inspection.

The most important part of a detection engine is a powerful multipattern matching algorithm, which can efficiently process the pattern matching task to keep up with the growing data volume in the network. However, conventional string-matching algorithms are impractical for packet inspection [3], [4], [5]. Due to the large pattern database, an effective detection engine must be able to search for a set of patterns simultaneously, rather than iteratively performing the single-pattern matching. While considering implementation issues of the network equipment, the performance of processing packets is not only affected by the computation time but also strongly affected by the memory latency. As is well known, the rate of improvement in processor speed exceeds that of improvement in memory speed [6]. The gap has been the largest problem for system builders. Therefore, the vital issue of designing a high-speed detection engine is to reduce the number of external memory accesses [8].

- *T.-F. Sheu is with the Department of Computer Science and Communication Engineering, Providence University, 200 Chung-Chi Rd., Shalu, Taichung 433, Taiwan, R.O.C. E-mail: fang@pu.edu.tw.*
- *N.-F. Huang is with the Department of Computer Science and Institute of Communication Engineering, National Tsing Hua University, 101, Section 2, Kuang-Fu Rd., Hsinchu 30013, Taiwan, R.O.C. E-mail: nfhuang@cs.nthu.edu.tw.*
- *H.-P. Lee is with the Department of Applied Information Sciences, Chung Shan Medical University, 110, Section 1, Jianguo N. Rd., Taichung City 402, Taiwan, R.O.C. E-mail: ping@csmu.edu.tw.*

Manuscript received 17 Aug. 2007; revised 12 May 2008; accepted 17 Sept. 2008; published online 6 Oct. 2008.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2007-08-0114.
Digital Object Identifier no. 10.1109/TDSC.2008.57.

This study proposes a novel *Enhanced Hierarchical Multipattern Matching Algorithm (EHMA)* for fast packet inspection, which simultaneously searches the packet payload for a set of patterns. This study contributes modifications to the hierarchical matching algorithm (HMA) [9] and introduces the idea of a *sampling window* and a *Safety Shift Strategy* in addition. EHMA is a two-tier and cluster-wise matching algorithm and can perform fast skippable payload scan. Based on the occurrence frequency of *grams*, this study discovers a small set of signatures from the *patterns* themselves to narrow the searching domain. A Min-Max strategy is used in the EHMA. The hit rate of the first-tier table in the EHMA is minimized, while the spread of patterns in the second-tier table is maximized. Accordingly, EHMA significantly reduces the number of memory accesses and pattern comparisons. EHMA can skip unnecessary payload scans by applying the proposed *Safety Shift Strategy*, which is based on a *frequency-based bad gram heuristic*. The frequency-based bad gram heuristic is a modification of the *bad grouped character heuristic* of Wu-Manber (WM) algorithm [10]. Therefore, EHMA has the advantages of both HMA and WM.

The memory space and the number of external memory accesses required by the proposed EHMA are much smaller than those required by state-of-the-art multipattern matching algorithms. EHMA needs less than 40-Kbyte memory space to construct required tables for the Snort patterns and, therefore, enables small-scale and cost-effective hardware implementations. Using only 768-byte on-chip memory, EHMA reduces the average number of external memory accesses to 0.06-0.19 and, thus, significantly improves the matching time of the detection engine. Simulation results reveal that EHMA outperforms the state-of-the-art algorithms. Even under real-life intense attack, EHMA still outperforms others. Because it employs only basic instructions and two small index tables, EHMA is very simple for hardware and software implementations. Consequently, the proposed EHMA is a very cost-effective and efficient mechanism for real-life network detection systems.

The rest of this paper is organized as follows: Section 2 presents previously proposed pattern matching algorithms and the fundamental definitions. Section 3 then describes the proposed EHMA in detail. Next, Section 4 presents the performance and memory requirements of EHMA. Conclusions are finally drawn in Section 5.

2 Related Work

This section discusses the main concepts and the limitations of the state-of-the-art exact string matching algorithms that have been used or modified for packet inspection. Some fundamental definitions and notations used in this study are presented.

2.1 Notations

An array is used to represent a *string* of characters from an alphabet set Λ. Namely, an element representing string T at the position i is given by $T[i]$, where $T[i] \in \Lambda$. The absolute value of an object means the size of the object. For instance, $|T|$ denotes the length of the string T, and $|\Lambda|$ is the number of elements in the set Λ. A function $\mathrm{sub}(T, i, B)$ is defined as the substring of T from $T[i]$ to $T[i + B - 1]$. A string can also be denoted as a set of B-grams, where a *gram* is defined as a group of characters, and B is the number of characters in a gram. For instance, the string "green" can be converted into a set of 2-grams {"gr", "re", "ee", "en"} when $B = 2$. The ith B-gram of a string T is represented as $T^B[i]$.

Let $\boldsymbol{P} = \{p_i\}$ be a set of distinct patterns, where p_i denotes a pattern with an identification number (ID) i. The payload of an input packet T and the pattern $p_i \in \boldsymbol{P}$ are both strings drawn over Λ with finite length $|T|$ and $|p_i|$, respectively. The notation $e.f$ denotes the value of the field (or offset) f at the entry (or address) e. If e is a table, then $e.f$ means all fields named f of the table e.

A single-pattern matching algorithm is used to search a string (or text) T for the *first* occurrence or *all* occurrences of *one* given pattern. A multipattern matching algorithm is applied to search the input T for *all* occurrences of *any* pattern $p_i \in \boldsymbol{P}$, or to corroborate that no pattern of $\boldsymbol{P}$ is in T, where the number of patterns is from hundreds to thousands. In other words, the algorithm aims to find *all* the matched patterns in T, say $\boldsymbol{P^M} \subset \boldsymbol{P}$ such that $\boldsymbol{P^M} = \{p_i \mid \forall p_i \subset T \text{ and } p_i \in \boldsymbol{P}\}$. $\boldsymbol{P^M}$ can be applied to any high-level detecting rule, such as the high-priority-win, first-matched-win, or other state-concerned rules.

2.2 Previous Work

Single-pattern matching algorithms were originally proposed to perform text searching problem in computer systems. In single-pattern matching, Boyer-Moore (BM)-based algorithms provide the best average-case performance in terms of computation complexity, which is sublinear to the input string [3], [13]. The BM algorithm uses the *bad character* and *good suffix* heuristics to build a *skip table* and a *shift table*, respectively [13]. The Boyer-Moore-Horspool (BMH) algorithm, which is a variant of BM, slightly modifies the bad character heuristic to construct a single *skip table* [3]. The tables of BM and BMH are precomputed and used to determine the number of safety shifts of *each character* for the searching process. Some characters of T can thus be skipped in the matching process on specific conditions. Several approaches apply the BM-based single-pattern matching algorithms *iteratively* to solve the multipattern matching problem. However, network equipment usually has a large pattern database. Iteratively performing the single-pattern matching for multipattern matching in the packet inspection engine is inefficient. Markatos et al.'s approach promotes Snort by using a bitmap filter before BMH but still searches for only one pattern in each iteration [11].

Several modifications to BM-based algorithms have been proposed for the multipattern matching. Risk and Varghese's (RV) approach groups all patterns to precalculate the number of safety shifts of each character [5]. The WM approach, which assumes that all patterns are larger than M characters, groups B-grams of the M-character prefixes of all patterns to build a *shift table* [10]. The WM's shift table contains the valid shifts of each B-gram. Liu et al.'s algorithm [a variant of the WM algorithm using a grouped prefix hash (WM-PH)] groups the B-character prefixes of all patterns to build a large hash table, in which each entry contains valid shifts of the corresponding B-character prefix [12]. However, the maximum shift value of RV and WM must be not larger than the minimum pattern length in $\boldsymbol{P}$, in order to avoid missing any pattern. Thus, RV and WM are unfeasible when the pattern set includes single-character patterns. The required memory space of the table in the algorithms WM and WM-PH is in the order of $O(|\Lambda|^B)$. Generally, $B = 3$, and the table consists of 16 million entries when the alphabet size is 256 as in 1-byte coding. The large tables must be stored in the external memory, which leads to long access delay during the matching process.

It has been pointed out that the Aho-Corasick (AC) algorithm provides the best worst-case computational time complexity [4]. Using a compressed structure, Tuck et al. proposed the AC algorithm with memory compression (AC-C), a modification of AC, and reduced the required memory to about 2 percent of AC [8]. ACM applied a magic number derived from the Chinese Remainder Theorem to AC [14]. ACM reduced the required memory space and computation complexity, thus improving the worst-case performance. However, the required memory of AC-C and ACM is typically too large to be cached in the on-chip memory of embedded systems, field-programmable gate arrays (FPGAs), and network-processor-based platforms. Although the AC-based algorithms have the best worst-case computational time complexity, the latency of external memory accesses dominates the processing performance rather than computational time. Coit et al. proposed a matching algorithm for Snort that combines BM and AC [15]. However, this algorithm requires three times the memory of the standard version and may produce inconsistent matching results.

A Piranha algorithm was proposed based on an idea that if a *least popular B-gram* of a pattern exists in a packet, then this packet may have a pattern [16]. A least popular gram of a pattern was chosen as an index key of a pattern. However, the Piranha algorithm cannot handle the patterns smaller than B, and the required memory space is very large ($O(|\Lambda|^B)$). Although the idea of least popular index keys can reduce the collisions of patterns, the hit rate of index table is increased, thus increasing the number of external memory accesses and pattern comparisons.

In the case of hardware solutions, Li et al. presented an FPGA-based detection engine for NIDSs, using the internal content addressable memory (CAM) technology to speed up multipattern matching [17]. Since an internal CAM of FPGA is not large enough to store all patterns, Li et al.'s approach has to dynamically reload a block of patterns into the CAM, causing long latency. Moreover, the patterns of varied lengths complicate the formulation of a CAM for exact matching, but Li et al.'s approach does not mention the solution for patterns with varied lengths. Dharmapurikar et al. used Bloom Filters (BFs) and Kim and Kim employed mask filters in the FPGA-based packet inspection [18], [19]. However, these two methods only act as prefilters and have to cooperate with another string matching algorithm to verify a match, and furthermore, this BF-based algorithm can be used only in the case that all patterns are longer than a certain length. Lu et al. used several binary CAMs and BFs to implement parallel compressed deterministic finite automata (DFA), and Dharmapurikar et al. combined AC with BFs for packet inspection [20], [21]. These two methods applied parallel BFs and assumed that BFs can execute one query every clock cycle. However, these architectures and assumptions can only be established in some specific hardware implementations. BFs are inefficient in the software implementations, because one BF consists of several hash functions and the computation time of hash functions is usually expensive in software [6].

