A Practical Intrusion Detection System for Internet of Vehicles

Wenliang Fu[1], Xin Xin[1*], Ping Guo[1], Zhou Zhou[2]

[1] School of Computer Science and Technology, Beijing Institute of Technology, Beijing, 100081, China
[2] National Engineering Laboratory for Information Security Technologies, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

Abstract: Internet of Vehicles (henceforth called IoV) is a public network system and high-value target for intrusions that may cause efficiency issues, privacy leakages or even physical damage. Conventional intrusion detection methods are normally designed for the Internet infrastructures which cannot directly apply in the context of IoV. This work proposes an FPGA based intrusion detection method that can not only achieve real-time scanning performance but also be applied in vehicular environment. We evaluate our scheme on a Xilinx FPGA based platform. Experiments show that the proposed system can achieve a throughput of more than 39 Gbps on existing FPGA platform which is about 15% higher than state-of-the-art techniques，and the total power consumption for the prototype is about 7.5 w. Moreover, the processing latency of the prototype is about 4 us and is about one sixtieth part of the popular software IDS systems.
Keywords: internet of vehicles; intrusion detection; VANET; FPGA

I. INTRODUCTION

IoV is an open and integrated network system that connects human intelligence, vehicles, things, environments and the public Internet [1]. Although many researches that support IoV have been proposed [2-4], academic attentions paid on intrusion detection are limited. In IoV premises, each entity can communicate with others directly and share information and intelligence. Unlike the Internet, no central security devices are deployed between two communication peers, thereby each entity of IoV should be responsible for its own security. Moreover, IoV is a high-value target for intruders to exploit for the massive private and driving-aid information exchanged in it.

To develop intrusion detection system for IoV, there are several facts should be considered. First, each entity in IoV should have an intrusion detection system (IDS) to protect itself. The physical size and power supply of the IDS are limited. Traditional Intrusion Detection System (IDS) do not consider those factors. Second, IoV is an intelligence-aid and real-time system. The deployment of IDS in each entity should not alleviate the real-time performance of IoV. In other word, the scanning performance and system latency of IDS for IoV should be competent enough for being deployed in real-time environment.

IDS replies on regular expression (henceforth called RegEx) matching engines for traffic inspection. Finite automata, which include

Deterministic Finite Automata (DFA) and Non-deterministic Finite Automata (NFA), are normally used to carry out RegEx matching engines. Two types of complexity are should be considered while using finite automata to recognize intrusion patterns: storage costs and processing complexity. DFA have low processing complexity but high storage costs while NFA have high processing complexity but require low storage costs. Therefore, a wide range of attempts have been made to eliminate the disadvantages of DFA and NFA by exploiting software and hardware features of various platforms including general purpose processor [5–7], multi-core processor [8], network processor [9], GPU [10-11], FPGA [12-13], etc. Because FPGA is relatively space saving, require less power, easy to deploy in cars and can be deeply customized for complex computation, there is a huge potential for developing FPGA based IDS system for IoV. Moreover, FPGA could have direct access to network interface, and it leads to better performance by eliminating processing latency caused by software driver, memory, PCIe interface, etc.

The processing capacity of FPGA based RegEx matching system is co-determined by two factors: operating frequency and the number of bytes the system can process every clock period. Norio Yamagaki, etc. propose a multi-stride conversion algorithm which can generate multi-stride NFAs of 2^k bytes [14]. Multi-stride NFA is a variation of NFA, and it can process multiple bytes each time. The conversion of NFA into multi-stride NFA is simply done by parsing and combining transitions of NFA. Essentially, multi-stride conversion on NFA is an effective approach to promote performance and reduce latency. But the scheme suffers from the following problems. First, transition number of multi-stride NFA grow dramatically in large stride length cases. The generated automaton can easily exhaust the hardware resource of state-of-the-art FPGAs. Details will be given in the problem formulation section. Second, with more transitions, the maximum operating frequency of the system is lowered because of increased processing complexity. A more efficient data model is needed to support real-time performance in the context of IoV.

This paper proposes an intrusion detection system which is designed for vehicular environment. It can achieve real-time performance while consuming less space and energy. To achieve the goals, we focus on FPGA platform and propose a novel data model based on current multi-stride NFA. Specifically, this paper explores the transition growth problem of multi-stride NFA, and tries to find its crux. Then, a novel NFA model, named Link-NFA, is proposed. At last, a bitmap based optimization scheme and a matching architecture of Link-NFA are presented.

The reminder of this paper is structured as follows. In section II we shall review previous work. Studies on transition explosion problem of multi-stride NFA and the root cause are given in Section III. In Section IV, Link-NFA concept and its construction algorithm are presented. We discuss an optimization and a matching architecture on Link-NFA in Section V. In Section VI we evaluate the implementation cost and performance of Link-NFA model on FPGA. Section VII concludes this paper.

In this paper, the authors propose an intrusion detection system for IoV which can achieve high performance, low latency and acceptable energy consumption.

II. Related Work

Many researches on RegEx matching have been proposed in recent years. Generally, those works can be put into two categories: memory based and non-memory based. The former type relies on memories to implement NFA or DFA; the latter type use FPGA logic resources such as LUTs and flap-flops to carry out automata.

Memory based methods carry out matching logic by running a lookup engine according to input characters and transition tables. Matching results can be obtained by looking up a result table using current active states. To improve the inspecting performance and rule-supporting capacity, researches have been proposed on increasing the efficiency of memory access and usage from the prospective of transition prediction [15, 16], alphabet reduc-

tion [15–16], increasing matching stride length [8, 13], etc. Because huge amount of table lookups are performed every second to support high speed input inspecting, memory has become a bottleneck to system performance. Moreover, to achieve high speed, memory based methods usually deploy multiple identical copies of matching engines with each have an independent set of tables. The redundant setup further exacerbates the problem of memory access and usage, and increase storage costs.

