Reduction of Power Consumption for Pipelined DPI Systems on FPGA*

HANSOO KIM1,2 AND JU WOOK JANG2,+
1Document and Image Department
National Forensic Service
Seoul, 158-707 Korea
2Electronic Engineering Department
Sogang University
Seoul, 121-742 Korea

We propose a scheme to reduce power consumption in pipelined AC-DFA (Aho-Corasick deterministic finite automaton) tries for deep packet inspection (DPI). It is based on our observation that the access frequency drops dramatically as the input goes through stages of the pipelined implementation of the AC-DFA trie. Experiments show that the access frequency of the fourth stage is one thousandth of the access frequency of the first stage. So, we slow down the stages that are not frequently used in the pipelined AC-DFA trie to reduce unnecessary power consumption. Also, we turn on the next stage before a stage performs a pattern matching, to reduce delays and clock skew without any additional hardware components. Our scheme shows a 25% reduction in power consumption, compared with the state-of-the-art DPI scheme [3] with a pipelined AC-DFA trie.

Keywords: intrusion detection, pattern matching, deep packet inspection, Snort, frequency scaling

1. INTRODUCTION

Deep packet inspection (DPI, Snort [8] for example) has become one of the most reliable and trustworthy systems to eliminate network threats such as viruses, malicious packets and DDoS attempts. The functions of DPI systems rely on multi-pattern string matching, which scans the input stream to find all occurrences of a predefined set of string-based patterns [2].

Many multi-pattern string matching solutions adopt the well-known Aho-Corasick (AC) algorithm, where the system is modeled as a deterministic finite automaton (DFA) [1]. Along with the considerations of speed and accuracy, the power consumption required for the functioning of the DPI system is of significant concern. The majority of the power consumption used for DPI is for multi-pattern string matching [5], hence a large amount of research on reducing the power consumption is focused on string matching.

In addition, pipelining has performed a great role in string matching, so a number of research studies have been done on reducing the power consumption in pipelined architecture. Caching is applied to frequently-used stages by an analysis of the traffic lo-

Received June 18, 2013; revised August 7 & September 7, 2013; accepted September 17, 2013.
Communicated by Cho-Li Wang.
* This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (NRF-2013R1A1A2011856).
+ Corresponding author: jjang@sogang.ac.kr.
HANSOO KIM AND JU WOOK JANG

Adaptive clocking is implemented [12], but it applies only to the entire processor and requires a large number of power-consuminating buffers. Power balancing through the stages of the pipeline [13, 14] is applied, but it does not directly address the power reduction.

Besides, a number of researches show that when pattern matching is performed with the pipelined AC-DFA trie, the number of accesses decreases drastically according to the stage of the pipelined AC-DFA trie [4, 5]. This may cause the imbalance of resource and waste of power consumption.

Motivated by these observations, we propose a power saving scheme for the pipelined AC-DFA trie in DPI systems. Our main contribution can be summarized as follows. First we slow down the clock frequency of the rarely-used stages to eliminate unwanted power consumption. Second, to reduce delays and clock skew without any additional hardware components, we turn on the next stage before a stage performs a pattern matching. As a result, our new scheme reduces power consumption by 25% when compared to the state-of-the-art scheme [3] in DPI systems.

The rest of this paper is organized as follows. Section 2 reviews multi-pattern string matching and relevant recent studies, and Section 3 shows the proposed scheme which reduces power consumption. Section 4 shows the experimental results and Section 5 concludes this paper.

2. RELATED WORKS

2.1 The AC-DFA Trie with Pipelining

The AC-DFA converts a pattern set which contains \( n \) characters into a deterministic finite automaton with \( O(n) \) states. Once the DFA which can be stored as a state transition table is built, it reads the input stream one character per clock cycle. Each input character is processed only once and results in exactly one state transition [2].

