Skip Finite Automaton: A Content Scanning Engine to Secure Enterprise Networks

Junchen Jiang
Institute for Theoretical Computer Science, Tsinghua University, China
Email:livejc@gmail.com

Yi Tang, Bin Liu
Tsinghua University, China

Yang Xu
Polytechnic Institute of NYU, USA

Xiaofei Wang
Dublin City University, Ireland

***Abstract*—Today's file sharing networks are creating potential security problems to enterprise networks, i.e., the leakage of confidential documents. In order to prevent such leakage, we propose the *Data Leakage Prevention System* (*DLPS*) which is applied at the entrance of the enterprise network to filter out the outgoing sensitive information. The DLPS is based on a content scanning engine which defines a new type of matching problem, called *longest overlap matching* which also exits in many other applications as a basic problem where contents are delivered by small blocks. We study the problem by comparing it with the traditional pattern matching problem in Deep Packet Inspection (DPI) of Network Intrusion Detection Systems (NIDS) whose solutions are based on finite automata. We develop a new finite automata representation called *Skip-Finite Automata* (*Skip-FA*) which detects the packets carrying sensitive information by using default transitions to implicitly track the overlapping parts between packets' payloads and sensitive files. The simulation results shows that our system achieves a matching speed of about 10B+ per memory access for small file set (> 20KB) and 100B+ per memory access for large file set (> 2500KB). We also find that the memory consumption of Skip-FA is almost the same to that of the original files.**

***Index Terms*—Pattern Matching, Deep Packet Inspection**

I. Introduction

File sharing is one of the most important functions of today's Internet. It provides users various accesses to digitally stored contents, such as videos, audios, documents, and executable programs. However, in spite of the convenient file access, the file sharing techniques (e.g., P2P, FTP, email, etc.) also create potential security problems to today's enterprise networks: the leakage of personal information or confidential documents. The security problem in the file sharing system has received increasing attention by many governments and enterprises [1].

In order to keep the sensitive information from leakage by the file sharing system, *Data Leakage Prevention System* (*DLPS*) should be applied at the entrance of the enterprise network (e.g., gateway or edge router) to filter out the outgoing sensitive information from a global view, rather than a host-level point of view. In Figure 1, we show a DLPS deployed at the gateway of an enterprise network. The *Sensitive File Database* (*SFD*) in the DLPS is used to store the information of the sensitive digital files. For every outgoing packet from the enterprise network, the content scanning engine in DLPS will compare the packet's payload against the information in the SFD. If the payload matches to any contiguous part of the digital file beyond a predetermined threshold, we say a matching is found, and the corresponding action (e.g., blocking or alert) will be performed.

It is worth mentioning that although the concepts are similar, the content scanning employed in DLPS is different from the Deep Packet Inspection (DPI) in a Network Intrusion Detection System (NIDS) [2] [3] (also shown in Figure 1). DPI is an intensively studied practice aiming to attack detection and protocol recognition by matching the content of packets against a set of predefined short signatures [4]–[9]. Similar topics like piracy detection has been studied and discussed in [10]–[14]. Compared to the signature-based DPI in NIDS, content scanning in DLPS has many unique characteristics and constraints as follows.

(1) In DPI, each signature specifies a group of common viruses or attacks, which are usually generated automatically by computer programs based on a certain pattern. However, the digital contents owned by the enterprises are usually created manually (e.g., documents), and it is infeasible to extract the common patterns from different digital files. As a result, the short-signature-based content scanning cannot be applied to implement DLPS.

(2) On Internet, each digital file is transmitted by multiple packets, each of which contains only one piece of the original file. Considering that even the leakage of partial of the file could lead to grave consequences, content scanning in DLPS must filter out the majority of pieces of the sensitive digital files going out from the enterprise network.

As a result, the matching technique in DPI cannot be directly applied in DLPS, and the main objective of the paper is to design a high-speed and memory-efficient algorithm. Specifically, our contribution can be summarized in the following aspects:

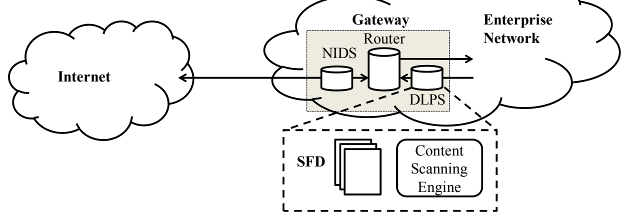


Fig. 1. DLPS in gateway of enterprise network.