3 The Enhanced Hierarchical Multi-Pattern Algorithm

Some network equipment is implemented by network processors, FPGAs, networks-on-chip (NOCs), or systems-on-a-programmable-chip (SOPCs) to improve the performance. The embedded memory of these platforms is typically very small. For instance, the Intel IXP2x00 network processor has only a 4-Kbyte instruction cache and a 2-Kbyte data cache in each microengine, while the Vitesse IQ2000 network processor has a 4-Kbyte data cache (2 Kbytes for local storage and 2 Kbytes for reserved header buffers) [22], [23]. Although high-end FPGAs providing up to 1-Mbyte embedded memory are available, linking many memory blocks degrades the chip performance. Nevertheless, the required memory of the previous pattern matching algorithms is generally larger than 300 Kbytes for NIDSs. Hence, the patterns and the tables built by matching algorithms need to be stored in external memory.

However, frequently accessing the external memory (to read patterns or tables) significantly decreases the matching efficiency due to the external memory access latency being very long and indeterminable. For example, Intel IXP2x00 needs about one cycle for one microprocessor instruction but about 150 cycles for each access from SRAM (or 250-300 cycles from DRAM) [7]. The memory latency strongly affects the throughput of pattern matching. Therefore, reducing the number of required external memory accesses is more important than reducing the amount of computational time.

This study proposes an EHMA based on a hierarchical and cluster-wise architecture. EHMA comprises two small index tables, namely the *first-tier table* (H^1) and the *second-tier table* (H^2). These two tables act as filters to avoid unnecessary external memory accesses and pattern comparisons and, thereby, pass the innocuous packets quickly in the online matching process. The second-tier procedure (*Tier-2 Matching*) activates only after the first-tier procedure (*Tier-1 Matching*) gets a match. Using H^2, which indicates a small subset of patterns that are similar to the input packet, EHMA compares only a few *selected* patterns of $\boldsymbol{P}$ with the *suspected* substrings of the packet, rather than comparing all patterns with all substrings of the packet. Furthermore, a *frequency-based bad gram heuristic* is proposed in the EHMA to determine the safety *shifts* on the input strings during the online matching process. In other words, some characters of the input packets can be safely skipped without any process. External memory accesses are needed only in the Tier-2 Matching state. Consequently, EHMA significantly enhances the matching performance and effectively reduces the number of external memory accesses, string comparisons, and character scans, by utilizing two small index tables.

This study proposes a *general frequent-common gram searching (GFGS)* algorithm and a *cluster balancing strategy (CBS)* to lower the size of the tables H^1 and H^2. The

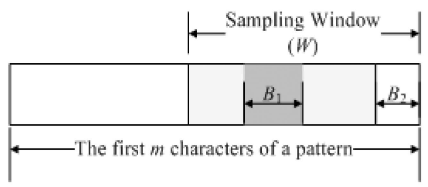


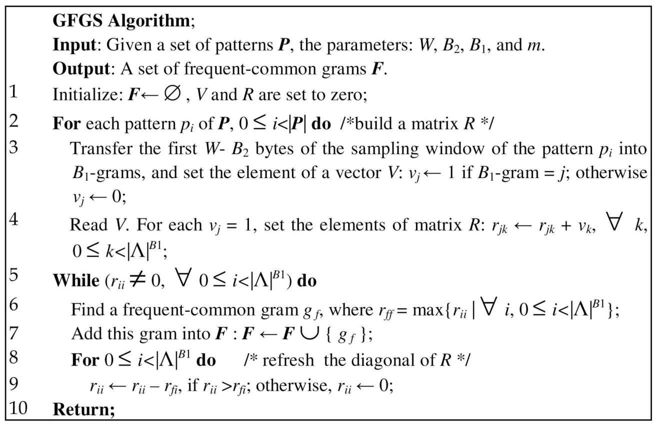
Fig. 1. The sampling window.

following sections describe the GFGS, CBS, and the Safety Shift Strategy in detail. The hierarchical online matching using these two index tables, namely Tier-1 Matching and Tier-2 Matching, is then shown.

3.1 The GFGS Algorithm

In the high-layer intrusion detection, patterns may appear *anywhere* in the packet payload, making the attacking packets difficult to recognize. GFGS assumes that a small set of *signatures* can be found from the patterns themselves, then the suspicious substrings of T may be easier to distinguish from the innocent parts, and the pattern matching is therefore faster. A set of *significant grams* is defined as representatives of a pattern set $\boldsymbol{P}$, given by $\Im \subset \Lambda^{B_1}$, where the size of a gram is B_1 characters. The set $\Im$ is much smaller than Λ^{B_1}. Only when at least a significant gram occurs in the payload, a pattern may exist. That is, when at least one B_1-gram of p_i belonging to $\Im$ occurs in the payload T, the pattern $p_i \in \boldsymbol{P}$ may be found in T. Many innocent B_1-grams of T, which do not belong to $\Im$, can be filtered in the Tier-1 Matching when scanning the packet payload. Obviously, smaller $\Im$ leads to fewer pattern comparisons and, thus, faster pattern matching. The GFGS is proposed to find the smallest $\Im$ from $\boldsymbol{P}$.

Define $\boldsymbol{P}_g$ as a subset of $\boldsymbol{P}$, with $\boldsymbol{P}_g = \{p_i \mid p_i \text{ has the gram } g, \forall\, p_i \in \boldsymbol{P}\}$, where g is called the *common gram* of those patterns in the set $\boldsymbol{P}_g$. Notably, if a common gram appears in the distinct patterns more frequently than other grams and it is selected as one of the significant grams, then a smaller $\Im$ is found. Based on this inference, the GFGS algorithm is designed to find the *frequent-common gram set* $\boldsymbol{F}$, such that $\boldsymbol{F}$ is the minimum set of significant grams to represent a pattern set $\boldsymbol{P}$. In the GFGS, the common grams are searched only from the *sampling window*, which is defined as the last W characters of the first m characters of a pattern. The range of m is $M \le m \le |p_i|$, where M denotes the minimum pattern length of all patterns, and $|p_i|$ is the current pattern length. Fig. 1 illustrates the sampling window, where B_1 is the size of a frequent-common gram, $B_1 \le W$, and B_2 is the size of the second pivot in the H^2 table, which is explained later.



```
   GFGS Algorithm;
   Input: Given a set of patterns $P$, the parameters: $W$, $B_2$, $B_1$, and $m$.
   Output: A set of frequent-common grams $F$.
1  Initialize: $F \leftarrow \varnothing$ , $V$ and $R$ are set to zero;
2  For each pattern $p_i$ of $P$, $0 \le i < |P|$ do  /*build a matrix $R$ */
3     Transfer the first $W$- $B_2$ bytes of the sampling window of the pattern $p_i$ into
      $B_1$-grams, and set the element of a vector $V$: $v_j \leftarrow 1$ if $B_1$-gram = $j$; otherwise
      $v_j \leftarrow 0$;
4     Read $V$. For each $v_j = 1$, set the elements of matrix $R$: $r_{jk} \leftarrow r_{jk} + v_k$, $\forall$ $k$,
      $0 \le k < |\Lambda|^{B1}$;
5  While ($r_{ii} \ne 0$, $\forall$ $0 \le i < |\Lambda|^{B1}$) do
6     Find a frequent-common gram $g_f$, where $r_{ff} = \max\{r_{ii} \mid \forall\ i, 0 \le i < |\Lambda|^{B1}\}$;
7     Add this gram into $F$ : $F \leftarrow F \cup \{ g_f \}$;
8     For $0 \le i < |\Lambda|^{B1}$ do     /* refresh the diagonal of $R$ */
9        $r_{ii} \leftarrow r_{ii} - r_{fi}$, if $r_{ii} > r_{fi}$; otherwise, $r_{ii} \leftarrow 0$;
10 Return;
```

Fig. 2. The GFGS algorithm.

The GFGS algorithm is presented in Fig. 2. A bitmap vector $V = (v_i)$ and a matrix $R = (r_{ij})$ are temporary memories, where $0 \le i,\ j < |\Lambda|^{B_1}$. Vector V records the occurrence of each B_1-gram in a pattern; R is used for recording frequency, where r_{ij}, $i \ne j$, indicates the number of concurrent occurrences of two B_1-grams g_i and g_j in $\boldsymbol{P}$; and r_{ii} records the frequency of the B_1-gram g_i occurring in distinct patterns. For instance, $r_{ij} = 2$ means there are two patterns, each containing both g_i and g_j. In the GFGS algorithm, each pattern is first transferred into a set of B_1-grams, and the occurrence of each B_1-gram is recorded in the bitmap V, where B_1 is predefined and depends on the available on-chip memory space. Matrix R is then derived from V (as shown in line 4 of Fig. 2). Second, the largest occurrence frequency r_{ff} is found, and its corresponding gram g_f is selected as one of $\boldsymbol{F}$. The elements of R relating to g_f are subtracted accordingly to renew R. GFGS is repeated until all elements on the diagonal of R become zero. GFGS uses only a matrix and a vector to discover $\boldsymbol{F}$ from $\boldsymbol{P}$.

Fig. 3 plots the pattern spectrum of the Snort patterns with different gram sizes. The pattern spectrum indicates the occurrence frequency of grams of patterns. Fig. 3a shows the distribution of 2-grams of patterns, and Fig. 3b is the distribution of 1-gram of patterns. As shown in the figures, they are not normally distributed and have several peaks, which mean that some grams obviously occur more

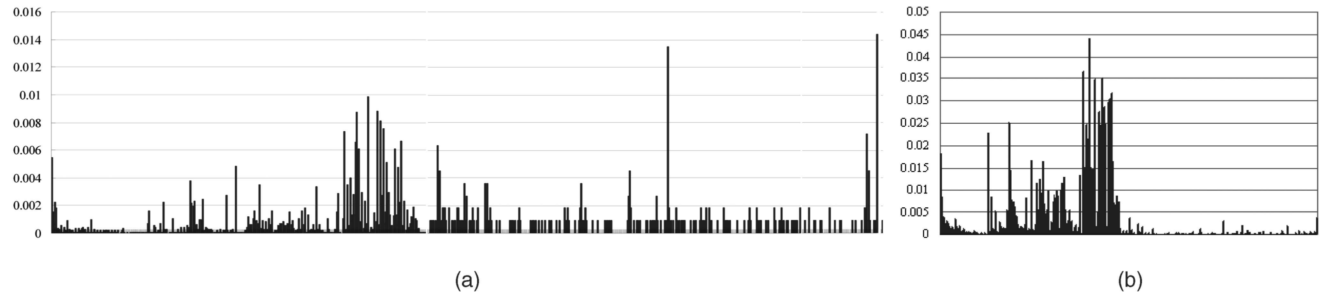


Fig. 3. The pattern spectrum when $|P| = 1,200$. (a) Spectrum of 2-grams. (b) Spectrum of 1-gram.

frequent than others. Hence, GFGS can easily discover the most frequent grams from patterns and obtain a small $\boldsymbol{F}$ as the signatures of patternset. Since both 1-gram and 2-gram spectrums have peaks, the gram size of $\boldsymbol{F}$ can be one or two, depending on the available size of on-chip cache.