Fig. 1 illustrates a construction of the AC-DFA for four patterns of ‘cell’, ‘cream’, ‘real’ and ‘ear’. The AC-DFA starts with constructing a trie (AC-trie) where the root is the default non-matching state. Each pattern to be matched adds a state to the trie, at a rate of one state per character, starting at the root and going to the end of the pattern [2]. This is called a goto transition [1]. For example, the pattern ‘cell’ adds states 1, 4, 8, and 12 as shown in Fig. 1. In case there is a mismatch in the goto transition, an additional
transition is added which is called a failure transition [1]. All states except states 4, 10 and 13 use the root as the default failure transition state. Some patterns share strings with other patterns. In this case, a failure transition can be made to other non-root state. This is illustrated in the following example. Patterns ‘real’ and ‘ear’ share a string ‘ea’. At state 10 in Fig. 1, if the input character ‘r’ is encountered, a transition can be made to state 11 instead of the root state. The goto transitions are called the forward transitions, and the failure transitions are called the cross transitions. In Fig. 1, the forward transitions are denoted as the solid lines with arrows, while the dotted lines with arrows denote the cross transitions (the cross transitions to the root state are not shown).

Fig. 2 shows a mapping of the AC-DFA trie in Fig. 1 onto a pipeline, with each level of the AC-DFA trie to one stage of the pipeline. Using the aspect that the cross transitions are added into the AC-DFA trie for a failed match in order to reuse the history information without restarting from the root, Pao et al. [3] deliver the input characters to all the pipeline stages in parallel, including the new input characters to the root with a one character offset at each clock cycle so as to remove the cross transitions to the stages that are on the pipelines. With this approach, all the cross transitions can be removed.

Suppose that the input stream is ‘crear’ and at clock 1, the first input character ‘c’ is fetched to the root node and comparison is made against ‘c’, ‘r’, and ‘e’ in order. Match. Node 1 in stage 0 is reached and the second input character, ‘r’, is fetched at clock 2. At the same time, the character ‘r’ is also fed to the root node in stage 0 to process an input stream staring with ‘r’. In this way we process virtually multiple input streams, ‘crear’, ‘rear’, ‘ear’, ‘ar’, and ‘r’ simultaneously. The cross transition from 13 to 11 can be removed since node 11 is eventually reached while processing ‘ear’.

2.2 The Dynamic Power Management Scheme in Semiconductor Devices

There are two primary types of power dissipation in CMOS circuitry: dynamic and static [6]. Because the dynamic power has been the dominant source of the power dissipation in VLSI circuits and systems, various power management architecture and power saving technologies of semiconductor devices have emerged using Eq. (1).

\[
P = \frac{1}{2} CV^2 f \alpha
\]  

(1)
$P$ is the dynamic power consumption, and $C$ is the capacitance, $V$ is the supply voltage, $f$ is the clock frequency and $\alpha$ is the switching activity. To decrease the dynamic power, any of the parameters in the formula may be reduced [6].

Among a large number of researches, clock gating is a highly effective method to minimize the energy dissipation without any appreciable degradation in the quality of service [7]. It scales the operating clock frequency to minimize the energy dissipation, but delays such as the processing latency and clock skew should be considered. Delays and clock skew may happen if a processing logic executes an instruction when its clock speed is scaled or lower than its original clock frequency. To reduce delays and clock skew in implementing clock gating, additional hardware components such as clock buffers and repeaters should be inserted in high frequency pipelined data paths which require additional power and delays that may lead to performance degradation [6, 7].

3. MOTIVATION AND THE PROPOSED SCHEME

3.1 Motivation

A number of researches show that when pattern matching is performed with the pipelined AC-DFA trie, the access frequency to different stages can vary a lot [4, 5]. When the input data is processed, the stages of the pipelined AC-DFA trie is sequentially accessed from the first stage, and if there is a mismatch in the preceding stage, the rest of the stages are not accessed. So, the first stage is always accessed by any input data, while the last few stages are seldom accessed [5].

Fig. 3 shows the number of accesses to each stage of the pipelined AC-DFA trie. The input data is randomly-generated 25 million characters with each character coded as a byte, and the pipelined AC-DFA trie has 150 stages (Not all the stages are shown in Fig. 3). The horizontal axis represents the stage of the pipelined AC-DFA trie and the vertical axis represents the number of accesses to the stage in a logarithmic scale. We observe the fact that the number of accesses decreases drastically as input data traverses
the stages of the pipelined AC-DFA trie [4].