1) We formulate the particular content scanning problem in DLPS and develop a novel finite automata representation called *Skip Finite Automata* (*Skip-FA*) which detects the packets carrying sensitive information by using default transitions to efficiently track the overlapping parts between packets' payloads and sensitive files. For large files, we also introduce a compression technique to save the memory usage.
2) We evaluate the prototype system on various file formats and show that our system achieves a matching speed of about 10B+ per memory access while one such instance for a 2500KB file set consumes only about 2300KB of memory.

We also show in [15] that the worst case matching speed of Skip-FA is linear to the packets' size while the memory usage is linear to the size of SFD in most cases. The rest of the paper is organized as follows. Sec. II formulates the problem. Sec. III provides the design of matching mechanism of our system. We show the simulation results in Sec. IV. Finally, Sec. VI concludes the paper.

II. Problem Formulation and Related Work

A. Problem Formulation

Content scanning engine of DLPS stores the entire information of the digital files and the payloads of packets are generally much shorter than those of the digital files. In practice, a packet might contain redundant information (e.g., header and application protocol) in its payload, therefore the content scanning is to find if any contiguous part in the payload of outgoing packets matches to any contiguous part of the digital files. We formally describe the matching problem as follows:

Input: A set of n string rules (called *rule set*) $\mathcal{S} = \{R_1, \ldots, R_n\}$ and a string W (called *input*) over the alphabet set Σ, i.e., $W \in \Sigma^*, R_i \in \Sigma^*, 1 \leq i \leq n$. Assume the length of string R_i is l_i and the length of W is l. Let $R_i = a_{i,1} \ldots a_{i,l_i}$ and $W = b_1 \ldots b_l$.

Output: The longest substring of input W which also appears in some rule $R \in \mathcal{S}$ as a *substring*. An overlapping part is called an *overlap* and the longest one is called the *longest overlap (LO)*.

We call the problem *Longest Overlap* (*LO*) matching. In byte-by-byte processing style, it is only needed to keep the longest suffix of the input which is an overlap at the same time. The left side of Fig. 2 shows an example of LO matching in byte-by-byte processing style where `abcd` is currently kept as the longest suffix which is also an overlap. By keeping these suffixes, we can easily find the LO since it must have once appeared as one of these suffixes. Thus, for simplicity, we refer the LO in our discussion as the longest suffix which is also an LO at any moment of the matching.

Different from the content scanning for DLPS, DPI and IP lookup take the longest prefix (LP) matching as their basis. In longest prefix matching, we have the same input as LO matching, but the output is the longest *prefix*, rather than the longest *substring*, of some rule, which is also the suffix of

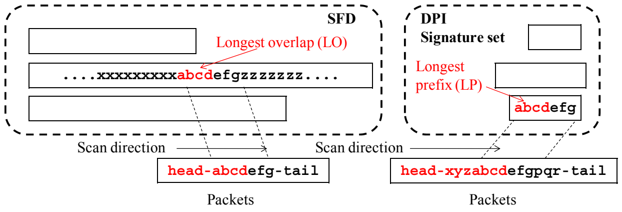


Fig. 2. The difference between LO matching (left) and LP matching (right).

the input string at any time of the matching. The right part of Fig. 2 illustrates the LP matching where the current LP `abcd` is a prefix of some rule. It is clear that LP matching is a *special case* of LO problem since a prefix itself is a substring. So in essence, the solutions developed for DPI can not be employed in DLPS directly.

III. Building Skip-FA from SFD

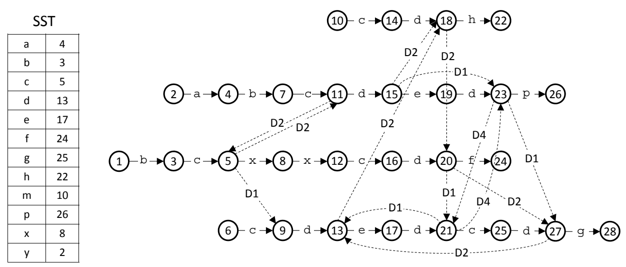
We begin with a basic model, called *Delayed Skip-FA*, which tracks the overlap part completely by default transitions at the cost of unbounded time delayed. For acceleration, we equivalently transform Delayed Skip-FA into *Skip-FA* (Optimal) by redirecting default transitions.