Non-memory based method utilizes FPGA logic to build automata. In 2008, Tamer F. Badran etc. introduced an automata concatenating algorithm which can support complicated meta-characters such as "*", "." and "?", and they also proposed a low level conversion method that can build circuits for processing k-byte each clock period [17]. In the same year, Yamagaki N etc. proposed a high level multi-stride scheme that can convert 1-byte input NFA to 2^k-byte input NFAs [14]. Experiments show that their method can achieve a throughput of no more than 10 Gbps. Non-memory based methods do not rely on table-lookup to recognize RegExs. Therefore, it eliminates potential system bottlenecks for memory access and usage. The key issue of non-memory is to build efficient automata that can not only achieve large stride length but also control processing costs (e.g. limited number of transitions).

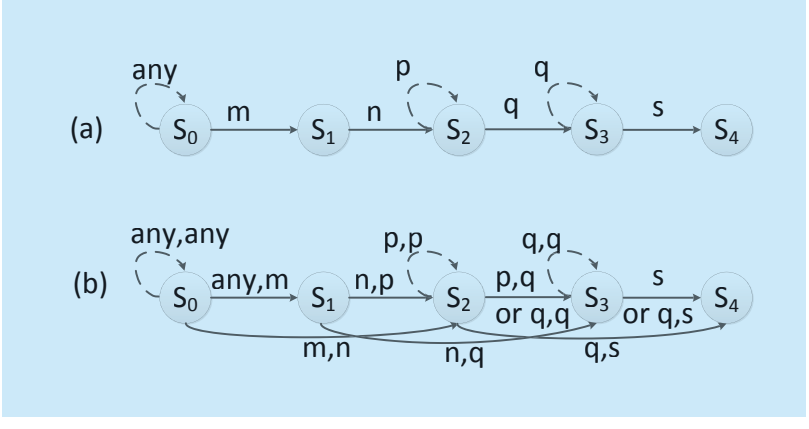

Fig.1 *1-byte (a) and 2-byte (b) NFAs for RegEx "mnp*q+s" ("any" matches any single character)*

Moreover, because the development speed of FPGA following Moore's law and it is much faster than that of memory bandwidth and capacity, there is more potential for non-memory based methods to keep up with the rapid development of network link speed. In this paper, we propose an extended NFA model named Link-NFA, which is targeted at eliminating the transition problem of multi-stride NFA. The design goal is to achieve a throughput of multiple tens of Gbps while accommodating thousands of practical RegEx rules based on current FPGA techniques.

III. Problem Formulation

Efficient data model for RegEx matching is the key to IDS. In this section, we firstly evaluate the problem of current multi-stride NFA model based on Snort [18], L7-filter [19] and Bro [20] rulesets and existing multi-stride scheme [14]. Then, the crux of the problem is analyzed. Snort and BRO are practical IDS rulesets; L7-filter is a representative RegEx ruleset that usually used as a test case to compare RegEx matching methods. To help understand the problem, a simple example of 1-byte NFA and its corresponding 2-byte stride NFA are given in Fig.1.

3.1 The transition explosion problem of multi-stride NFA

We generate multi-stride NFAs of different stride length and plot the transition growth trend for each ruleset in Fig.2. For Bro ruleset, the growth rate of transitions increases gently according to stride length and never exceeds 10 times. But for Snort and L7-filter rulesets, the trends increase rapidly and the growth rates can scale up to as many as about 1500 and 560 respectively in the 32-byte stride cases. Consider that it requires more than 1% of an Altera Stratix II EP2S180 FPGA's logic resource to implement 1-stride NFAs for 128 Snort rules [18], the costs for the corresponding 32-stride NFAs are too expensive for current FPGAs.

Through the results, following conclusions

can be drawn. First, complexity of RegEx rules has a significant effect on the number of transitions of the corresponding multi-stride NFA. Second, current multi-stride model is only suitable for building NFAs of small stride such as 4-byte. A more efficient NFA model is required for large stride cases.

3.2 Analysis

RegEx is a sequence of characters that is capable of describing a certain string pattern. Each character in a regular expression is either understood to be a regular character with its literal meaning or a meta-character with its special meaning. From the perspective of functions, there are mainly two kinds of meta-characters: one is for representing a set of regular characters (e.g. "." represents any single character) and the other is for describing repeat patterns of the preceding elements (e.g. "+" and "*"). To carry out the logic of repeat meta-character, repeat transition, which points to its starting state, has been introduced such as those in dotted lines in Fig.1. Because multi-stride conversion algorithm combines multiple 1-byte transitions for each multi-byte transition, repeat transition can lead to automata explosion for the following reasons.

First, each repeat transition might be employed 0 to k times to generate k+1 k-stride transitions (k is the stride length and the transition growth is related to k). Second, if multiple repeat transitions exist, they can cooperatively lead to a drastic growth of the k-byte transitions. The situation can be much worse when stride length is larger. For example, in the conversion of a 32-byte NFA from the NFA described in Fig.1(a), the number of k-byte transitions between S_0 and S_4 is dependent on how many combinations we can get by picking up 28 choices from the three repeat transitions sequentially (4 is necessary for reaching S_4 from S_0). The same cases occur when generating transitions between S_1 and S_3, etc. Third, for transitions that are destined to the final states of a k-byte stride NFA, it may cost less than k bytes to reach the final states. Taking the NFA in Fig.1(a) for example, multi-stride transitions that are pointing to S_4 can accept n bytes where n is a natural number and is in the range of 1 to k. The number of those ending transitions is also impacted by the repeat transitions.

To further examine the impacts of repeat meta-characters on the scale of multi-stride NFA, we generate 32-byte NFAs for Snort, BRO and L7-filter rulesets and gather statistics about the relations between transition growth and the number of repeat meta-characters. The results are shown in Table 1. For RegEx rules that averagely have no more than 1 repeat meta-character, the transition growth rate is no more than 10. As the number of repeat-characters increases, the transition growth rate follows. For RegEx rules that have more than 5 repeat meta-characters, such as those in the last column of Table I, the transition growth rate can be more than 10k times.