Our scheme is to reduce power consumption by slowing down those stages which are rarely used.

### 3.2 The Proposed Scheme

For clarity, we define the following terms.

- $N_c$: the number of input characters to be processed in the DPI system
- $N_a$: the total number of accesses to the pipelined AC-DFA trie in processing pattern matching
- $N_s$: the length of the longest pattern
- $N_m$: the maximum number of accesses to the AC-DFA trie
- $P_a$: the power consumption of a stage of the AC-DFA trie when active
- $P_r$: the power consumption of a stage of the AC-DFA trie when ready
- $P_{r, \text{slow}}$: the power consumption of a stage of the AC-DFA trie when the clock frequency of the ready stage is slowed down
- $P_{\text{conv}}$: the total power consumption of the conventional DPI system [3]
- $P_{\text{prop}}$: the total power consumption of the proposed DPI system

The number of input characters ($N_c$) is the number of characters of the input data to the pipelined AC-DFA trie. $N_a$ is the total number of accesses to the pipelined AC-DFA trie in processing pattern matching. $N_s$ is the length of the longest pattern, which equals to the height of the AC-DFA trie. $N_m$ is the maximum number of accesses to the AC-DFA trie, which is $(N_a \times N_s)$. $P_{\text{conv}}$ is the total power consumption of the conventional DPI system [3] processing $N_c$. $P_{\text{prop}}$ is the total power consumption of the proposed DPI system processing $N_c$.

A stage of the pipelined AC-DFA trie performs a pattern matching when it fetches a character from the input stream and compares the character against the patterns stored in the stage. A stage of the AC-DFA trie is defined to be active when the stage performs a pattern matching. A stage of the AC-DFA trie is defined to be ready when the stage is ready for a pattern matching. A stage of the AC-DFA trie is defined to be slowed down when the stage is ready but its clock frequency is decreased. A stage is turned on when the stage changes from slowed down to ready. A stage is turned off when the stage changes from ready to slowed down.

$P_a$ is the value of the power consumption of a stage when the stage is active. $P_r$ is the value of the power consumption of a stage when the stage is ready. $P_{r, \text{slow}}$ is the value of power consumption of a stage when the stage is slowed down. We assume that all the pipeline stages in DPI systems have the same hardware architecture and the power consumption of all the stages is equal for each status — active, ready or slowed down.

Fig. 4 shows an example of our architecture. When a pipeline stage performs the pattern matching, it signals to the following stage to turn it on. The signal is called the toggle signal and denoted as chain lines with arrows in Fig. 4. The stage results in the match notification to the following stage as a result of the pattern matching. In Fig. 4, dotted lines with arrows denote the match notifications ($H$ denotes the number of the pipeline stages).
Consider Fig. 2 as an example and assume the input stream as ‘crearemo’. According to Fig. 2, the number of the pipeline stages $H = 6$. As the initial state, the toggle signals from Stage 1 to Stage 4 are set to 0 ($x = 0$) that lead to the clock frequency of Stages 2 to 5 as Clk/4. We set the slowed down clock frequency as 1/4 of the original clock frequency (Clk) rather than 0, because turning on a stage from 0 to Clk of its clock frequency causes considerable delays [6, 7]. The appropriate value of the scaling constant (1/4) depends on the practical hardware system and implementation techniques, and the value of 1/4 is suitable for the system we implement.

At clock 0, Stage 0 fetches the first character ‘c’ from the input stream and compares it against the patterns ‘c’, ‘r’ and ‘e’ stored in Stage 0. Since there is a match with the pattern ‘c’ between node 0 and 1 of the AC-DFA trie in Fig. 2, Stage 0 sends the match notification to Stage 1 to perform the pattern matching. At clock 1, Stage 1 fetches the second character ‘r’ from the input stream and compares it against the patterns ‘e’ and ‘r’ stored in Stage 1. Before the pattern matching, Stage 1 sends a toggle signal ($x = 1$) to Stage 2 to turn it on so that the clock frequency of Stage 2 is set to Clk. Since there is a match with the pattern ‘r’ between node 1 and 5 of the AC-DFA trie in Fig. 2, Stage 1 sends the match notification to Stage 2 in the same manner. At clock 2, Stage 2 performs the pattern matching with the third input character ‘e’ and the pattern ‘e’ stored in Stage 2, after turning on Stage 3. In short, the stage following the active stage is turned on before the active stage performs the pattern matching. A stage can perform the pattern matching without any delays or clock skew, because it is already running on the original clock frequency (Clk) when it performs the pattern matching.