3.2 Cluster Balancing Strategy (CBS)

Most packets are innocent in general situations. Even a harmful packet may contain only few patterns. Therefore, comparing all of the patterns in the large $\boldsymbol{P}$ with each input packet is time consuming. If the patterns in $\boldsymbol{P}$ can be distributed into different small *clusters* based on their similarity, then only the pattern in each cluster that is most similar to the suspected packet needs to be compared, thus improving the efficiency of the matching process. This section presents strategies to attain this goal. First, the method of clustering a set $\boldsymbol{P}$ based on the similarity of patterns is described. Then, a CBS is adopted to balance the cluster size. A *second-tier table* (H^2) for online matching can be constructed based on the clusters.

The *clustering pivots* are the keys used to distribute patterns, where each clustering pivot is a common gram of patterns defined previously. Two common grams are employed as a pair of clustering pivots, called a *pivot pair*, say (a, b), where the first pivot is a frequent-common gram, and the second pivot is the substring following the frequent-common gram. Let $\boldsymbol{P}_{a,b}$ represent a cluster of selected patterns (a subset of patterns) with the pivot pair (a, b), which means that $\boldsymbol{P}_{a,b} = \{p_i \mid \text{`}ab\text{'} \subset p_i, a \in \boldsymbol{F} \text{ and } b \in \Lambda^{B_2}\}$, where '$ab$' is the combination of two strings a and b and is a substring of p_i; $\boldsymbol{F}$ is the result of GFGS, and B_2 is the length of the second pivot. Notably, a pattern is assigned to only one cluster in the clustering strategy, although a pattern may have more than one pivot pair. That is, the clusters have the following properties: for any cluster $\boldsymbol{P}_{a,b} \subset \boldsymbol{P}$, $\cup_{\text{all } a,b}\boldsymbol{P}_{a,b} = \boldsymbol{P}$ and $\cap_{\text{all } a,b}\boldsymbol{P}_{a,b} = \emptyset$. Since a pattern may have several opportunities to select a cluster, a better assignment can lower the maximum cluster size and, thereby, improve the worst-case performance of EHMA.

The pattern grouping is based on $\boldsymbol{F}$. To lower the worst matching time, CBS is adopted to balance the size of all clusters. In CBS, an $|F| \times |\Lambda|^{B_2}$ matrix $N = (n_{a,b})$ is used to record the current size of every cluster $\boldsymbol{P}_{a,b}$ during the pattern grouping procedure. The CBS is given as follows:

1. First, read one pattern at a time from $\boldsymbol{P}$ and scan the pattern.
2. According to GFGS, for any given p_i, there exists a B_1-gram $g \in \boldsymbol{F}$, where B_1 is the length of a frequent-common gram. To balance the cluster size, CBS finds the smallest $n_{a,b}$, given by $n_{x,y}$, among all available pivot pairs (a, b)'s of p_i, for all $a \in \boldsymbol{F}$ and 'ab' $\subset p_i$.
3. After grouping p_i into the smallest cluster $\boldsymbol{P}_{x,y}$, the corresponding $n_{x,y}$ is also incremented.

All patterns are distributed sequentially into the designate clusters. Accordingly, GFGS and CBS divide the large $\boldsymbol{P}$ into smaller subsets.

3.3 Safety Shift Strategy

This section presents a safety shift strategy to derive the values of the *shift* fields of H^1 and H^2. H^1 and H^2 can use the same strategy to derive their safety shifts, respectively. As mentioned previously, as long as no frequent-common gram is matched in input strings, then no pattern exists. Therefore, if no frequent-common gram is missed, then no pattern will be missed. The safety shift strategy is based on a modified *bad grouped character heuristic* [7], named *frequency-based bad gram heuristic* in this study. The safety shift strategy ensures that no frequent-common gram is missed during a skippable scanning process. The proposed strategy helps EHMA to speed up the online matching process, since certain characters can be skipped unhesitatingly.

Assume that x identifies all possible index keys and that the length of x is B. Because the index keys of H^1 and H^2 are different, the parameters used to determine the *shift* fields of these two tables are different. For H^1, as the length of a frequent-common gram is B_1, thus $x \in \Lambda^{B_1}$ and $B = B_1$. For H^2, since x is all the possible of the pivot pairs (a, b), $x \in \boldsymbol{F} \times \Lambda^{B_2}$ and $B = B_1 + B_2$. The basic concept of the safety shift strategy is that: if x is not a gram of any pattern, and any suffix of x is not any prefix of any pattern in $\boldsymbol{P}$, then it is safe to shift m when x is scanned; otherwise, the number of safety shifts is the offset between the rightmost occurrence position of x and the position of the frequent-common gram nearest to x. Two parameters are needed to derive the safety shifts, namely W and m, as shown in Fig. 1. Assume that $B \le W \le m$, and define the safety shifts of each entry $(H(x).shift)$ as follows:

1. Initially, all *shift* fields of the table H are set as

   If $m > W$, then
   $$H(x).shift = m - W + q,$$
   where $q = \min\{q \mid \exists\, \text{sub}(x, q+1, B-q) = \text{sub}(p, 1, B-q), \forall p \in \boldsymbol{P} \text{ and } 1 \le q < B\}$ when $B > 1$ and q exists; otherwise, $q = B$.

   Else
   $$H(x).shift = r,$$
   where $r = \min\{r \mid \exists\, \text{sub}(x, r+1, B-r) = \text{sub}(f, 1, B-r), \forall f \in \boldsymbol{F}, 1 \le r < B, \text{ and } r + B < W\}$ when $B > 1$ and r exists; otherwise, $r = B$.

2. Scanning every pattern p, for each ith B-gram of each pattern $p^B[i]$, where $1 \le i \le m - W$, set $x \leftarrow p^B[i]$ if the entry $H(x)$ exists:

   If the current $H(x).shift > m - W - i + 1$, then update the entry, so that
   $$H(x).shift = m - W - i + 1.$$

3. For each ith B-gram of each pattern $p^B[i]$, where $m - W < i \le m - B + 1$, set $x \leftarrow p^B[i]$ if the entry $H(x)$ exists:

   If $x \in \boldsymbol{F}$, then
   $$H(x).shift = 0;$$
   Else If the current $H(x).shift > r$, then update the entry:
   $$H(x).shift = r,$$
   where $r = \min\{r \mid \exists\, \text{sub}(x, r+1, B-r) = \text{sub}(f, 1, B-r), \forall f \in \boldsymbol{F}, 1 \le r < B, \text{ and } r+B<W\}$ when $B > 1$ and r exists; otherwise, $r = B$.

Notably, the maximum shift of EHMA is m while $W = B$. The frequent-common grams and the sampling window are introduced in the proposed frequency-based bad gram heuristic to improve the flexibility and the efficiency. Additionally, comparing EHMA with WM, the maximum safety shift is raised from $m - B + 1$ to m. The shift value of the proposed EHMA is similar to but larger than the shift value of WM, when the given parameters are $m = M$ and $W = B$.

3.4 Table Construction

The result of GFGS, $\boldsymbol{F}$, is used to construct the small table H^1, which is stored in the on-chip memory. A direct index table of $|\Lambda|^{B_1}$ entries is used for H^1 to achieve fast lookup. B_1 is usually very small ($B_1 = 1$ or 2) and is predefined according to the available size of on-chip memory. An entry of H^1 is denoted as $H^1(a)$, where a is a B_1-gram, and each entry has three fields: the frequent-common gram ID, $H^1(a).fid$; the pattern ID when a itself is a pattern, $H^1(a).pid$, and the safety shift number in the Tier-1 Matching, $H^1(a).shift$. Namely, $H^1(a).fid = \{i \mid a = fi \in \boldsymbol{F}\}$, and $H^1(a).pid = \{i \mid |p_i| = |f_i| = B_1, p_i = \text{'}a\text{'} \text{ and } p_i \in \boldsymbol{P}\}$. The unused fields of H^1 are set to NULL. Since H^1 is a small table (for instance, 256 entries in the case of 1-byte coding and $B_1 = 1$), it can be stored in the on-chip cache. Later, H^1 acts as a filter in the online matching to quickly discover whether the packet contains a pattern. Namely, EHMA employs H^1 to quickly scan and jump over the innocent substrings of the input packets and to narrow the searching field to the most likely clusters.

The H^2 table is built based on the cluster assignments. H^2 contains the pattern contents and formatted information of patterns for fast online matching. Let $H^2(a, b)$ denote an entry of H^2, indicating the head pattern of the cluster $\boldsymbol{P}_{a,b}$, and defined as

$$H^2(a, b) = H^1(a).fid \times |\Lambda|^{B_2} + b,$$

where B_2 is the length of the second pivot b and is predefined according to the available size of the external memory. Each entry $H^2(a, b)$ consists of six fields: the safety shift number in the Tier-2 Matching $H^2(a, b).shift$, the position of the frequent-common gram in the pattern $H^2(a, b).offset$, the pattern size $H^2(a, b).size$, the pattern content $H^2(a, b).data$, the pattern ID $H^2(a, b).pid$, and a pointer $H^2(a, b).next$ to the entry of the next pattern in the same cluster $\boldsymbol{P}_{a,b}$ or the fragmented content of the current pattern. Transferring the information of patterns into a predefined format can accelerate the matching procedure. The patterns in the same cluster $\boldsymbol{P}_{a,b}$ point to the same head entry $H^2(a, b)$ and are linked by the linked list structure to optimize the memory usage. The required memory size of H^2 is $|F| \times |\Lambda|^{B_2}$ entries plus the shared memory pool.