Therefore, we can safely draw the conclusion that the multiple occurrences of repeat meta-characters in RegEx rules cause the transition explosion problem of multi-stride NFA. It is possible to reduce transition number by

Table I *The number of repeat meta-characters according to degrees of transition growth (Times) of 32-byte stride NFA*

Growth	<10	10-100	100-1k	1k-10k	>10k
Snort	0.87	2.16	3.3	4	8.3
Bro	1	N/A	N/A	N/A	N/A
L7-filter	0.64	2.27	2.9	3.8	5.3

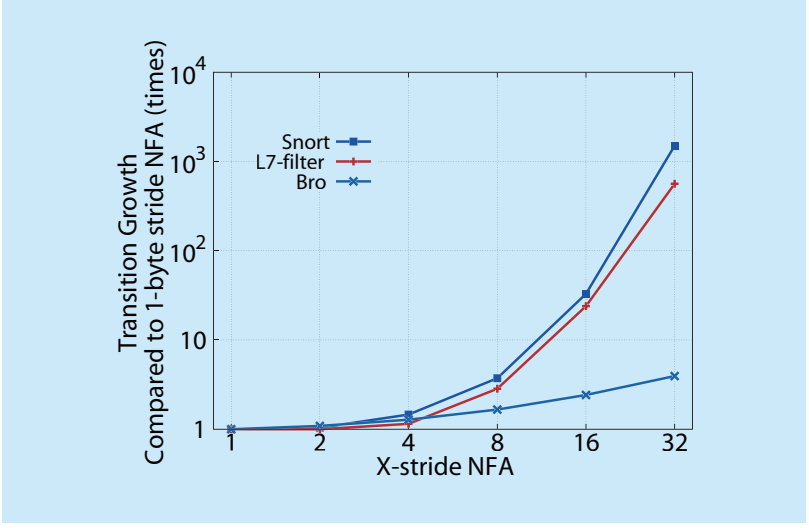

Fig.2 *Average transition growth rate (# of x-stride NFA vs. # of 1-stride NFA)*

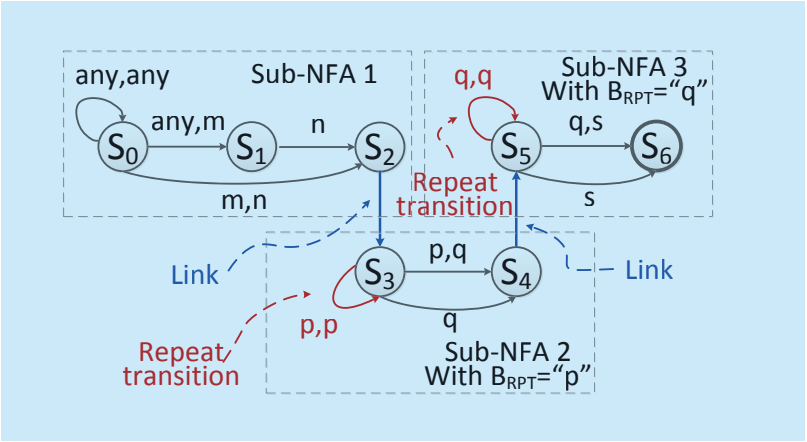

Fig.3 *An example of 2-byte stride Link-NFA for RegEx "mnp*q+s"*

Table II *Notations for Link-NFA Tuples*

Tuple	Explanation
S	A finite set of states
$\sum$	A finite input character set
s_0	The starting state
F	A set of accepting states
k	How many bytes processed each time
σ_{basic}	Transition function: $S \times \sum^k \rightarrow S$
σ_{link}	Transition function: $S \rightarrow S$
L	Which bytes of the input have been processed
θ	Input update function: $\sum^k \times L \times B_{RPT} \rightarrow \sum^k$

introducing an extended NFA model which can cut off the interactions of meta-characters when generating multi-stride transitions.

IV. Approach

4.1 Link-NFA concept

In this section, we describe an extended model, namely Link-NFA, which utilizes multiple small and linked multi-stride NFAs (henceforth called Sub-NFA) to replace one large multi-stride NFA. Each Sub-NFA contains one repeat transition at most (pointing to its start state) and is concatenated with others by a special type of transitions called link. Multi-stride conversion is done based on Sub-NFA therefore transition number can be controlled. An easy example of Link-NFA is shown in Fig.3.

The repeat transition of each Sub-NFA accepts k identical characters, as shown in Fig.3. We named the character the repeat character (henceforth called B_{RPT}) of the Sub-NFA. Link-NFA model is formally represented as a 9-tuple, $\{S, \sum, s_0, F, k, \sigma_{basic}, \sigma_{link}, L, \theta\}$, as are described in Table II.

A major difference between multi-stride NFA and Link-NFA is that there are two types of transition functions in Link-NFA premise: σ_{basic} and σ_{link}. The former one is used to determine next states within a Sub-NFA and the latter one is used to find next states across Sub-NFAs. Specifically, if current state is not a Sub-NFA's last state, function σ_{basic} is used to determine which state in current Sub-NFA is the next state; else, function σ_{link} is used to find the first state of the next Sub-NFA according to current state.

When moves to a new Sub-NFA, it is possible that the previous Sub-NFA only consumes the first L bytes of the input and L is no more than k. Input update function θ is used to demilitarized the used bytes by replacing them with the same number of the new Sub-NFA's B_{RPT}. Therefore the matching process can carry on with the updated input and the new Sub-NFA. Updating input while crossing Sub-NFAs is another important difference in Link-NFA from multi-stride NFA. The key point is that the matching logic corresponds to the RegEx not only in a Sub-NFA but also across Sub-NFAs. More detailed explanation will be provided in the following sections.

All Sub-NFAs follow a similar style as the examples in Fig.3. In the figure, each Sub-NFA has at most one repeat transition and is very simple compared with one large multi-stride NFA. Targeted at this feature, there is a potential to reduce the implementation costs of Link-NFA on FPGA platform. Details will be discussed in the following section.

4.2 RegEx matching algorithm on Link-NFA

The pseudo code for RegEx matching on Link-NFA is as shown in Algorithm 1. The

function *REGEX_MATCH_TOP* reads input from buffer and then starts/continues matching processes. The function *PROC_INPUT* activates state according to the k-byte input, current state, byte indicator L, σ_{basic} and σ_{link} , and determines if a match is triggered. Function *PROC_INPUT* will not stop recursion until the input k bytes are processed or a matching result is obtained.