A. A Basic Model: Delayed Skip-FA

In Fig. 3, we first give a sample Delayed Skip-FA of rule set {`cdh, abcdedp,bcxbcdf,cdedcdg`}. As shown, a Delayed Skip-FA consists of forward transitions, called *basic transitions or basic-trans*, which link the states within each rule (e.g., the transition from state 2 to state 4) and some dashed transitions, called *default transitions or D-trans*.

Algorithm 1 Matching procedure of input $W = b_1 \ldots b_l$ against rule set $\mathcal{S} = \{R_1, \ldots, R_n\}$ by Skip-FA

```
1:  Procedure Matching($b_1 \ldots b_l$)
2:  $c \leftarrow 1$; $pos \leftarrow 1$   ▷ $c$: the counter value, $cur$: the current state, $pos$: current point at the input
3:  Search SST with $b_{pos}$ and set $cur$ to be the content;
4:  while $pos \leq l$ or $cur \leq |Q|$ do ▷ $Q$: state set of Delayed Skip-FA
5:      if $cur$ has a basic-tran of $b_{pos}$ to state $next$ then
6:          $cur \leftarrow next$
7:          $c \leftarrow c + 1$; $pos \leftarrow pos + 1$; $cur' \leftarrow cur$;   ▷ Record the current state in $cur'$
8:          continue;
9:      else if $c < v$ then   ▷ counter is less than the lower threshold $v$
10:         goto Line 3   ▷ Restart at $b_{pos}$
11:     else if $cur' = cur$ then   ▷ Delay path is cyclic
12:         goto Line 20;   ▷ Decrease counter
13:     else if $\exists i = \min\{i : c \leq i, cur$ has $(b_{pos}, i)$-tran$\}$ then   ▷ *Omit* this case if Delayed Skip-FA
14:         $next$ is the next state of $(b_{pos}, i)$-tran from $cur$;
15:         $cur \leftarrow next$; $c \leftarrow c + 1$; $pos \leftarrow pos + 1$; continue;
16:     else if $\exists i = \min\{i : c \leq i, cur$ has D$i$-tran$\}$ then
17:         $next$ is the next state of D$i$-tran from $cur$;
18:         $cur \leftarrow next$; continue;
19:     else if $\exists j = \max\{j : cur$ has an D$j$-tran$\}$ then
20:         $c \leftarrow \max\{j : cur$ has an D$j$-tran$\}$;
21:         $next$ is the next state of D$c$-tran from $cur$; continue;
22:     end if
23: end while
```



a	4
b	3
c	5
d	13
e	17
f	24
g	25
h	22
m	10
p	26
x	8
y	2

Fig. 3. The Delayed Skip-FA of rule set {cdh, abcdedp, bcxbcdf, cdedcdg}, the dashed arrows show the D-trans.

We remark some points on the structure of Delayed Skip-FA that should be clarified. Firstly, a Delayed Skip-FA has no start state(s) and, in fact, we find the first state of a matching by using a start state table (SST) which be will introduced later. Secondly, rather than the classic trie structure, with only basic-trans, the Delayed Skip-FA consists of only strands of linked states each of which represents one rule. Thirdly, as we will show later, our system is state-number-insensitive, i.e., as long as no two states have the same number, the system will work well regardless of the numbering of states. Fourthly, every state represents a prefix of some rule, e.g., state 18 represents a prefix cd of rule cdh, but not represents the prefix cd of rule cdedcdg.

Each D-tran is labeled with one combination of a symbol 'D' and an integer i, and called a Di-tran. To give the definition of Di-trans, let us denote the suffix of length i of the prefix represented by state q as $Str(q,i)$. For a state q, we denote the set of characters on its outgoing basic-trans as $Out(q)$ (e.g., $Out(1) = \{\texttt{b}\}$). The Di-trans are defined as follows:

Definition 1: (Default transitions, D-trans) State q has a Di-tran, pointing to state p if and only if:
(a) p is the smallest one[1] in $\Gamma = \{a|a > q, a$ meets (i)(ii)(iii)$\}$, if $\Gamma \neq \emptyset$, otherwise
(b) p is the smallest one in $\Gamma = \{a|a$ meets (i)(ii)(iii)$\} \neq \emptyset$, where (i) $Str(p,i) = Str(q,i)$, (ii) $Out(p) \neq Out(q)$ and (iii) $Str(p,i+1) \neq Str(q,i+1)$.