For example, if p_i is clustered to $\boldsymbol{P}_{a,b}$ by CBS and $H^2(a, b)$ is empty, then the information of pattern p_i is saved into $H^2(a, b)$, where $H^2(a, b).size = |p_i|$, $H^2(a, b).data = p_i$, and $H^2(a, b).offset = k$ if the kth B_1-gram of p_i is a, $H^2(a, b).pid = i$, and $H^2(a, b).next$ is NULL. If another p_j is also clustered to $\boldsymbol{P}_{a,b}$, then a free entry is also assigned to p_j and linked with the previous pattern p_i. Similarly, if the pattern size of p_i is larger than the width of data field, then

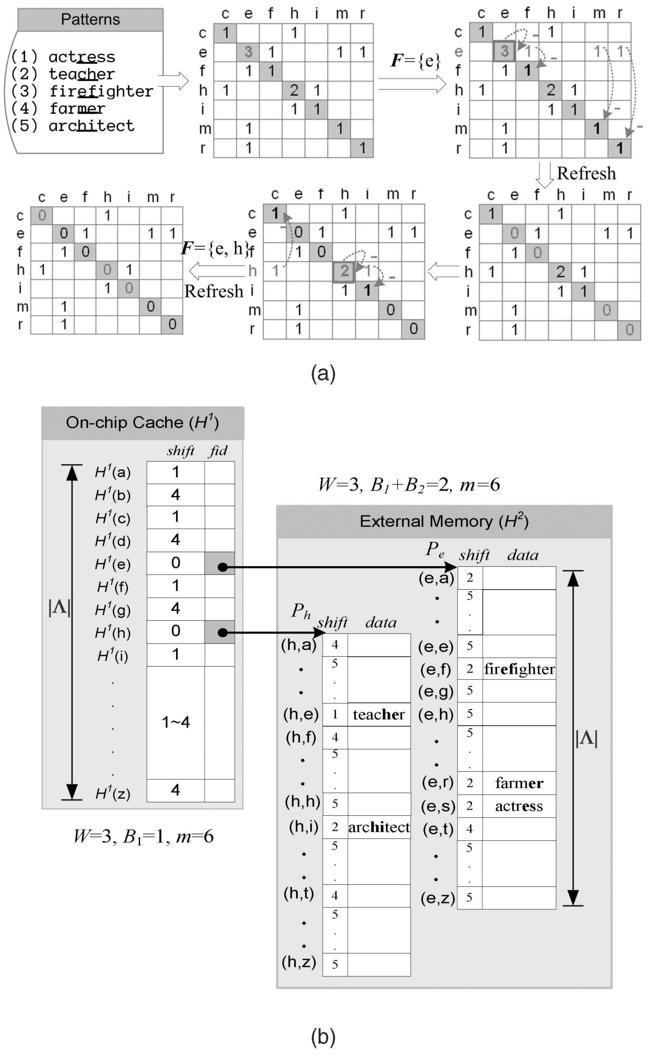


Fig. 4. An example of EHMA, where $B_1 = 1$, $B_2 = 1$, $m = M = 6$, $W = 3$, and $\boldsymbol{F} = \{e, h\}$. (a) An example of GFGS. (b) The architecture of the hierarchical hash tables.

p_i is fragmented, and the remaining part is saved in a free entry of the shared memory pool, and the address is saved in $H^2(a, b).next$.

Fig. 4 shows an example of EHMA, which has five patterns: "actress," "teacher," "firefighter," "farmer," and "architect," where the alphabet set comprises the 26 English letters. The parameters for EHMA are assumed $B_1 = 1$, $B_2 = 1$, $m = 6$, and $W = 3$. Fig. 4a demonstrates the GFGS. According to the GFGS (lines 2-4 of Fig. 2), after scanning the first $W - B_2$ characters of the sampling window of every pattern (the underlined characters of the patterns in Fig. 4a), the matrix R is obtained and shown in the figure. In the first run, the maximum value on the diagonal of R is three, and thus the corresponding gram "e" is added into $\boldsymbol{F}$. After refreshing the elements on the diagonal of R (lines 8 and 9 of Fig. 2), GFGS finds that the maximum value on the diagonal of R is two in the second run, and the corresponding gram is "h." GFGS stops while all elements on the diagonal of R are zero, and gets $\boldsymbol{F} = \{\text{'e'}, \text{'h'}\}$. Fig. 4b displays the logical architecture of the two-tier tables of EHMA. Because $B_1 = 1$, and the H^1 table has only 26 entries, the H^1 table can be stored in the cache memory. The *fid* fields of H^1 point to the corresponding offsets of H^2. As the pattern "actress" has 'e' $\in \boldsymbol{F}$ and the pivot pair "es," according to CBS it is grouped to the cluster $\boldsymbol{P}_{e,s}$. The *shift* fields of H^1 and H^2 are obtained from the proposed safety shift strategy. Initially, since $B_1 \leq 1$, $H^1.shift = 4$. While $B_1 + B_2 > 1$, $H^2.shift$ is set to 5 for those entries whose second pivot is not the prefix of any pattern (that is, $b \notin \{\text{'a'}, \text{'f'}, \text{'t'}\}$); otherwise, $H^2.shift$ is set to 4. When scanning the pattern "actress," the *shift* fields of $H^1(\text{'a'})$, $H^1(\text{'c'})$, and $H^1(\text{'t'})$ are updated to 3, 2, and 1, respectively (the second safety shift strategy); the *shift* fields of $H^1(\text{'r'})$ and $H^1(\text{'s'})$ are both updated to 1, while the $H^1(\text{'e'}).shift$ is updated to 0, because 'e' $\in \boldsymbol{F}$ (the third strategy). As for the table H^2, only the existing entry $H^2(\text{'e'}, \text{'s'})$ has to be updated to 2, because $B = B_1 + B_2 = 2$, and no prefix of $\boldsymbol{F}$ is the suffix of "es" (the third strategy). The remainders of the patterns follow the same clustering and safety shift strategy. The *shift* fields of H^1 and H^2 tables are updated when the new *shift* is less than the previous one. Let us see $H^1(\text{'a'})$ for example. When scanning the pattern "actress," $H^1(\text{'a'}).shift = 3$ (as $p^1[i] = \text{'a'}$, $i = 1$ and $m - W - i + 1 = 3$); while scanning the pattern "teacher," $H^1(\text{'a'}).shift$ is updated to 1 (as "a" is the third character of "teacher": $i = 3$, then $m - W - i + 1 = 1$), because the new value is smaller than the previous one (the second strategy). Finally, $H^1(\text{'a'}).shift = 1$ is saved in the table because the remaining patterns do not have $H^1(\text{'a'}).shift$ smaller than one. Notably, the maximum shift of H^1 and H^2 is large (4 and 5, respectively). Consequently, the number of scans and comparisons can be significantly reduced.

3.5 The Online Hierarchical and Cluster-Wise Matching

The previous sections presented the offline stage of EHMA, which builds two index tables H^1 and H^2, holding the indexing and pattern information in the cache memory and external memory, respectively. These two tables are regarded as the two-tier filters and indices for the online matching. This section presents the online matching procedure in detail.

In network intrusion detection systems, an input packet is forwarded to a detection engine. The detection engine then returns the search results of matched patterns $\boldsymbol{P}^M$. This study focuses on the payload inspection and assumes that each input is a packet payload T. As a hierarchical matching, the online matching procedure of EHMA is divided into two tiers: Tier-1 Matching and Tier-2 Matching. The hierarchical architecture is applied to decrease the number of external memory accesses. The small H^1 is stored in the cache of the processing unit for Tier-1 Matching, while the H^2 with pattern content is in the external memory for Tier-2 Matching. The external memory access is necessary only when the Tier-2 Matching is invoked. This process is described in detail in the following sections.

3.5.1 Tier-1 Matching

In online matching, the payload T is scanned from left to right, and each B_1-gram of T is the key to fetch the entry $H^1(t_1)$, where $t_1 = T^{B_1}[i]$. The H^1 acts as the first-tier filter of EHMA, by checking whether T may likely contain patterns belonging the pattern set $\boldsymbol{P}$. Because H^1 is small enough to be stored in the on-chip memory during the online matching procedure, the latency of accessing H^1 is very small.

In the Tier-1 Matching, first the *shift* field is checked. If $H^1(t_1).shift \neq 0$, i.e., $t_1 \notin \boldsymbol{F}$, then no external memory is necessary. The obtained $H^1(t_1).shift$ also determines the number of grams that can be skipped without further process. The next gram to check is then $T^{B_1}[i + H^1(t_1).shift]$. After reading the next gram, the matching process repeats as in the previous steps and remains in the Tier-1 Matching. Because $|\boldsymbol{F}| \ll |\Lambda|^{B_1}$, the probability of $t_1 \in \boldsymbol{F}$ is small and most grams of T gain the shifts, thus avoiding the Tier-2 Matching. Consequently, both the number of string comparisons and the costly memory accesses can be significantly reduced.

Otherwise, if $t_1 \in \boldsymbol{F}$, then T may contain a malicious pattern $p_k \in \boldsymbol{P}$, where $t_1 \subset p_k$. Simply stated, if $H^1(t_1).shift = 0$, then T may have a pattern that belongs to the cluster of pivot pair (t_1, t_2), where $t_2 = T^{B_2}[i + B_1]$. Therefore, the matching procedure activates Tier-2 Matching to identify the pattern. If $H^1(t_1).pid$ is not NULL, then the current gram t_1 itself is a pattern, and this matched pattern is also added into $\boldsymbol{P}^M$.

3.5.2 Tier-2 Matching

After the Tier-1 Matching, if $H^1(t_1).shift = 0$, then the matching procedure proceeds to the Tier-2 Matching. The function $H^2(t_1, t_2)$ indicates the location of the corresponding cluster according to input T. Since EHMA is a cluster-wise matching algorithm, only the patterns in the small cluster of pivot pair (t_1, t_2), which are similar to T, are loaded to the processing unit for further checks.

Tier-2 Matching first checks the *pid* field of H^2. If $H^2(t_1, t_2).pid$ is NULL, then the cluster (t_1, t_2) contains no pattern, and no pattern comparison is necessary. Otherwise, if $H^2(t_1, t_2).pid$ is not NULL, then this cluster contains patterns. The pattern content in the $H^2(t_1, t_2).data$ is then compared with the corresponding substring of T: $\mathrm{sub}(T, i - H^2(t_1, t_2).offset, H^2(t_1, t_2).size)$. If $H^2(t_1, t_2).next$ is valid and points to the next entry, here given by $H^2(a, b)$, then the cluster contains other patterns. Similarly, the pattern in $H^2(a, b).data$ is also fetched and compared with the substring of T starting at $T[i - H^2(a, b).offset]$ of length $H^2(a, b).size$. Every matched pattern is added to the matched pattern set $\boldsymbol{P}^M$ and its corresponding matched *pid* set $\boldsymbol{PID}^M$ in order. Until all patterns in this cluster are checked, the next gram $T^{B_1}[i + H^2(t_1, t_2).shift]$ is then read, and the online matching procedure returns to the Tier-1 Matching. $H^2(t_1, t_2).shift$ also indicates the number of characters of T that can be skipped, since the next possible frequent-common gram may only appear far away than $H^2(t_1, t_2).shift$.