Fig.4 shows how to process string "mmnpqqqs" on a 2-byte stride Link-NFA. The Link-NFA is described in Fig.3. In the starting of each processing cycle, a batch of 2-byte input is loaded from buffer. Processing starts with S0, then activates S0 and S1 according to σ_{basic} and the 2-byte input at PC_1. Because the input at PC_1 has been processed, processing clock 2 starts with a newly loaded input. Then, S2, which is the last state of Sub-NFA 1, is activated because of σ_{basic} and the first byte of the input at PC_2. Since only one byte is consumed, L is set to 1 and Cur_State is set to S3 according to σ_{link} . In the same time, the first byte of the input is replaced by the BRPT of Sub-NFA 2. S3 is activated again according to σ_{basic} , the updated input and Cur_State. The matching process continues until the accept state S6 is activated and reports a hit.

Algorithm 1 RegEx Matching Algorithm on Link-NFA

Function: REGEX_MATCH_TOP
1: *Start_State* ← the first state of the Link-NFA
2: **while** Input Buffer is not empty **do**
3: *Cur_Input* ← read k bytes from *Input_Buffer*
4: **for each** activated state *Active_State* at current processing cycle **do**
/* *Continue matching from activated states* */
5: Call PROC INPUT(Active State, Cur Input, 0)
6: **end for**
/* *Start new matching process*/
7: Call PROC_INPUT(Start State, Cur Input, 0)
/* *End of current processing cycle* */
8: end while
Function: PROC_INPUT(*Cur_State*, *Cur_Input*, *L*)
1: **if** L is greater than 0 **then**
2: B_{RPT} ← repeat byte of the new Sub-NFA
/* *Update input according to L and* B_{RPT} */
3: Replace the first L bytes of *Cur_Input* with L B_{RPT}
4: **end if**
5: *Next_State* ←σ_{basic}(*Cur_State*,*Cur_Input*)
6: **if** *Next_State* is a final state of current Sub-NFA **then**
7: **if** *Next_State* is also a final state of the Link-NFA **then**
8: Report a match and Exit
9: **else**
10: n ← number of bytes processed in last transition
/* *Update byte locator L* */
11: L ← L + n
/* *Move to next Sub-NFA* */
12: *Next_State* = σ_{link}(*Cur_State*)
13: Call *PROC_INPUT*(*Next_State*, *Cur_Input*, *L*)
14: **end if**
15: **else**
/* *Will be the continued in the next cycle* */
16: Mark *Next_State Active_State* in next processing cycle
17**: end if**

4.3 Constructing Link-NFA

The flow diagram of constructing Link-NFA is described in Fig.5. It mainly consists of 4 steps.

Step 1: Compiling RegEx to NFA: RegExs can be converted into NFA according to classic construction algorithm.

Step 2: Splitting NFA: The NFA is split into multiple single-byte but linked NFAs, called Split-NFA. Each Split-NFA contains no more than one repeat transition and is connected with others by links. The splitting method is simple: it parses NFA in depth-first order and splits a Split-NFA if there is more than one repeat transition. To guarantee the matching logic holds across Split-NFAs, two adjacent Split-NFAs have one overlapped state. A simple example of Split-NFAs, which is originated from Fig.1(a), is shown in Fig.6. It is worth noting that *S*2 and *S*3 in the original NFA are splitting points. Two states are generated as marked in dotted line for relaying match logic of between Split-NFAs.

Step 3: Multi-stride conversion: Multi-stride conversion is done based on Split-NFA according to current scheme [14]. Links are not considered as transitions in the conversion process.

Step 4: Generating Link-NFA: Link-NFA utilizes multiple Sub-NFAs to recognize string patterns. Therefore each Sub-NFA should be capable of passing down enough information so that others can continue the matching pro-

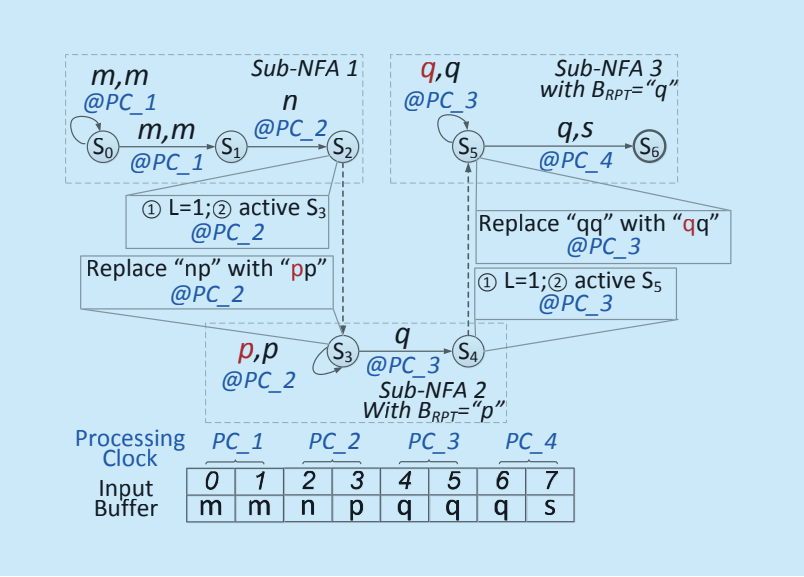

Fig.4 *The matching process to accept string "mmnpqqqs" on a 2-byte stride Link-NFA (the Link-NFA is described in Fig.3)*

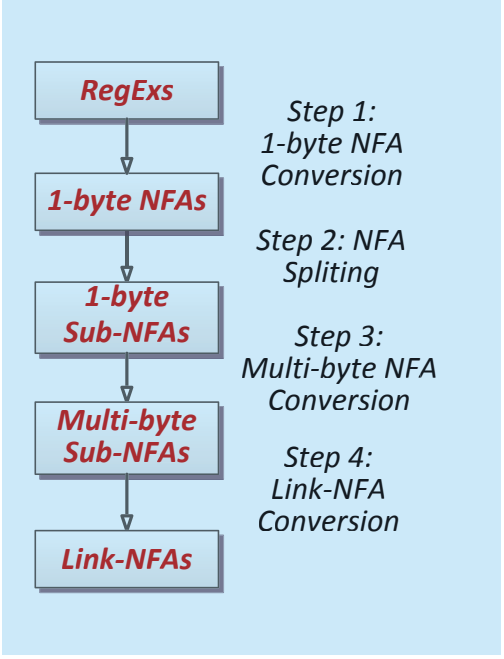

Fig.5 *Flow diagram for constructing Link-NFA*

cess. The job is done by labeling final transitions with the number of bytes it can process (which are destined to ending states).