The intuition of condition (i) is that, we can implicitly keep a suffix of length i of the input string by tracking Di-trans. Conditions (ii)(iii) are used to reduce the number of D-trans without distorting the intuition of (i). For example, for state 15, state 18, 20 and 27 meet condition (i)(ii)(iii) since they share a 2-character suffix cd. For a state q, there are possibly more than one destination state that meets (i)(ii)(iii), and cases (a) and (b) ((a) is prior to (b)) guarantee that only one state is chosen for the Di-tran of q. For example, we choose 18 as the destiny of the D2-tran of state 15 by case (a). In fact, using (a) and (b), from a state q and integer i, we can visit all states meeting (i)(ii)(iii) by tracking Di-trans, e.g., we can visit 18, 20, 21 and 27 by tracking D2-trans from state 15.

The D-trans here are essentially an extension of failure transitions in the well-known Aho-Corasick finite automaton (AC-FA) [16]. Being designed to LP matching (see Subsec. II-A), AC-FA always records the overlap as a prefix of some rule; however, the overlap in the LO matching can start at any position, so every possible starting points shall be checked.

[1]We order the states according to their states' numbers.

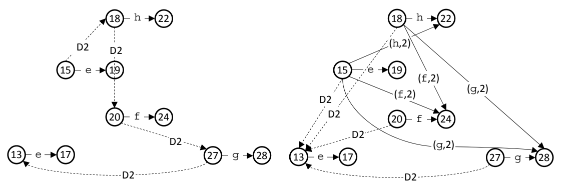


Fig. 4. Reducing the delay diameter by adding skip-trans (many transitions are ignored for simplicity).

We employ an index table, called *Start State Table* (SST). For a block B indexed by v characters, we store the state number $\min\{q : Str(q,v) = B\}$ as its contents and the integer v is called *lower threshold*. Given an input string, we match the first v-character block by searching it in the SST and find the starting state of further matching (see Algorithm 1). Fig. 3 shows an SST with lower threshold $v = 1$. The size of the index table is bounded by the size of rule set, since the number of v-character blocks is less than the number of character in the rule set.

The matching procedure is presented in Algorithm 1 (The `if` block at Line 13 is ignored in Delayed Skip-FA). During the matching phase, we maintain a counter which is used to record the length of current LO. It is initially set zero and once SST successfully matches the first v-character block, it becomes v. Especially, the counter value is incremented only when a character is matched by a basic-tran or the SST. Note that the matching process is *deterministic* since for each state, there is at most one Di-tran for each i. Now, suppose that the input string is bcdf and we match it against the rule set {cdh, abcdedp, bcxbcdf, cdedcdg} using the Delayed Skip-FA in Fig. 3. At the first character b, we lookup it in the SST and find that we should start the matching from state 3. At the same time, the counter is set one, because b is successfully matched by SST. Similarly, after matching the second character c, the current state becomes state 5 and the counter is 2. When matching the third character d at state 5, we find that state 5 has no transition labeled d, so we use its D2-tran (since the counter value is not larger than 2), and reach state 11. We repeat the procedure in state 11 and transfer the current state to state 15 by successfully matching d, and the counter is now 3. Since state 15 has neither a basic-tran of f nor D3-tran, the counter value gets 2 since 15 has a D2-tran to state 18. Then since state 18 has no basic-tran labeled f, we go along the D2-tran of state 18 and reach state 20. Finally, f is successfully matched in state 20.

$$1 \xrightarrow[counter=1]{b \text{ by SST}} 3 \xrightarrow[counter=2]{\texttt{c}} 5 \xrightarrow[counter=2]{D2} 11 \xrightarrow[counter=3]{\texttt{d}}$$
$$15 \xrightarrow[counter=2]{D2} 18 \xrightarrow[counter=2]{D2} 20 \xrightarrow[counter=3]{\texttt{f}} 24$$

For the interest of space, we show the full proof of correctness in our technical report [15].