Notably, if a pattern p_k exists in T, then all grams of p_k appear in T. The clustering pivot pair of pattern p_k $(p_k^{B_1}[j], p_k^{B_2}[j + B_1])$ is certainly scanned, say at t_1 and t_2, so that $t_1 = p_k^{B_1}[j] \in \boldsymbol{F}$ and $t_2 = p_k^{B_2}[j + B_1]$. Pattern p_k is then recognized when T is compared with the patterns in the cluster (t_1, t_2) during the online matching procedure. Based on the Safety Shift Strategy, EHMA never skips any frequent-common gram. Consequently, no patterns in the payload T are missed.

```
   Procedure Tier-1Matching($T$, $H^1$, $M$, $W$, $B_1$)
   Input: Packet payload $T$, a first-tier hash table: $H^1$, the minimum pattern length
   $M$, the length of the frequent-common gram $B$ and the length of the sampling
   window $W$.
   Output: The output of Tier-2Matching.
 1 $i \leftarrow M-W+1$;
 2 While $i <= |T| - B_1$ do
 3   Read the $i$-th $B_1$-gram of $T$: $gram \leftarrow T^{B1}[i]$;
 4   If $H^1(gram).shift > 0$, then $shift \leftarrow H^1(gram).shift$;
 5   Else
 6     If $H^1(gram).fid \neq$ NULL, then $shif \leftarrow$ Tier-2Matching($T$, $H^2$, $B_2$, $i$);
 7     If $H^1(gram).pid \neq$ NULL, then
         $P^M \leftarrow P^M \cup \{gram\}$;
       If $shift == 0$, then $shift \leftarrow 1$;
 8   Jump over the string: $i \leftarrow i+shif$;     /*shift and read the next*/
 9 End While
10 Return;

   Procedure Tier-2Matching($T$, $H^2$, $B_2$, $i$)
   Input: Packet payload $T$, a preprocessed indexing table: $H^2$, the length of the
   second pivot $B_2$, and the current pointer $i$
   Output: A safety shift number for Tier-1 Matching: $shift$, the matched pattern
   set of $T$: $P^M$, and its corresponding $pid$ $PID^M$
 1 Load data from the external RAM at entry $H^2(T^{B1}[i], T^{B2}[i+B_1])$ to a local
   buffer $LB$;
 2   $shift \leftarrow LB.shift$;
 3 While ($k \leftarrow LB.pid) \neq$ NULL do
 4   Compare the substring of $T$: sub($T$, $i$-$LB.offset$, $LB.size$) with the pattern
     $LB.data$;  /*Assume no fragmentation here*/
 5   If it is matched then $P^M \leftarrow P^M \cup \{p_k\}$ and $PID^M \leftarrow PID^M \cup \{k\}$;
 6   If $LB.next \neq$ NULL then
 7     Load data from the external RAM at entry $LB.next$ to the local buffer $LB$;
 8   Else
 9     Jump to Line 10;
10 End While
11 Return $shift$;
```

Fig. 5. The online matching procedure, including Tier-1 Matching and Tier-2 Matching.

The online matching procedure of EHMA is described in Fig. 5, including Tier-1 Matching and Tier-2 Matching. Since EHMA introduces H^1 and H^2 as filters, and CBS is employed, only a few suspected patterns are loaded from external memory and compared with T. Because generally most of the packets are innocent over the network, and the frequent-common grams ($\boldsymbol{F}$) narrow the searching field, EHMA performs a fast scan over the packets. The returned result $\boldsymbol{P}^M$ includes all matched patterns for a given T and is applied to make the final decision and analyze the impending attacks. The final decision depends on decision-making rules.

An example is provided to demonstrate the online matching of EHMA. Assume that the H^1 and H^2 tables have been built as Fig. 4, where $W = 3$ and $M = 6$. Assume that the input T is "kangaroo" as given in Fig. 6. The scan runs from left to right. The scan starts at "g" ($(M - W + 1)$th gram), obtaining $H^1(\text{'g'}).shift = 4$. Therefore, Tier-1 Matching shifts four characters. Because the pointer goes beyond $|T| - B_1$ after the shift, EHMA completes scanning the input T. This example only requires one on-cache table lookup and no external memory access. By only checking T with the embedded table H^1, EHMA can know that T contains no pattern.

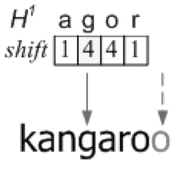


Fig. 6. An example of matching process with input "kangaroo."

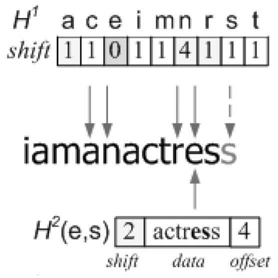


Fig. 7. An example of matching process with input "iamanactress."

Considering another example where $T = \text{'iamanactress'}$ as shown in Fig. 7, the first scanned B_1-gram is "a," yielding $H^1(\text{'a'}).shift = 1$. Thus, the matching process stays in the Tier-1 Matching, and the next B_1-gram "n" is read after shifting one character, yielding $H^1(\text{'n'}).shift = 4$. Similarly, staying in the Tier-1 Matching, and the next B_1-gram "n" is read after shifting one character, yielding $H^1(\text{'n'}).shift = 4$. Similarly, staying in the Tier-1 Matching, the matching process obtains $H^1(\text{'r'}).shift = 1$ and $H^1(\text{'e'}).shift = 0$ in order after shifting. While $H^1(\text{'e'}).shift = 0$, the Tier-2 Matching is activated. After checking the field $H^2(\text{'e'}, \text{'s'}).pid$ and finding that it is not NULL, EHMA knows a suspected pattern may exist. The Tier-2 Matching then compares input T with the pattern in the cluster $\boldsymbol{P}_{e,s}$, where $H^2(\text{'e'}, \text{'s'}).data = \text{'actress'}$, and gets a match. Because this cluster contains no other patterns, the matching process returns to Tier-1 Matching with $H^2(\text{'e'}, \text{'s'}).shift = 2$. Since the pointer goes beyond $|T| - B_1$ after shifting two characters, the matching process for the input T is finished. In this case, H^1 is checked four times, and H^2 is fetched only once for the string T of 12 characters. EHMA thus significantly reduces the latency caused by memory accesses.

3.6 Incremental Update

EHMA can achieve incremental update by adding a *count* field in the H^2, which records the current size of every cluster. The *count* field has the same function as the matrix N of CBS. When a pattern p is added into $\boldsymbol{P}$, after checking the *count* fields of the possible entries according to the pivot pairs of p, the smallest cluster, say $\boldsymbol{P}_{x,y}$, can be found. Then, p is added into the cluster $\boldsymbol{P}_{x,y}$ by following the steps of the table construction mentioned previously. If no B_1-gram of p belongs to $\boldsymbol{F}$ and p finds no existing entry in the H^2, then a random B_1-gram of p, say g, is chosen and added into $\boldsymbol{F}$ ($H^1(g)$ is modified accordingly), and a memory space is allocated for cluster set $\boldsymbol{P}_g$ in H^2. A random pivot pair of p, say (g, h), is chosen and then p is added into the cluster $\boldsymbol{P}_{g,h}$. The *shift* fields of H^1 and H^2 may be modified because of the added p. Since the safety shift strategy scans the patterns *one by one* to calculate the *shift* values, no modification to the safety shift strategy is required for pattern addition. The added p can be recognized as the last scanned pattern of the safety shift strategy. At most $|p| - B_1 + 1$ fields of H^1 and $|p| - B_2 + 1$ fields of H^2 are modified for a pattern addition.

To delete a pattern p from $\boldsymbol{P}$, the first step is to find the pattern. When p is found, just link p's previous entry to p's next entry by modifying its *next* field in H^2 and delete p from tables. Then, subtract the *count* field of the cluster that p belongs to. The shift fields are not modified for pattern deletion. Because the *shift* values are universal minimum in the safety shift strategy, they may not be optimum after pattern deletion. However, no error will occur after pattern deletion, even while the *shift* fields are not modified. Consequently, EHMA needs not recalculate the whole index tables as long as the pattern database is changed. EHMA can refresh the index tables when the system is not busy.

3.7 Worst Case

If a given string T, which has to do the exact string comparisons the most times, is formed badly and no character of T can be skipped during the online matching process, processing this badly formed T is the worst case of EHMA. Assume the largest cluster size is L_c. When every character $T[t] \in \boldsymbol{F}$, $H^1(T[t]).shift = 0$, and each corresponding indexed cluster is the largest ($|\boldsymbol{P}_{T[t],T[t+1]}| = L_c$), T is a badly formed string and this is the worst scenario of EHMA. As for all $T[t]$, $T[t] \in \boldsymbol{F}$ and $H^1(T[t]).shift = 0$, the probability to fetch the table H^2 for the badly formed T is one. Thus, the number of external memory accesses per character in the worst case is

$$N_{RAM}^{WST} = \frac{(|T| - B_2) \times L_c}{|T|} < L_c,$$

assuming that fetching one pattern needs one memory access. Define the largest pattern size in $\boldsymbol{P}$ as L_p. When every input character points to the largest cluster, in which every pattern has the longest size, this badly formed T requires the largest number of comparisons. Hence, the number of character comparisons per input character is

$$N_{CMP}^{WST} = N_{RAM}^{WST} \times L_p < L_c \times L_p.$$

Obviously, the worst-case performance depends on L_c. To derive L_c, assume there is a largest cluster, say $\boldsymbol{P}_{x,y}$. Since $\boldsymbol{P}_{x,y}$ is the largest cluster, assume that the cluster size is always larger than one, and initially, the probability that its cluster size increases from 0 to 1 is one.

As $\boldsymbol{P}_{x,y}$ is the largest cluster, based on CBS, a given pattern p will not be clustered into $\boldsymbol{P}_{x,y}$, unless all available pivot pairs of p are not in the set $\boldsymbol{F} \times \Lambda$ except (x, y). Since the pattern database is usually predefined and static, assume the given patterns are uniformly distributed. Therefore, the probability that $|\boldsymbol{P}_{x,y}|$ increases from i to $i+1$ is

$$\Pr\{|P_{x,y}| = i \rightarrow i+1\} = \left(\frac{|\Lambda|^2 - |F| \times |\Lambda| + 1}{|\Lambda|^2}\right)^{|p|-B_2-1}.$$

As in the worst-case scenario, every pattern has the longest size L_p, the equation is rewritten as

$$\Pr\{|P_{x,y}| = i \rightarrow i+1\} = \left(\frac{|\Lambda|^2 - |F| \times |\Lambda| + 1}{|\Lambda|^2}\right)^{|L_p|-B_2-1}.$$

Thereby, the probability that the cluster size of $\boldsymbol{P}_{x,y}$ is maximum (L_c) is derived as follows:

$$\Pr\{|P_{x,y}| = L_c\} = \left(\frac{|\Lambda|^2 - |F| \times |\Lambda| + 1}{|\Lambda|^2}\right)^{(|L_p|-B_2-1)(L_c-1)}.$$

TABLE 1
The Simulation Parameters

Items	Value
Time of one RISC instruction or one local memory access (w_I)	1 cycle
Latency for each external memory access (w_E)	10, 100 cycles
Packet payload length for Model I	512 bytes
Number of patterns in P ($\|\boldsymbol{P}\|$)	200, 400,…,5000
Simulation time for Model I	10 million packets

When $|\boldsymbol{P}|$ is 1,200 with $|\boldsymbol{F}| = 77$, $|\Lambda| = 256$, and $L_p = 128$, the probability that $L_c = 4$ is only 7×10^{-79}. When replacing L_p with the average pattern size, which is about 11 in the Snort, then the probability that $L_c = 4$ is about 3.6×10^{-6}. The probability that $L_c = 4$ is very small, which infers that EHMA has a small L_c, and thus N_{RAM}^{WST} and N_{CMP}^{WST} are small. Consequently, the worst-case performance of EHMA is moderate and acceptable because L_c is much smaller than $|\boldsymbol{P}|$.