Because multi-stride conversion uses multiple 1-byte transitions to build k-byte transitions, the repeat transition, which located on the first state of a Split-NFA, might be employed 0 to k times for k+1 k-byte transitions. In other words, the B_{RPT} is used as padding byte while generating k-byte transitions. This feature guarantees that RegEx matching on the Link-NFA accords with the identical logic of the RegEx.

V. Optimization and Matching Architecture

This section firstly describes an optimization method on Sub-NFA which can reduce the processing complexity of RegEx matching on Link-NFA. Then, a high speed matching architecture for RegEx matching on Link-NFA is discussed.

5.1 Optimization on Link-NFA

Fig.7 shows a typical example of 32-byte stride Sub-NFA. There are totally 34 transitions in the Sub-NFA. From the perspective of implementation costs, those multi-stride transitions totally consume LUTs/FFs for 592 byte comparisons in FPGA.

At matching stage, for each Sub-NFA, the k-byte input possibly consists of three sections: processed section, soft matching section and hard matching section as described in Fig.8. The processed section filled with bytes that have been handled by the previous Sub-NFA. The soft matching section possibly contains multiple B_{RPT}. The hard matching section starts with non-B_{RPT} bytes and materially decides the next activated states in the Sub-NFA. To reduce costs, we firstly locate the hard matching section and then start matching from there.

The location of hardware matching section is done based on a bitmap of the input. Specifically, a k-bit bitmap can be created for indicating if the corresponding location in the k-byte input is B_{RPT}. In other words, if the i_{th} byte of the input is B_{RPT}, the i_{th} bit of the bitmap is set to 1; otherwise it is set to 0. It costs k byte comparisons for generating the bitmap where k is the stride length. The length of the soft matching section can be obtained by finding the left-most 0 of the bitmap in the knowledge of L. Specifically, a batch of if-else

bit comparisons can be used and it requires (1+k)*k/(2*8) byte comparisons. The offset of hard matching section can be calculated according to k, L and the length of soft matching section (L is either passed down by the previous Sub-NFA or is 0). At last, matching results can be obtained by starting comparison from hard matching section whose maximal length is set and is at most k-byte.

The optimization requires hardware resources for three parts: generating bitmap of the input, locating hard matching section on bitmap and determining the next states. The three parts respectively consume hardware resources for implementing k, (1+k)*k/(2*8) and t byte comparisons, where t is the minimal byte requirement for satisfying the Sub-NFA (t is 2 in the Sub-NFA of Fig.7). In the specific case of Fig.7, it requires hardware sources of about 130 byte comparisons which reduces costs by 83% compared with the un-optimized case. The optimization method is more effective when stride length is larger.

It is worth noting that the bitmap based optimization method cannot apply if B_{RPT} is a set meta-character, and it includes B_1 as exemplified in Fig.7. That is because an input byte can be regarded as B_{RPT} and B_1 in the same time, and it causes confusion about the location of the hard matching section. The problem will be solved in next section.

5.2 Link-NFA Pipeline

In Link-NFA, each Sub-NFA is only responsible for a part of the RegEx rules, and its in-

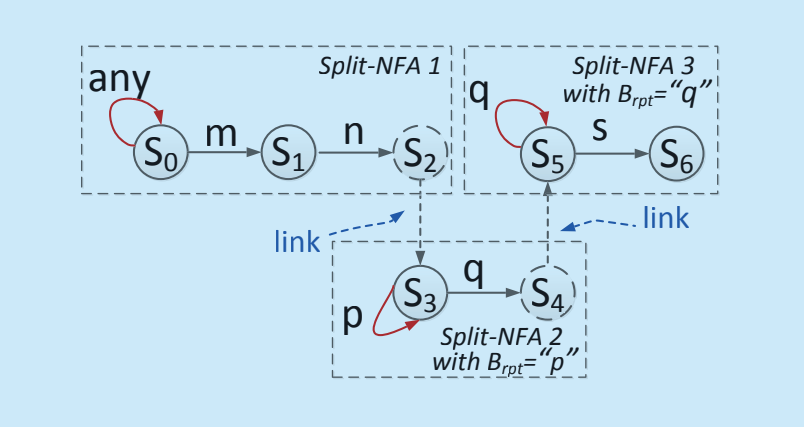

Fig.6 *Split-NFA Example for RegEx "mnp*q+s"*

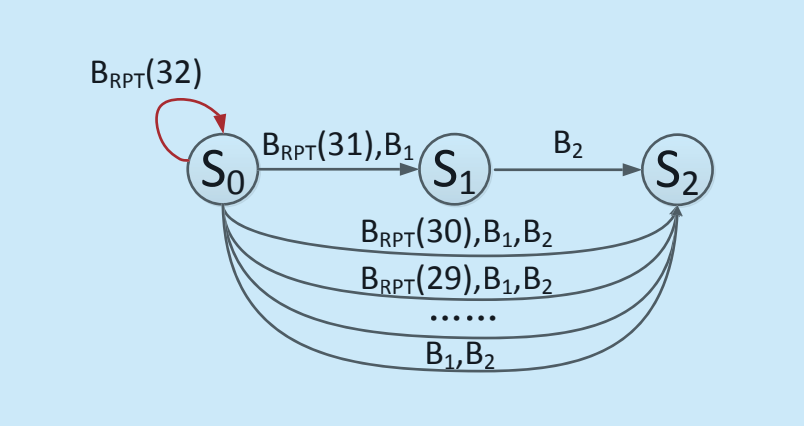

Fig.7 *An example of 32-byte stride Sub-NFA (the number in brackets indicates how many times the left element repeats and link transitions are omitted)*

Table III *The average number of transitions in Link-NFA according to stride length and the growth rate for each rule (GR, # k-byte stride vs. # 1-byte stride)*

Ruleset	1-byte stride		2-byte stride		4-byte stride		8-byte stride		16-byte stride		32-byte stride	
	#Tran.	GR	#Tran.	GR	#Tran.	GR	#Tran.	GR	#Tran.	GR	#Tran.	GR
Snort	43.18	1	47.86	1.11	59.36	1.37	89.81	2.08	161.96	3.75	307.58	7.12
L7-filter	47.91	1	54.16	1.13	71.71	1.50	135.65	2.83	453.11	9.46	1489.69	31.09
BRO	13.6	1	14.61	1.07	16.61	1.22	20.61	1.51	28.61	2.10	44.61	3.28