B. An Optimized Model: Skip-FA

In the motivation example, processing bcdf costs the Delayed Skip-FA 6 transitions and 3 of them are D-trans. In practice, Delayed Skip-FA may travel many D-trans for only one character, resulting in a large delay. Though the time for

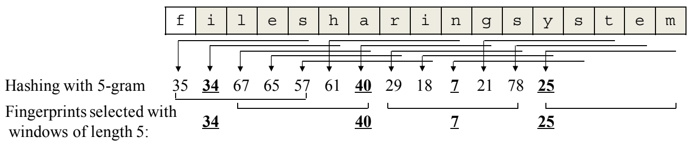

f i l e s h a r i n g s y s t e m
Hashing with 5-gram
35 34 67 65 57 61 40 29 18 7 21 78 25
Fingerprints selected with windows of length 5:
34 40 7 25

Fig. 5. Winnowing sample text. Gray part is covered by the selected fingerprints

	Ave. length(B)	Percent single-byte block (%)	Ave.length of single-byte block (B)	Percent duplicated part (%)	Ave.# of duplicated part
TXT	2370	0	0	1.10	2.3
CPP	1745	0	0	3.40	3.9
PDF	8426	18.3	22.0	13	2.6
EXE	7281	16.8	10.1	16	2.3

TABLE I
BASIC CHARACTERISTICS OF TEST SETS

one character from q with a given counter value c is bounded by the length of the c-delay path from q, the total delay would be unacceptable. To reduce the delay, we redirect the original D-trans by adding new transitions. The intuition is that by redirecting D-trans properly, the lengths of the delay paths can be strictly bounded. We call the length of the longest delayed path of a Skip-FA or a Delayed Skip-FA, *delay diameter*.

Fig. 4 is an illustration of how to reduce the delay diameter from 4 to 1 by adding transitions across rules called *skip transitions* (*skip-trans*) (ones labeled with combinations of a character and an integer). One skip-tran is produced by redirecting one default transition. Accordingly, if q, p, r are neighboring states on the j-delay path from state p and there is a basic-tran $p \xrightarrow{b} p'$ where b is some character, we can redirect the Di-tran ($j \leq i$) between q, p to be from q to r and add an *i-skip-tran* labeled (b, i) from q to p'. Skip-trans are employed to keep the resulting Skip-FA equivalent with the original Delayed Skip-FA. The result of redirecting D-trans and adding skip-trans to a Delayed Skip-FA is called a *Skip-FA*. The matching procedure of Skip-FA follows the complete pseudocode in Algorithm 1. For instance, the matching procedure of input `bcdf` in Skip-FA after the transformation of Fig. 4 is:

$$1 \xrightarrow[counter=1]{b \text{ by SST}} 3 \xrightarrow[counter=2]{\texttt{c}} 5 \xrightarrow[counter=2]{D2} 11 \xrightarrow[counter=3]{\texttt{d}} 15 \xrightarrow[counter=3]{\texttt{f}} 24$$

With the aid of skip-trans, though we are enabled to redirect all D-trans, the memory usage will be greatly raised. As shown in Fig. 4, to reduce the longest delay path to one, we add 5 2-skip-trans. Hence, we consider the following question:

Under the constrain that the delay diameter is m, i.e., there is no delay path with m D-trans or more, how to minimize the number of newly added skip-trans?

It is easy to reduce the bounded diameter minimum cost spanning tree problem, one of the classical NP-hard problem [17], to the above task. Alternatively, we can reduce the cost of each delay path by redirecting D-tran so that the local delay diameter is less than m. It's valid since redirecting any D-tran would never increase the delay diameter. In [15] we give an algorithm following the idea above.

C. Winnowing Algorithm for File Compression

The memory usage of Skip-FA is much related to the number of states, which is the size of rule set, i.e., the total number of characters of the rules. To curb the space explosion when dealing with large files, it is necessary to extract more compact representations from the original.