When there is a mismatch on Stage $k$ ($1 \leq k \leq H-2$), it sends a toggle signal ($x = 0$) to Stage $k+1$ to turn it off and the clock frequency of Stage $k+1$ is set to Clk/4. At clock 5, Stage 0 fetches the sixth character ‘e’ from the input stream and compares it against the patterns ‘c’, ‘r’ and ‘e’ stored in Stage 0. Since there is a match with the pattern ‘e’ between node 0 and 3 of the AC-DFA trie in Fig. 2, Stage 0 sends the match notification to Stage 1. At clock 6, Stage 1 fetches the seventh character ‘m’ from the input stream and
compares it with the pattern ‘a’ stored in Stage 1, after turning on Stage 2. Since there is a mismatch with ‘m’ and ‘a’, Stage 1 turns off Stage 2 by sending a toggle signal \((x = 0)\) and the clock frequency of Stage 2 is set to Clk/4. Detailed procedure is shown in Fig. 5.

### 3.3 Slowing Down the Rarely Used Stages

We decrease the clock frequency of the stages that are rarely used, because scaling the operating clock frequency is most effective power saving scheme when there are a large number of inactive blocks in CMOS circuitry [7]. A large reduction in power consumption can be achieved by slowing down the clock frequency of the stages that are rarely used.

In the conventional pipelined AC-DFA trie [3], the total power consumption can be described as in Eq. (2).

\[
P_{\text{conv}} = N_a \times P_a + (N_m - N_a) \times P_r
\]  

After slowing down the clock frequency of the rarely-used stages, the total power consumption can be described as in Eq. (3).

\[
P_{\text{prop}} = N_a \times P_a + (N_m - N_a) \times P_{r, \text{slow}}
\]  

\(P_r\) is greater than \(P_{r, \text{slow}}\), according to the Eq. (1) and the definition of \(P_r\) and \(P_{r, \text{slow}}\). So, we can reduce the power consumption by slowing down the clock frequency of the rarely-used stages as \((N_m - N_a) \cdot (P_r - P_{r, \text{slow}})\). The more rarely-used stages there are, the more power is saved in our scheme.

For example, consider Fig. 2. Assuming the input stream as ‘crearemo’, the number of accesses at stage 0, 1, 2, 3, 4 and 5 is 8, 5, 4, 2, 1 and 0, respectively. So, \(N_c = 8, N_s = 6, N_a = 8 + 5 + 4 + 2 + 1 = 20\) and \(N_m = 48\). Assume that the global clock frequency of the system (Clk) is 200 (MHz) and the clock frequency of a slowed down stage is set to a quarter of the global clock frequency, which is 50 (MHz). According to the commonly-used power consumption estimator [15] with its basic configuration, the values of \(P_a, P_r\) and \(P_{r, \text{slow}}\) are estimated as 1.9 (W), 0.9 (W) and 0.6 (W), respectively. Applying the estimated values of \(P_a, P_r\) and \(P_{r, \text{slow}}, P_{\text{conv}} = 63.2\) (W) and \(P_{\text{prop}} = 54.8\) (W) in Fig. 2. We obtain that 13.3% of the power consumption is reduced using the proposed scheme.

### 3.4 Reducing Delays and Clock Skew

To reduce delays and clock skew while slowing down the clock frequency of the rarely-used stages, we turn on the next stage before a stage performs a pattern matching instead of using additional hardware components such as clock buffers or repeaters. The clock frequency of the stage following the active stage is set to the original clock frequency (slowed down to ready), so the power consumption of the stage is changed to \(P_r\) from \(P_{r, \text{slow}}\). Given this, Eq. (3) can be rewritten as in Eq. (4).

\[
P_{\text{prop}} = N_a \times P_a + N_s \times P_r + (N_m - N_a - N_s) \times P_{r, \text{slow}}
\]  

In the example of Fig. 2, the modified \(P_{\text{prop}}\) is obtained as 57.2 (W) which leads to
the power reduction of 9.5% compared to the conventional scheme [3]. The pseudo-code of the proposed scheme is shown in Fig. 5.