4 Results

As the number of network security threats rises, the NIDS has become one of the most important applications of packet inspection [24], [25]. Therefore, this study demonstrates the feasibility of integrating the proposed EHMA with the promising NIDS. This section presents the simulation results of EHMA deployed in the NIDS, compared with the original HMA [9], BMH algorithm [3], WM algorithm [10], WM-PH [12], and AC-C [8]. In the simulations, the assembly-like microprograms were emulated for EHMA, BMH, WM, WM-PH, and AC-C using RISC instructions of general network processors (such as ADD, XOR, MOV), and the number of instructions and the number of memory accesses needed to process a packet were calculated. To simplify the evaluation, the simulation assumed that one microprocessor was employed.

4.1 Measurements

Define I as the average number of RISC instructions (including comparisons and calculations) and L as the average number of local memory accesses (including reading data from the cache to the registers for further processes), for each payload character in the pattern matching. E represents the average number of external memory accesses per input character, which includes loading the input packets, querying the entries of tables in the external memory, and fetching the patterns. w_I indicates the time needed by one instruction or one local memory/register access, and w_E indicates the time for one external memory access. The following measurements are given: the average computation cycles $\psi_I = I \times w_I$; the average memory latency $\psi_M = E \times w_E + L \times w_I$; and the total average matching time $\Psi = \psi_I + \psi_M$, which is regarded as the overall performance.

In the simulations, the skip table of BMH was assumed to be small enough to be loaded into the cache memory, and therefore, only one external memory access was counted during the matching process of BMH for each pattern. One external memory access was assumed for AC-C, although it typically needs two memory references to fetch the transition matrices, and the fail table or the matched patterns. Table 1 lists the simulation parameters.

TABLE 2
The Pattern Size Distribution of Snort Rule Set R_1

Pattern Size	=1	≤ 4	≤ 8	≤ 12	≤ 16	>16
Ratio	0.028	0.245	0.482	0.653	0.813	0.187

4.2 Traffic Models

The simulations used two free and real pattern sets, R_1 and R_2, from Snort in August 2004 and May 2008, respectively [1], although the pattern set can be self-defined or any commercial pattern set. The number of *distinct* patterns is about 1,250 in the R_1, where the average length of a pattern is about 11.2 bytes (the statistics of the pattern set listed in Table 2); while the number of distinct patterns becomes up to about 5,000 in the R_2. Since Snort patterns are written in mixed plain text and hex formatted bytecodes, the alphabet size ($|\Lambda|$) was set to 256 in the simulations. In the simulation traffic models, Models I and II use R_1, and Model III uses R_2 as the matching pattern sets.

Table 3 shows the relationships between the number of patterns $|\boldsymbol{P}|$ and the number of frequent-common grams $|\boldsymbol{F}|$ of the EHMA, where the lengths of patterns are in the range from 1 to 122, $m = |p_i|$, and the patterns are randomly selected from R_1. The results in Table 3 reveal that the growth rate of $|\boldsymbol{F}|$ is much slower than that of $|\boldsymbol{P}|$.

4.2.1 Model I

In Model I, the synthetic malicious packets are generated by randomly choosing patterns from the pattern set $\boldsymbol{P}$ and spreading over the packet payloads. The attack load λ is defined to represent the expected number of malicious patterns existing in one packet. For instance, if $\lambda = 2$, then each packet contains two harmful patterns on average. Except for the injected patterns parameterized by λ, the background characters of a packet were randomly drawn from Λ to imitate the normal packet content. Hence, the random background may unconsciously contain patterns.

4.2.2 Model II

To evaluate the performance of algorithms in a real intense attack, a trace from the Capture-the-Flag contest held at Defcon9 was adopted as the input traffic in Model II. The Defcon Capture-the-Flag contest is the largest security hacking game, in which competitors try to break into the servers of others while protecting their own servers, each hiding several security holes [26].

4.2.3 Model III

Model III uses a real 2-hour trace as the input traffic, and the more recent Snort rules R_2 as the pattern set $|\boldsymbol{P}|$. This real trace recorded all IP packets in a laboratory of Providence University for 2 hours. The laboratory has an FTP server, a Web server, and three PCs running several network application clients.

TABLE 3
The Number of Frequent-Common Grams
versus the Pattern Set Size

$\lvert\boldsymbol{P}\rvert$	100	200	300	400	500	600	700	800	900	1000	1100	1200
$\lvert\boldsymbol{F}\rvert$	11	28	32	37	45	49	52	58	66	74	75	77

TABLE 4
The Memory Requirements

	EHMA	HMA	WM	WM-PH	AC-C	BMH	BMH-O
Cache Memory	$O(\lvert\Lambda\rvert)$	$O(\lvert\Lambda\rvert)$	$O(1)$	$O(1)$	$O(1)$	$O(\lvert\Lambda\rvert)$	$O(1)$
External Memory	$O(\lvert\boldsymbol{F}\rvert \times \lvert\Lambda\rvert)$	$O(\lvert\boldsymbol{F}\rvert \times \lvert\Lambda\rvert)$	$O(\lvert\Lambda\rvert^3)$	$O(\lvert\Lambda\rvert^3)$	$O(S)$	$O(\lvert\boldsymbol{P}\rvert \times \lvert\Lambda\rvert)$	$O(\lvert\boldsymbol{P}\rvert \times \lvert\Lambda\rvert)$

Table 5 lists the statistics of the traffic traces used in Model II and Model III, where the values are measured by traffic analysis tools: *tcpstat* and *tcptrace*.

4.3 Memory Requirements

For fast lookup and matching, the lookup information and patterns are usually saved in the memory using a tabular structure. Therefore, the memory requirements are estimated according to the number of entries. Since all algorithms need to keep the pattern content in the (external) memory, this section only discusses the extra memory requirement for the tables of each algorithm. In the simulations, the numbers of characters in the clustering pivots (B_1 and B_2) were both assumed to be 1. Because the H^1 of EHMA is a direct index table, the cache memory space (M_I) of EHMA comprises $|\Lambda|$ entries. Based on GFGS and CBS, the number of entries in H^2 is the total number of possible clusters (plus a small memory pool). Since the domain of possible pivot pairs is $\boldsymbol{F} \times \Lambda$, the external memory space for H^2 (M_E) of EHMA is $\mathrm{O}(|\boldsymbol{F}| \times |\Lambda|)$. HMA has the same memory requirement as EHMA. The *shift* table of WM is also a direct hash table. The gram size of WM (block size B) was 3 in the simulations, so the *shift* table of WM had $|\Lambda|^3$ entries. The grouped *skip* table of WM-PH used in the simulations was a direct prefix hash table with a prefix length of three characters. Therefore, the *skip* table of WM-PH comprises $|\Lambda|^3$ entries. Every pattern in the BMH has its own *skip* table of $|\Lambda|$ entries, so that the table of BMH has $|\boldsymbol{P}| \times |\Lambda|$ entries. Because each *skip* table of BMH (for one pattern) is small enough to be loaded into the local memory, for fairness, a cache memory space was allocated to lower the number of external memory accesses. The BMH-O is the original BMH with no local cache and assesses the latency penalty. Notably, WM-PH, AC-C, and BMH-O also require cache memory to store the skip value or one state during the matching process. Table 4 lists the memory requirements of EHMA, HMA, WM, WM-PH, BMH, and AC-C. The scale relation of the parameters is $|\boldsymbol{F}| < |\Lambda| \ll |\boldsymbol{P}| < S \ll |\Lambda|^3$.

TABLE 5
The Statistics of the Traffic Traces

Statistics	Model II	Model III
Average Packet Size (Byte)	467.71	896.1
The Standard Deviation of the Size of each Packet (Byte)	651.06	690.99
Data Transmission Rate (Kbps)	254.13	280.03
Number of Packets per second	69.55	40

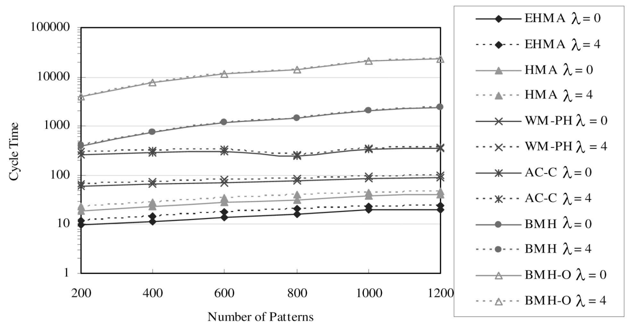


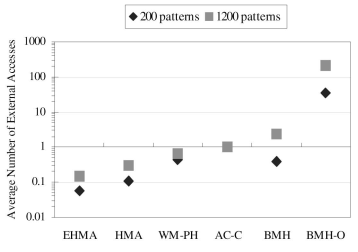
Fig. 8. The average matching time (Ψ) versus the number of patterns $(|P|)$, using Model I with $\lambda = 0$ and $\lambda = 4$, where $w_E = 100$.

In the simulations using Model I, when $|P|$ is 1,200, the H^1 and H^2 of EHMA needs 256 and 19,712 entries, respectively (about 768 bytes on-chip memory and 38.5-Kbyte external memory, including the shared memory pool); HMA has the same number of entries as EHMA but needs smaller entry size as HMA has no *shift* field; the table of WM needs more than 16 million entries (16-Mbyte external memory, in the case without using an additional prefix table); the table size of WM-PH is the same as that of WM; BMH and BMH-O need more than 300,000 entries (300-Kbyte external memory); and AC-C needs 10,731 states (461 Kbytes with each node of 44 bytes). The memory size of all algorithms listed previously excludes pattern content. Obviously, the required memory space of EHMA is quite small.

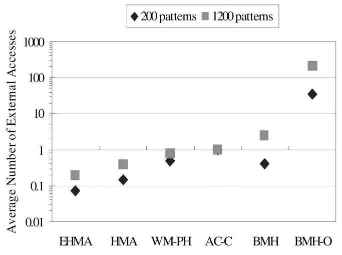
4.4 Results and Discussion

The minimum pattern length of the feeding patterns in Figs. 8, 9, 10, and 11 is only one character, i.e., $M = 1$. Because the minimum pattern length of WM is restricted to be larger than the gram size, in this case three characters, WM is not compared in these figures. In Figs. 8, 9, 10, and 11, the results labeling EHMA in the following simulations use the sampling window with parameters $W = m = |p_i|$, which means that each pattern is sampled in its entirety.