Table IV *FPGA Resource Usages on Virtex6 xc6vlx550t (Total LUT/Register number: 687360)*

Ruleset	1-byte stride		2-byte stride		4-byte stride		8-byte stride		16-byte stride		32-byte stride	
	Used	Util.	Used	Util.	Used	Util.	Used	Util.	Used	Util.	Used	Util.
Snort	8250	1.20%	8646	1.26%	15157	2.21%	30833	4.48%	43351	6.31%	57920	8.43%
L7-filter	6113	0.89%	6339	0.93%	7043	1.02%	9067	1.32%	14759	2.15%	24512	3.57%
BRO	1624	0.24%	3118	0.45%	5156	0.75%	8644	1.26%	18653	2.71%	28814	4.19%

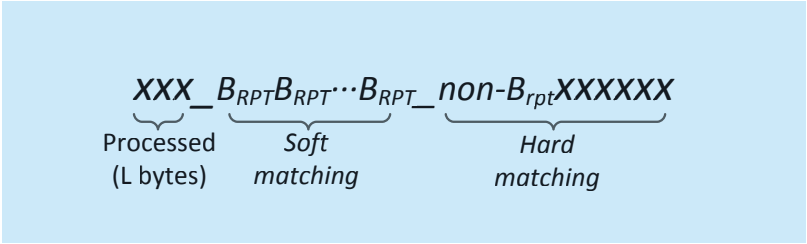

Fig.8 *An example of input with possible three sections.*

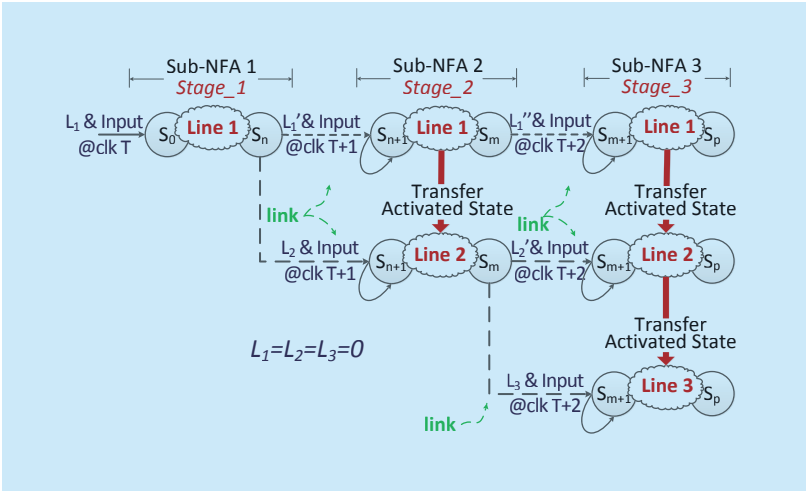

Fig.9 *RegEx matching architecture on Link-NFA with Sub-NFAs abundantly arranged for three possible matching processes*

put data is related to the previous Sub-NFAs. Therefore, RegEx matching on Link-NFA is basically a sequential process. Sophisticated matching architecture is needed to achieve high performance.

The Sub-NFAs of a Link-NFA can be arranged in pipeline style, as shown in Fig.9. In the figure, each pipe line handles matching process that is started from its first stage. For example, Line 1, Line 2 and Line 3 in Fig.9 process states that are active in Stage_1, Stage_2 and Stage_3. If a state is activated in Stage_2 of Line 1 at the end of current cycle, it should be move downward to Line 2 so that pipeline Line 1 is ready for new matching processes in next cycle. The number of stages in the matching architecture is decided by the number of repeat characters exist in the corresponding RegEx rule.

If a Sub-NFA's B_{RPT} is a set meta-character, and it includes B_1 as shown in Fig.7, multiple hit results might be reported by this Sub-NFA. Those results have the same input and different byte locators. In this case, FIFOs should be deployed to temporally store intermediate results. The impacts on resource consumption will be measured in evaluation section.

Taking data structure, optimization method and matching architecture of Link-NFA into account, FPGA is well fit for it for the following reasons. First, the number of NFAs in Link-NFA would be N times that of multi-stride NFA, where N is related to the average number of repeat meta-characters of RegEx rules. Only FPGA is capable of handling hundreds or even thousands of small NFAs in parallel. Second, the optimization method requires fast and large amount of bit comparisons, which is efficient to implement in FPGA but can cause high computation complex in non-FPGA based hardware platform.

VI. Evaluation

In this section, Link-NFA is evaluated from two aspects. First, we convert RegEx rules into Link-NFAs and measure the transitions growth of the output Link-NFA. Second, we implement the Link-NFAs on FPGA and measure the hardware costs and the performance of it.

The rulesets we used to generate Link-NFAs are from Snort, BRO and L7-filter. The reason for choosing them is that those rulesets come from representative application scenarios and have different complexities. Each ruleset contains 128 random selected RegEx rules. The hardware implementation is based on a Xilinx Virtex-6 xc6vlx550t FPGA using Xilinx Design Suite 13.4 and Verilog language.

6.1 Number of transitions

The number of transitions is a direct indicator to show if Link-NFA can efficiently control transition growth. Table III shows the average numbers of transitions and growth rates for each ruleset with respect to different stride length ranging from 1-byte to 32-byte.