<table>
<thead>
<tr>
<th>functions and variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
</tr>
<tr>
<td>-------</td>
</tr>
</tbody>
</table>
| BOOL  | matched | 1: matched  
0: not matched |
| BOOL  | toggle_signal \(k\) | \(1: \text{turn on Stage } k \text{ if turned off}\)  
\(0: \text{turn off Stage } k \text{ if turned on}\) |
| BOOL  | clock_status \(k\) | \(1: \text{Stage } k \text{ is turned on}\)  
\(0: \text{Stage } k \text{ is turned off}\) |
| BOOL  | pattern_match \(k\) | \(\cdot \text{the pattern matching}\)  
\(\cdot \text{comparison of the input character against the patterns at Stage } k \text{ of the pipeline}\)  
\(\cdot \text{returns 0 or 1 as a result}\) |
| void   | match_notification \(k\) | \(\cdot \text{notify Stage } k \text{ to perform the pattern matching}\) |

< initialization >

// Stage 0 and 1 are always turned on and other stages are turned off
// \(H\): the number of stages of the pipeline

toggle_signal (0) = 1

toggle_signal (1) = 1
toggle_signal \(k\) = 0 \((2 \leq k \leq H - 1)\)

// performance at stage \(k\) \((k \geq 1)\)

if [match_notification \(k\)]
{
    toggle_signal \((k + 1)\) = 1
    // turn on the following stage

    matched = pattern_match \(k\)
    // perform the pattern match

    if (matched)
    {
        match_notification \((k + 1)\)
        // notify the following stage to perform the pattern match

        if \([\text{clock_status } (k - 1) == 0 || (k == 2 && \text{match_notification } (1) == 0)]\)
            toggle_signal \(k\) = 0
        // turn off the current stage if the previous stage is turned off
    }
    else
    {
        toggle_signal \((k + 1)\) = 0
        // turn off the following stage
    }
}

Fig. 5. The pseudo-code of the proposed scheme.
4. EXPERIMENTAL RESULTS

Experiments on the ruleset from Snort [8], the well-known DPI system, are performed. We construct the original AC-DFA trie from the content field text data of the ruleset and apply our system. The number of distinct rule sets is 86,248 and the constructed AC-DFA trie has a maximum of 150 levels (the largest ruleset has 150 characters) with a total of 425,248 states.

The DPI system with the AC-DFA trie is implemented with Microsoft Visual C++ 2008. Within the system, the pipelined architecture of the string matching is implemented with SystemC 2.2 [9] and the power consumption is measured using the modified PowerSC [9, 10]. The system is implemented in software but mainly designed to be applied on FPGA, so the essential hardware parameters are adopted from Xilinx Vertex-4 XC4VLX100 [15] as our target device and from the DDR3 DRAM [16] in the commonly used DPI system.

Two data sets are used as the input data. 25 million characters are randomly generated as the first input data and an English text of 341,362 characters (a text version of English Fairy Tales [11]) is used as the second input data.

Two pipelined DPI systems are implemented for the experimental analysis: (1) the simple linear pipeline system and (2) the partial match pipeline system of Pao et al. [3]. The simple linear pipeline system constructs the pipeline, with the number of the pipeline stages equal to or more than the maximum length of the patterns. We set the number of the pipeline stages of the simple linear pipeline system as 48 (the partially pipelined AC-DFA trie [2] is adopted because there are no cross transitions beyond level 48 of the AC-DFA trie we implement), and the experimental results from two input data are shown in Tables 1 and 2.

### Table 1. Results from randomly generated data on the simple linear pipeline system.

<table>
<thead>
<tr>
<th></th>
<th>Pao et al. [3]</th>
<th>proposed scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>average number of stages</td>
<td>active</td>
<td>ready</td>
</tr>
<tr>
<td>1.04</td>
<td>46.96</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 2. Results from an English text on the simple linear pipeline system.