Fig. 8 compares the average matching time (Ψ) of EHMA, HMA, WM-PH, AC-C, BMH, and BMH-O using Model I with different attack loads $\lambda = 0$ and $\lambda = 4$, respectively. It also shows the impact of the number of patterns $(|P|)$ on the matching time. Simulation results reveal that EHMA outperforms others even when $|P|$ and λ increase. EHMA has slightly higher growth rate than WM-PH, because it has a

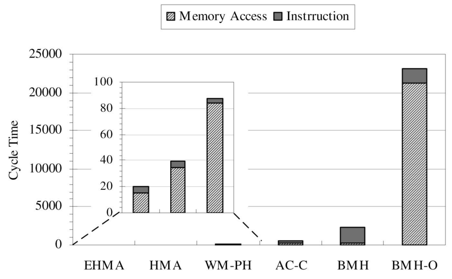


(a)

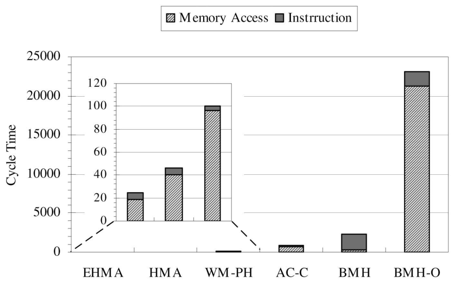


(b)

Fig. 10. The comparisons of average number of external memory accesses (E) using Model I with $w_E = 100$. (a) $\lambda = 0$. (b) $\lambda = 4$.

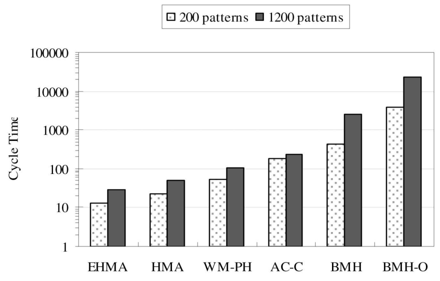


(a)

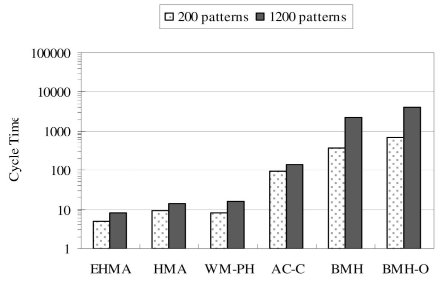


(b)

Fig. 9. The proportion of ψ_I to Ψ and ψ_I to Ψ using Model I with $|P| = 1{,}200$ and $w_E = 100$. (a) $\lambda = 0$ and $|P| = 1{,}200$. (b) $\lambda = 4$ and $|P| = 1{,}200$.



(a)



(b)

Fig. 11. The average matching time (Ψ) versus the number of patterns $(|P|)$ using Model II. (a) $w_E = 100$. (b) $w_E = 10$.

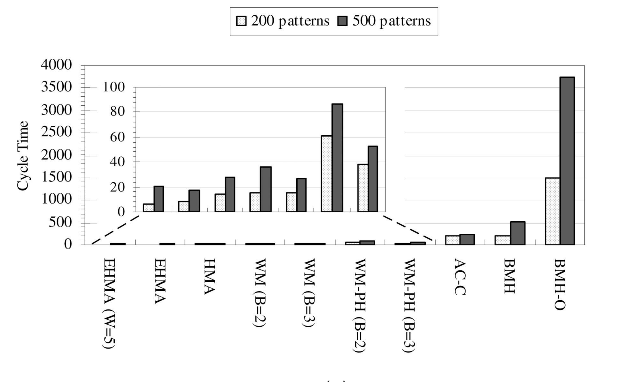
much smaller table size. WM-PH gains performance by having a large direct index table. Notably, the matching time of the original AC using basic structure is independent from $|P|$ and λ. The curves of AC-C increase with $|P|$ and λ owing to the *popsum* used in the AC-C algorithm. The increasing $|P|$ makes the matching time of BMH (BMH-O) rise steeply, because the BMH is originally a single-pattern matching algorithm that simply executes iteratively for multipattern matching.

The case $\lambda = 0$ means that the traffic has no malicious packets. In this case, the proposed EHMA needs only 9.5-19.9 cycles per character on average, which is about 0.9, 3.3-5.3, 16.3-26.8, 40-117, and 408-1,161 times less than the matching time of HMA, WM-PH, AC-C, BMH, and BMH-O, respectively, under various pattern set sizes. We can say that EHMA is very appropriate for network equipment, because generally most packets are innocent ($\lambda \approx 0$). The time available for the detection engine to process the malicious packets rises as the innocent packets are processed more quickly.

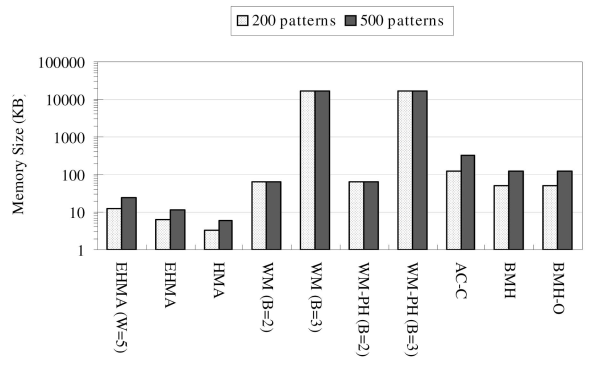
When $\lambda = 4$, then the systems are under heavy attack, and the traffic contains many monitored patterns. In this situation, the matching time of EHMA is about 0.89-0.94, 3.1-4.5, 14.1-24.9, 33.2-96.4, and 335-957 times less than that of HMA, WM-PH, AC-C, BMH, and BMH-O, respectively. Additionally, the performance of EHMA is quite stable, since Ψ rises only slightly as λ or $|P|$ rises.

The processing time of the pattern matching includes the time necessary for instructions (ψ_I) and the time for memory accesses (ψ_M). To investigate their impacts on the algorithms, these two measurements are separated from overall matching costs since different systems introduce different implementation overheads. Fig. 9 displays the proportion of ψ_I to Ψ and ψ_M to Ψ, respectively, for all approaches using Model I with $|P| = 1,200$, where Fig. 9a shows the results under $\lambda = 0$, and Fig. 9b shows the results under $\lambda = 4$. In Fig. 9, the upper part of the bar is ψ_I and the lower part of the bar is ψ_M. The results show that the ψ_I of EHMA is close to that of HMA and WM-PH, but ψ_M of EHMA is much less than others. The proportion of ψ_M to Ψ of BMH seems smaller than others, because the whole skip table of a pattern is idealistically assumed to be loaded within one external memory access and kept in the cache during the matching process for each pattern. Because AC-C compresses the data structure of the state machine, it requires more time to derive the next state pointer. Therefore, AC-C does not have the smallest ψ_I. Simulation results show that the ψ_I does not significantly rise with λ in any of the experiments, because each algorithm has already tried to reduce the computation load (ψ_I). However, ψ_M dominates the overall matching cost. This reveals that the number of external memory accesses is the bottleneck of almost all algorithms. This result also reflects our opinion mentioned previously that the essential issue in designing a high-speed detection engine is to reduce the number of required external memory accesses.

Fig. 10 compares the average number of external memory accesses per character (E) of the state-of-the-art pattern matching algorithms. The figure shows that the E of EHMA is only 0.06-0.19, which is much smaller than others. In other words, EHMA can successfully filter out about



(a)



(b)

Fig. 12. The costs versus the number of patterns ($|P|$) using Model II, $w_E = 100$ and $M = 10$. (a) Average matching time. (b) Extra memory requirement.

94 percent payloads when $|P| = 200$ and 81 percent when $|P| = 1,200$, requiring no external memory accesses and string comparisons. The E of EHMA rises only slightly with rising λ. The increasing rate of E is slightly higher in EHMA than in WM-PH when $|P|$ rises, because EMHA has much smaller table size than WM-PH. Since BMH is based on the single-pattern matching algorithm, its E is proportional to $|P|$. Consequently, the hierarchical matching along with the safety shift strategy is highly effective in reducing the memory latency.

Figs. 11 and 12 adopted Model II as a real-life network environment under intense attack to evaluate the performance of the state-of-the-art algorithms. Since different implementation systems may have different external memory costs (w_E), Fig. 11 illustrates two results with $w_E = 100$ and $w_E = 10$, respectively. To lower the impact of w_E on an algorithm, a very small value of w_E is adopted in Fig. 11b. The results in Fig. 11 indicate that EHMA significantly outperforms others in both cases of small and large pattern set sizes even in the intense attack. EHMA still performs better than others even when the penalty on the external memory access (w_E) is reduced (as shown in Fig. 11b). Comparing EHMA with HMA in Figs. 8, 9, 10, and 11 reveals that the proposed safety shift strategy significantly reduces the number of external memory accesses and thus improves the matching performance.

The minimum length of Snort patterns is one character. However, some detection systems, such as virus detection systems, have larger minimum pattern lengths. The performance of matching algorithms with long minimum pattern

TABLE 6
The Impact of the Size of Sampling Window (W) in the Shift Values of Tables, $|F|$, Actual Matching Shifts, and E Using Model II

|P|	200				500			
	EHMA	EHMA (W=7)	EHMA (W=5)	EHMA (W=3)	EHMA	EHMA (W=7)	EHMA (W=5)	EHMA (W=3)
H^1.shift	0.94	2.71	3.66	4.74	0.91	1.86	2.02	2.49
H^2.shift	1.99	4.89	6.79	8.71	1.99	4.84	6.72	8.65
|F|	13	20	25	39	23	33	47	65
Average Shift	1.5	1.74	1.79	1.84	1.49	1.68	1.74	1.8
E	0.0377	0.0441	0.0431	0.0434	0.1243	0.16	0.1635	0.2512

lengths was examined using Model II, including only the patterns with lengths greater than 10 $(M = 10)$ from Snort patterns, as drawn in Fig. 12. Since the number of patterns whose length is larger than 10 characters in R_1 is around 500, Fig. 12 shows the cases of $|\boldsymbol{P}| = 200$ and $|\boldsymbol{P}| = 500$, respectively. Fig. 12a shows the average processing time (Ψ); Fig. 12b shows the memory requirement of the fast index/hash tables, excluding the memory for pattern contents. Since here M is larger than the gram size of WM, which is three as mentioned before, the performance of WM is compared here. The result labeling $\mathrm{EHMA}(W = 5)$ is the case using EHMA algorithm with $m = M = 10$ and $W = 5$. Recall that the sampling window of EHMA is the entire pattern content, that is, $m = M = |p_i|$. To observe the performance of WM and WM-PH with smaller hash tables, Fig. 12 also displays two additional cases with block size of two characters, $\mathrm{WM}(B = 2)$ and $\mathrm{WM\text{-}PH}(B = 2)$.