To this end, we employ the *winnowing algorithm* [18] to shorten both the original files and the payloads of input packets before building Skip-FA. The winnowing algorithm is a document fingerprinting scheme, which ensures that the matched parts between the files and the input are preserved in the matching results of the processed files and input (consisting of fingerprints). In Fig. 5, we first transform the text into a sequence of hashes of the k-grams ($k = 5$ in Fig. 5), so each hash value presents a k-character block. Then by sliding a window of length w ($w = 5$ in Fig. 5) from the leftmost, we select a specific hash value from each window, e.g., we always select the smallest hash value in each window[2]. In the example, we select four fingerprints from a text of length 17 while almost all contents are represented in the fingerprints selected (the gray segment). In general, we can widen the gaps between fingerprints with larger window size w, and make more contents covered by selected fingerprint with large k.

IV. SIMULATION RESULTS

A. Experiment Setup and Test Sets

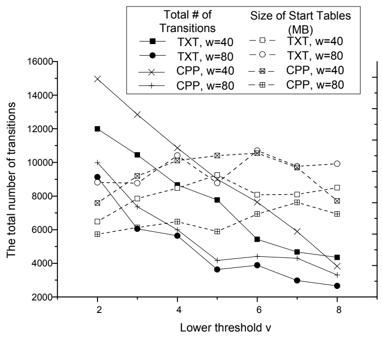
To evaluate the efficiency of the proposed algorithm, we implement a full-featured prototype of our content scanning engine, Skip-FA, with real datasets. The prototype of Skip-FA is implemented in Java. We use a 1.60GHz machine with 1.93GB of main memory to run all the experiments. In all simulation, we employ w-gram hash function in winnowing algorithm where w is the window size. All our simulation use 16-bit hash value for winnowing, i.e. hash value range [0,65535]. We choose four test sets each of which consists of 10 files in the same format which is picked from the following types; TXT: text files(.txt), CPP: code files of C++(.cpp), PDF: Adobe files(.pdf), EXE: executable files(.exe). We believe that they are representative to some basic characteristics (shown in Table I) that impact the performance of Skip-FA.

B. Memory Usage

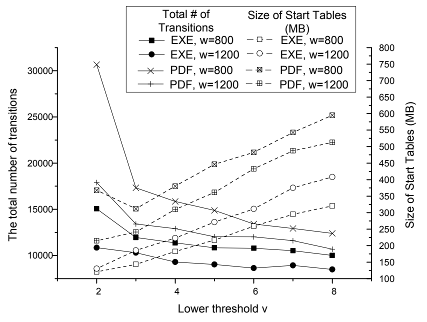
We evaluate the memory usage of Skip-FA by converting the four test sets into four Skip-FAs under various configurations of parameters. We use the transition number as the metric. Fig. 6(a) and 6(b) show that the memory usage drops when the lower threshold v increases. When $v = 8$, the numbers of D-trans are almost the same with those of the basic-trans, but the memory reduction is obtained at the cost of producing large SSTs whose sizes are illustrated by dashed lines.

The plots also reflect characteristics of different file formats in Table I. We see that the memory usages of TXT and CPP decrease quite gently, while PDF shows a fast setback when v is small. It possibly results from the large amount of small duplicated parts whose lengths are commonly of 2 to 3 bytes.

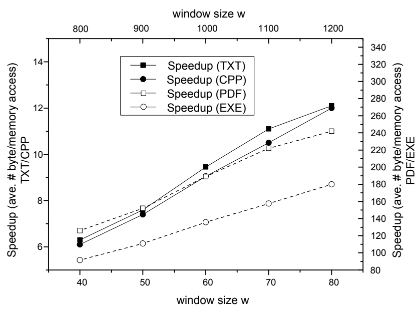
[2]If there is more than one smallest hash value, we choose the rightmost one.



(a) Memory usage of TXT and CPP



(b) Memory usage of EXE and PDF



(c) Speedup of TXT and CPP

Fig. 6. (a),(b): Memory usage (total number of transitions and size of Start Table); (c): matching speedup (average number of bytes per memory access).

When the lower threshold is larger than 2, most of such duplications would no longer produce D-trans. In contrast, TXT and CPP would produce D-trans for repeated blocks of different lengths, leading to a gentle reduction in memory.