For the L7-filter case, which shows the

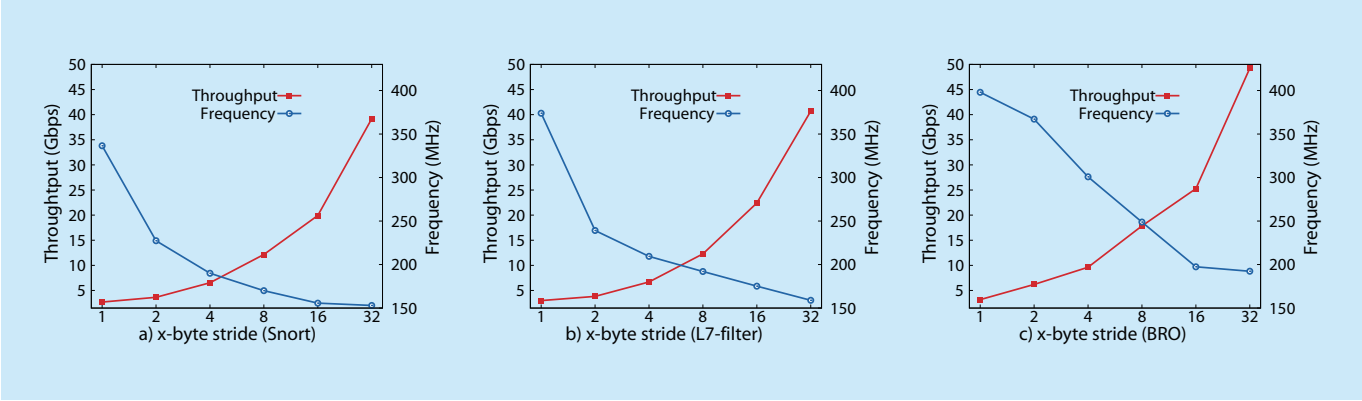

Fig.10 *Average throughput and operating frequency of RegEx matching on Link-NFA*

most notable trend, the transition growth rate is about 31.09 times that of the 1-byte stride Link-NFAs; for the Snort case, the growth rate is about 7.12; for the BRO case, the growth rate is about 3.28. The results verify our argument that the multiple occurrences of repeat meta-characters, such as star (*) and plus (+) in regular expression rules, indeed cause transition explosion problem in multi-stride NFA model. Moreover, Link-NFA could reduce transition growth by up to 2 orders of magnitude compared with multi-stride NFA in the case of 32-byte stride.

6.2 FPGA resources usage

Because the utilization rates of registers for all rulesets never exceed 2% and are irrelevant to stride length, we put our focus on LUT usage, results are shown in Table IV.

For 32-byte stride cases, rulesets Snort, L7-filter and BRO consume about 8.43%, 3.57% and 4.19% of the FPGA's LUTs. The results are affected by two reasons jointly. Firstly, the complexity of each RegEx has direct impact on the scale of the corresponding Link-NFA. Secondly, the bitmap based optimization method can greatly reduce hardware costs, and it is more effective when stride length is larger. The results indicate that Link-NFA can successfully control the hardware overheads for large stride situations over popular RegEx rulesets.

Block RAM based FIFOs are used to store intermediate results for some of Sub-NFAs which may report multiple results. For Snort, L7-filter and BRO rulesets, about 7.9%, 7.1% and 0.5% of total 16KB-BRAM resources are used respectively in the experiments. It is worth noting that the usage of FIFO is directly related to the RegEx rules rather than stride length.

Another important issue is the capability of accommodating RegEx rules. Because Virtex6 xc6vlx550t FPGA has 687360 LUTs/Registers, it is possible for it to support more than 1.5k Snort rules, 3k L7-filter rules and 3k Bro rules. If more advanced FPGAs are used, such as the Virtex7 serials, it is possible to triple the numbers.

6.3 Performance

The performance of Link-NFA based intrusion detection system, P, is decided by Eq. (1).

$$P = N \times Freq \quad (1)$$

Where N is the number of byte processed in each clock period, and $Freq$ is the operating frequency of the FPGA based system.

Fig.10 shows the average operating frequency and throughput for each ruleset with respect to stride length. As is expected, operating frequency for each ruleset declines as stride length increases because of increased processing complexity (e.g. comparing more bytes for each transition). The throughput for each ruleset increases monotonically following stride length. The number reaches the maximum when it comes to the 32-byte stride case. For Snort case, which shows a relatively low

Table V *Performance of current RegEx matching techniques and 32-byte Link-NFA*

Name	Performance	Platform
L7-filter[21]	< 1 Gbps	x86
Multi-stride NFA[14]	8.12 Gbps	FPGA
B-FSM[16]	18.4 Gbps	PowerEn Processor
GregEx[23]	25.6 Gbps	GPU
Lookabead[13]	34 Gbps	FPGA
Link-NFA(Snort)	39.1 Gbps	FPGA
Link-NFA(L7-filter)	40.6 Gbps	FPGA
Link-NFA(Bro)	49 Gbps	FPGA

Table VI *System Latency of Link-NFA and Snort Software*

Name	Latency	Platform
Snort Software[18]	240.83 us	x86
Link-NFA(Snort)	4.12 us	FPGA

performance, the throughput of the prototype can exceed 39 Gbps; for other cases, such as L7-filter, the throughputs exceed 40 Gbps. It is worth noting that the experiments are done on Virtex6 which is not the most state-of-the-art FPGA. If more advanced FPGAs are utilized, higher performance can be expected.

Table V compares the performance of Link-NFA and other RegEx matching methods. Based on the Xilinx Virtex6 FPGA, Link-NFA can achieve a throughput of more than 39 Gbps. It is higher than other methods, and can roughly satisfy the link speed of 40 Gbps. Link-NFA is more economic for low deployment and maintenance costs.

6.4 Power consumption

We use Tektronix PA1000 [22] to measure the power consumption of the prototype. The power consumption of the prototype is about 7.5 w while it is working normally. The reading is stable and steady during the experiment. Our method is clearly more applicable to vehicular environment from the perspective of power consumption because other hardware platforms usually consume more than 200 w, such as general purpose CPU and GPU.

6.5 Latency

In this section, we measure and compare the latencies of the prototype (implementing Snort ruleset) and the software method of Snort. The processing latency is obtained by injecting packets into an IDS system and measuring the input and output time. The Snort software is running at Linux operation system with Intel Core 2 Quad 2.83 GHz CPU and 4 gigabyte memory.

The results of the experiments are shown in Table VI. The system latency of Link-NFA prototype is about 4.12 us and is about one sixtieth part of that of current software method. Consider popular real-time operating systems such as VxWorks that usually consume about 100 us to respond an interrupt, the latency of our prototype is acceptable.