Before discussing the simulation results of Fig. 12, Table 6 presents the effect of the size of sampling window (W) on the performance of EHMA in terms of the average shift values of H^1 and H^2, the size of the set of frequent-common grams $(|\boldsymbol{F}|)$ derived from GFGS, the average number of actual shifts, and the average number of external memory accesses, using the same traffic model as in Fig. 12. Table 6 shows that the number of candidate common grams increases with increasing W, resulting in smaller $|\boldsymbol{F}|$. The average number of $H^1.shift$ and $H^2.shift$ increases when W decreases. Since the traffic spectrum is not normally distributed, the actual average number of shifts during matching process is not the same as the average of $H^1.shift$ and $H^2.shift$. However, the trend is the same. E is effected by both $|\boldsymbol{F}|$ and the actual average shift.

Fig. 12a shows that $\mathrm{EHMA}(W = 5)$ outperforms EHMA and others when $|\boldsymbol{P}| = 200$; while EHMA performs better than $\mathrm{EHMA}(W = 5)$ and others when $|\boldsymbol{P}| = 500$. Therefore, reducing $|\boldsymbol{F}|$ becomes more important than increasing the average number of shift values when $|\boldsymbol{P}|$ is large. Since all algorithms need a copy of the pattern contents, Fig. 12b only displays the extra memory requirement of every algorithm for the index/hash tables. Fig. 12b shows that the required memory of EHMA is only slightly larger than that of HMA but much smaller than that of others. The required memory of EHMA grows moderately with $|\boldsymbol{P}|$. The memory of $\mathrm{EHMA}(W = 5)$ is greater than that of EHMA due to the larger $|\boldsymbol{F}|$. As shown in Fig. 12, EHMA is highly effective in reducing the required external memory, providing efficient performance even in the virus-detection-like model.

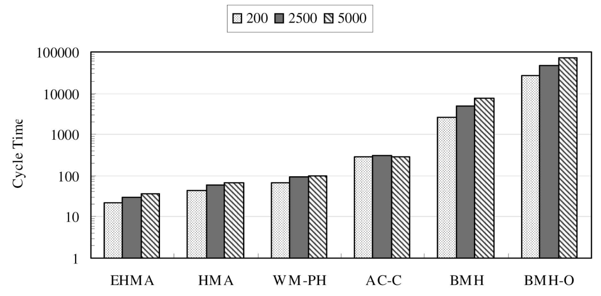


Fig. 13. The average matching time (Ψ) versus the number of patterns $(|P|)$ using Model III, $w_E = 100$.

Fig. 13 uses Model III as real-life normal traffic to show the performance of the algorithms. Meanwhile, to demonstrate the effect of the rising number of patterns on the matching performance, a more recent Snort rule set R_2 of about 5,000 patterns are used in Model III. Fig. 13 shows that EHMA performs better than others even when the pattern set is very large. The matching time of EHMA only moderately increases with the rising $|\boldsymbol{P}|$.

5 Conclusions

The increasing variety of network applications and stakes held by various users are creating a strong demand for fast in-depth packet inspection. The most important component of in-depth packet inspection is an efficient multipattern matching algorithm. This study proposes a novel EHMA for packet inspection. EHMA applies the frequent-common grams obtained by the proposed GFGS to narrow the searching scope and to quickly filter out the innocent packets. The matching process then focuses only on the most suspected packets. EHMA concentrates the patterns into a small on-chip table and performs simple and fast checks. Additionally, EHMA uses the frequency-based bad gram heuristic to speed up the scanning process. The hierarchical matching significantly reduces the average number of external memory accesses to only 6 percent to 19 percent, thus improving the matching performance. The required memory of EHMA is only about 40 Kbytes in addition to the pattern contents of Snort rules. Particularly, EHMA is very simple and can be easily implemented in both software-based and hardware-based platforms. This study also discusses and evaluates current multipattern matching algorithms for NIDSs. Simulation results show that EHMA performs about 0.89-1,161 times better than others. Even under real-life intense attack, EHMA significantly outperforms others. EHMA also works well for the systems with larger minimum pattern size, such as virus detection systems. In conclusion, EHMA facilitates the creation of efficient and cost-effective pattern detection engines for packet inspection.

Acknowledgments

This work was supported in part by MediaTek and in part by the National Science Council of the Republic of China, Taiwan, under Grant NSC -94-2213-E007-02 and Grant NSC 95-2221-E007-054.

REFERENCES

[1] Snort, http://www.snort.org, 2008.

[2] S. Antonatos, K.G. Anagnostakis, and E.P. Markatos, "Generating Realistic Workloads for Network Intrusion Detection Systems," *Proc. Fourth Int'l ACM Workshop Software and Performance (WOSP),* 2004.

[3] R.N. Horspool, "Practical Fast Searching in Strings," *Software Practice and Experience,* vol. 10, no. 6, pp. 501-506, 1980.

[4] A.V. Aho and M.J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Comm. ACM,* vol. 18, no. 6, pp. 330-340, June 1975.

[5] M. Fisk and G. Varghese, "Fast Content-Based Packet Handling for Intrusion Detection," UCSD Technical Report CS2001-0670, May 2001.

[6] O. Erdogan and P. Cao, "Hash-AV: Fast Virus Signature Scanning by Cache-Resident Filters," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '05),* Nov. 2005.

[7] S. Lakshmanamurthy, K.-Y. Liu, Y. Pun, L. Huston, and U. Naik, "Network Processor Performance Analysis Methodology," *Intel Technology J.,* vol. 6, Aug. 2002.

[8] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection," *Proc. IEEE INFOCOM '04,* Mar. 2004.

[9] T.-F. Sheu, N.-F. Huang, and H.-P. Lee, "A Novel Hierarchical Matching Algorithm for Intrusion Detection Systems," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '05),* Nov. 2005.

[10] S. Wu and U. Manber, "A Fast Algorithm for Multi-Pattern Searching," Technical Report TR94-17, Dept. Computer Science, Univ. of Arizona, May 1994.

[11] E. Markatos, S. Antonatos, M. Polychronakis, and K. Anagnostakis, "Exclusion-Based Signature Matching for Intrusion Detection," *Proc. IASTED Int'l Conf. Comm. and Computer Networks (CCN '02),* Oct. 2002.

[12] R.-T. Liu, N.-F. Huang, C.-H. Chen, and C.-N. Kao, "A Fast String Matching Algorithm for Network Processor-Based Intrusion Detection System," *ACM Trans. Embedded Computing Systems,* vol. 3, no. 3, Aug. 2004.

[13] R.S. Boyer and J.S. Moor, "A Fast String Searching Algorithm," *Comm. ACM,* vol. 20, no. 10, pp. 762-772, Oct. 1977.

[14] T.-F. Sheu, N.-F. Huang, and H.-P. Lee, "A Time- and Memory-Efficient String Matching Algorithm for Intrusion Detection Systems," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '06),* Nov. 2006.

[15] C.J. Coit, S. Staniford, and J. McAlerney, "Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort," *Proc. Second DARPA Information Survivability Conf. and Exposition (DISCEX),* 2001.

[16] S. Antonatos, M. Polychronakis, P. Akritidis, K.G. Anagnostakis, and E.P. Markatos, "Piranha: Fast and Memory-Efficient Pattern Matching for Intrusion Detection," *Proc. 20th IFIP Int'l Information Security Conf. (SEC '05),* May 2005.

[17] S. Li, J. Torresen, and O. Soraasen, "Exploiting Reconfigurable Hardware for Network Security," *Proc. 11th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM),* 2003.

[18] S. Kim and Y. Kim, "A Fast Multiple String-Pattern Matching Algorithm," *Proc. 17th AoM/IAoM Int'l Conf. Computer Science,* Aug. 1999.

[19] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep Packet Inspection Using Parallel Bloom Filters," *Proc. 11th Symp. High Performance Interconnects,* Aug. 2003.

[20] H. Lu, K. Zheng, B. Liu, X. Zhang, and Y. Liu, "A Memory-Efficient Parallel String Matching Architecture for High-Speed Intrusion Detection," *IEEE J. Selected Area in Comm.,* vol. 24, no. 10, Oct. 2006.

[21] S. Dharmapurikar and J. Lockwood, "Fast and Scalable Pattern Matching for Network Intrusion Detection Systems," *IEEE J. Selected Area in Comm.,* vol. 24, no. 10, Oct. 2006.

[22] *Vitesse Network Processors,* http://www.vitesse.com, 2008.

[23] *Intel Network Processors,* http://www.intel.com/design/network/products/npfamily/index.htm, 2008.

[24] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer, "Stateful Intrusion Detection for High-Speed Networks," *Proc. IEEE Symp. Security and Privacy (SP '02),* May 2002.

[25] M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," *Proc. Ninth USENIX Security Symp.,* 2000.

[26] C. Cowan, "Defcon Capture the Flag: Defending Vulnerable Code from Intense Attack," *Proc. DARPA Information Survivability Conf. and Exposition (DISCEX III '03),* Apr. 2003.

Tzu-Fang Sheu received the PhD degree in communication engineeering from National Tsing Hua University, Taiwan, in 2009, and the BE and ME degrees in electrical engineering from Tamkang University, Taiwan, in 1998 and 2000. Since 2009, she is an assistant professor of the Department of Computer Science and Communication Engineering at Providence University, Taiwan. Her current research interests include network security, pattern matching, and telecommunication networks. She has been a member of the IEEE since 2000.

Nen-Fu Huang received the BSEE degree from the National Cheng Kung University, Tainan, Taiwan, in 1981 and the MS and PhD degrees in computer science from the National Tsing Hua University, Hsinchu, Taiwan, in 1983 and 1986, respectively. Since 2008, he has been a distinguished professor in the Department of Computer Science, National Tsing Hua University, where he was an associate professor from 1986 to 1994, the chairman from 1997 to 2000, and a professor from 1994 to 2008. His current research interests include network security, high-speed switch/router, mobile networks, IPv6, and P2P-based video streaming technology. He is a member of the IEEE.

Hsiao-Ping Lee received the BE degree in electrical engineering from National Cheng Kung University, Taiwan, in 1992, the ME degree in information engineering and computer science from Feng Chia University, Taiwan, in 2001, and the PhD degree in computer science from National Tsing Hua University, Taiwan, in 2010. Since 2005, he's taught in the Department of Applied Information Sciences at Chung Shan Medical University, Taiwan, R.O.C. He received the award of the 44th Ten Outstanding Young People of Taiwan in 2006. His current research interests include the network security, pattern matching, Bioinformatics, and assistive technology.