C. Matching Speed

During matching phase, the payloads of packets are first converted to fingerprint sequences by winnowing algorithm, and then sent to Skip-FA. The two stages are executed in parallel. The speedup θ is defined by the average number of bytes processed in each memory access. We also assume that each transition costs one memory access, hence,

$$\theta = \frac{\#\text{ of bytes}}{B+S+D} = \frac{\#\text{ of bytes}}{\#\text{ of fingerprints} + D} = \frac{d}{1 + D/\#\text{ of fingerprints}}$$

where B, S and D refer to the numbers of basic-trans, skip-trans and D-trans, respectively and d refers to the density of fingerprints. The number of fingerprints is equal to the sum of B and C according to Algorithm 1. The speedup is clearly influenced by both the number of default transitions visited during matching, and the fingerprint selection scheme. We plot in Fig. 6(c) the speedup, i.e., the number of processed bytes per memory access, of the four test sets under different window size w. Since the number of density of fingerprints d by winnowing algorithm is about $\frac{w+1}{2}$ on average which is proportional to w. Learnt from the plots, the speedup is also about proportional to w, so the delay for each fingerprint, i.e., "$D/\#$ of fingerprints" is about constant. Actually, to successfully consume one fingerprint, Skip-FA needs to go through $3 \sim 4$ D-trans.

V. Acknowledgement

This work was supported in part by the National Basic Research Program of China Grant Nos.2007CB807900,2007CB807901; the National Natural Science Foundation of China Grant Nos.60604033,60553001;the Hi-Tech research and Development Program of China Grant No.2006AA10Z216; NSFC (60625201, 60873250, 60903182), 973 project(2007CB310701), 863 high-tech project (2007AA01Z216) and Tsinghua University Initiative Scientific Research Program.

VI. Conclusion

In this paper, to filter out the outgoing sensitive information from the gateway of enterprises, we propose the Data Leakage Prevention System (DLPS) together with the hardware-based content scanning engine, called Skip-Finite Automata (Skip-FA). As shown by the theoretical analysis and the numerical results, the system performs nicely in both memory usage and matching speedup. In essence, the Skip-FA is developed to deal with a new class of matching problem, the Longest Overlap (LO) matching, which not only exists in DLPS, but also in many other potential applications as a basic problem. In fact, for files, whose information must be all stored in the scanning engine, but are spread by small units, the Skip-FA actually enables the scanning services in such scenarios.

References

[1] G. Sandoval, "Congress to probe p2p sites over 'inadvertent sharing'," 2009.
[2] Snort. http://www.snort.org/.
[3] "Clam antivirus." http://www.clamav.net.
[4] R. Smith and et al., "Xfas: Faster signature matching with extended automata," IEEE Symposium on Security and Privacy (Oakland), 2008.
[5] N. Hua, H. Song, and T. Lakshman, "Variable-stride multi-pattern matching for scalable deep packet inspection," Proc. of INFOCOM, 2009.
[6] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," Proc. of ACM SIGCOMM, 2007.
[7] R. Smith and et al., "Deflating the big bang: fast and scalable deep packet inspection with extended finite automata," Proc. of SIGCOMM, 2008.
[8] B.C. Brodie and et al., "A scalable architecture for high-throughput regular-expression pattern matching," Proc. of ISCA, 2006.
[9] J. Jiang and Y.T.B.L. Y. Xu, T. Pan, "Pattern-based dfa for memory-efficient scalable multiple regular expression matching,," IEEE ICC, 2010.
[10] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," Communications Surveys & Tutorials, IEEE, 2005.
[11] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan., "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," In Proc. ACM SOSP, Oct., 2003.
[12] F. Zhao, T. Kalker, M. Medard, and K. Han, "Signatures for content distribution with network coding," in Proc. of IEEE ISIT07, (Nice, France), July 2007.
[13] K. Zetter, "KaZaA Delivers More Than Tunes," The Wired Magazine, January 2004.
[14] S. Shin, J. Jung, and H. Balakrishnan, "Malware prevalence in the kazaa file-sharing network," In ACM SIGCOMM Internet Measurement Conference (IMC), 2006.
[15] http://s-router.cs.tsinghua.edu.cn/SFA.pdf.
[16] A.V. Aho and M.J. Corasick, "String matching: An aid to bibliographic search," 1975.
[17] G.M. R. and J.D. S., "Bounded component spanning forest," Computers and Intractability: A Guide to the Theory of NP-Completeness, 1979.
[18] S. Schleimer, D.S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in ACM SIGMOD, 2003.