<table>
<thead>
<tr>
<th></th>
<th>Pao et al. [3]</th>
<th>proposed scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>average number of stages</td>
<td>active</td>
<td>ready</td>
</tr>
<tr>
<td>3.12</td>
<td>44.88</td>
<td>0</td>
</tr>
</tbody>
</table>

The number of the pipeline stages of the partial match pipeline system can be equal to or less than the maximum length of the patterns. Patterns which are longer than the number of the pipeline stages are segmented with the length of each segment equal to or
less than the number of the pipeline stages. If the first segment of a pattern is matched with the input stream, the pattern is said to be partially matched [3] and the next segment is performed the pattern matching with the following input data. We set the number of the pipeline stages of the partial match pipeline system as 16, and the experimental results from two input data are shown in Tables 3 and 4.

| Table 3. Results from randomly generated data on the partial match pipeline system. |
|---|---|---|---|
| Pao et al. [3] | proposed scheme |
| average number of stages | power consumption (W) | average number of stages | power consumption (W) |
| active | ready | slowed-down | active | ready | slowed-down |
| 1.04 | 14.96 | 0 | 1782.6 | 1.04 | 1.00 | 13.96 | 1392.7 | 21.9 |

| Table 4. Results from an English text on the partial match pipeline system. |
|---|---|---|---|
| Pao et al. [3] | proposed scheme |
| average number of stages | power consumption (W) | average number of stages | power consumption (W) |
| active | ready | slowed-down | active | ready | slowed-down |
| 3.11 | 12.89 | 0 | 28.3 | 3.11 | 1.03 | 11.86 | 23.7 | 16.0 |

In the simple linear pipeline system, our scheme reduces 25.4% of power consumption on randomly generated data and 23.9% on an English text, compared to Pao et al. [3]. In the partial patch pipeline system, our scheme reduces 21.9% of power consumption on randomly generated data and 16.0% on an English text, compared to Pao et al. [3]. Also, there are no significant differences of the total processing time between our scheme and Pao et al. [3]. While processing the input data of 25 million characters in the simple linear pipeline system, the average number of stages that are active or ready in a clock on Pao et al. [3] is 48 and the average number of stages that are active or ready in a clock on the proposed scheme is 2.04 (which is merely 4.3% of that of Pao et al. [3]).

5. CONCLUSION

The proposed scheme reduces the power consumption by 25% on the Snort pattern set, compared to the state-of-the-art pipelined AC-DFA pattern matching system [3]. Our algorithm also shows that slowing down the clock frequency of the rarely-used stages is effective in reducing the power consumption, for multi-pattern string matching such as Snort, the well-known DPI system. We will extend our scheme to other multi-pattern string matching applications and power-aware systems in the future.

REFERENCES


**Hansoo Kim** received the B.E. and M.E. degrees in Electronic Engineering from Sogang University, Korea, in 2002 and 2004, respectively. After that, he worked as a Junior Engineer at the Digital Media Research Laboratory in LG Electronics until 2005, and as a Senior Engineer in Nextreaming Corp. until 2006. He is currently a Researcher and forensic expert in National Forensic Service, Korea, and also a Ph.D. candidate in Electronic Engineering from Sogang University. His research interests and
work experiences include string matching, application-layer network protocols, home networking, IPv6, multimedia streaming, digital forensics, document analysis, the internet security and cybercrime.

**Ju Wook Jang** received the B.S. degree in Electronic Engineering from Seoul National University, Seoul, Korea, the M.S. degree in Electrical Engineering from the Korea Advanced Institute of Science and Technology (KAIST), and the Ph.D. degree in Electrical Engineering from the University of Southern California (USC), Los Angeles, USA. From 1985 to 1988 and 1993 to 1994, he was with Samsung Electronics, Suwon, Korea, where he was involved in the development of a 1.5-Mb/s video codec and a parallel computer. Since 1995, he has been with Sogang University, Seoul, Korea, where he is currently a Professor. His current research interests include WiMAX protocols, mobile networks and next generation networks. He received LG Yonam overseas research grant in 2001. He has also built systems for videoconferencing, streaming, home networks and ad hoc networks using protocols like RTP, SIP, multicast, and IPv6.