VII. Conclusions and Future Works

In this paper, we propose an intrusion detection system for IoV which can achieve high performance, low latency and acceptable energy consumption. Specifically, we focus on FPGA platform, and propose a novel data model based on current multi-stride NFA. We evaluate our scheme on a Xilinx FPGA based platform. Experiments show that the proposed system can achieve a throughput of more than 39 Gbps on existing FPGA platform and the total power consumption for the FPGA is about 7.5 w. Moreover, the processing latency of the prototype is about 4 us, and is about one sixtieth part of the latency of the popular software IDS. Our proposal could meet current need of IoV for IDS techniques.

But the proposed method has following drawbacks. First, Link-NFA is specified for FPGA, it is unlikely to be implemented in other hardware platform such as those based on GPU and CPU. Second, the number of NFAs in Link-NFA is related to RegEx rules, and it may grow rapidly in some extreme cases. Third, as is discussed in 5.1, Link-NFA utilizes FIFOs to temporarily store intermediate results. Overrun may occur if too many intermediate results are reported by Sub-NFAs.

To cope with issues mentioned above,

following works should be considered in the future. First, researches on actual IDS rules of IoV are necessary. Second, deploying Link-NFA in IoV environments and further evaluate its effectiveness. Third, more efficient scheme should be developed to prevent FIFO overruns.

Acknowledgements

This work is supported by the National Science Foundation of China under Grant No. 61402474 and by the Excellent Young Scholar Research Fund of Beijing Institute of Technology.

References

[1] YANG F, WANG S, LI J, LIU Z, and SUN Q, "An overview of Internet of Vehicles[J]", *China Communications*, 11(10): 1-15, 2014.

[2] FAN Cun-qun, WANG Shang-guang, GU Wen-zhe, SUN Qi-bo, YANG Fang-chun, "Enhanced-throughput multipath routing algorithm based on network coding in IoVs[J]", *Journal on Communications*, 34(Z1): 133-141, 2013.

[3] FAN Cun-qun, WANG Shang-guang, SUN Qi-bo, ZOU Hua, YANG Fang-chun, "IoV vertical handoff research based on Bayesian decision[J]", *Journal on Communications*, 34(7): 34-41, 2013.

[4] FAN Cun-qun, WANG Shang-guang, SUN Qi-bo, WANG Hong-man, ZHANG Guang-wei, YANG Fang-chun, "A Trust Evaluation Method of Sensors Based on Energy Monitoring[J]", *ACTA ELECTRONICA SINICA*, 41(4):646-651, 2013.

[5] Antonello Rafael, et al., "Design and optimizations for efficient regular expression matching in DPI systems[J]", *Proceedings of Computer Communications*, 61: 103-120, 2015.

[6] R. SMITH, C. ESTAN, and S. JHA, "XFA: Faster signature matching with extended automata[C]", *Proceedings of IEEE Symposium on Security and Privacy*, pp.187–201, IEEE, 2008.

[7] WANG X, XU Y, JIANG J, ORMOND O, LIU B, and WANG X., "StriFA: Stride finite automata for high-speed regular expression matching in network intrusion detection systems[J]", *Systems Journal*, IEEE, vol.7, no.3, pp.374–384, 2013.

[8] Shukla Surendra Kumar, et al., "A survey of approaches used in parallel architectures and multi-core processors[J]", For Performance Improvement. Proceedings of Progress in Systems Engineering, 2015: 537-545.

[9] Wang Kai, Zhe Fu, Xiaohe Hu, and Jun Li, "Practical regular expression matching free of scalability and performance barriers,"[J] Proceedings of Computer Communications 2014: 97-119.

[10] Vasiliadis Giorgos, et al., "GASPP: a GPU-accelerated stateful packet processing framework[C]", *Proceedings of 2014 USENIX conference on Annual Technical Conference*. Philadelphia: USENIX, 2014.321-332, 2014.

[11] FEITOZA SANTOS A., DE LACERDA FERNANDES S F, et al., "Multigigabit traffic identification on GPU[C]", *Proceedings of the first edition workshop on High performance and programmable networking*, pp.39–44, ACM, 2013.

[12] Chad R. Meiners, Jignesh Patel, Eric Norige, Alex X. Liu, and Eric Torng., "Fast Regular Expression Matching Using Small TCAM[J]", *IEEE/Acm Transactions On Networking*, Vol. 22, No. 1, February 2014.

[13] BANDO M, ARTAN N S, and CHAO H J., "Scalable lookahead regular expression detection system for deep packet inspection[J]", *Networking, IEEE/ACM Transactions on*, vol.20, no.3, pp.699–714, 2012.

[14] YAMAGAKI N, SIDHU R, and KAMIYA S., "High-speed regular expression matching engine using multi-character NFA[C]", *Proceedings of Field Programmable Logic and Applications*, pp.131–136, IEEE, 2008.

[15] Rathod, Prashantkumar M., Nilesh Marathe, and Amarsinh V. Vidhate, "A survey on Finite Automata based pattern matching techniques for network Intrusion Detection System (NIDS) [C]", *Advances in Electronics, Computers and Communications (ICAECC)*, IEEE, 2014.

[16] VAN LUNTEREN J and GUANELLA A., "Hardware-accelerated regular expression matching at multiple tens of gb/s[C]", *Proceedings of INFOCOM 2012*, pp.1737–1745, IEEE, 2012.

[17] BADRAN T F, AHMAD H H, and ABDELGAWAD M., "A reconfigurable multi-byte regular-expression matching architecture for signature-based intrusion detection[C]", *Proceedings of Information and Communication Technologies: From Theory to Applications (ICTTA 2008)*, pp.1–4, IEEE, 2008.

[18] Snort [EB/OL]. http://www.snort.org, 2015-08-29.

[19] L7-filter [EB/OL]. http://l7-filter.sourceforge.net/, 2015-08-29.

[20] Bro [EB/OL].https://www.bro.org/, 2015-08-29.

[21] FU Wen-liang, SONG Tian, ZHOU Zhou, "RocketTC: A high throughput traffic classification architecture on FPGA[J]", *Chinese Journal of Computers*, 37(2): 414-422, 2014.

[22] http://www.tek.com/power-analyzer/pa1000, 2015-08-29.

[23] WANG L, CHEN S, TANG Y, and SU J., "Gregex: GPU based high speed regular expression matching engine[C]". *Proceedings of Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pp.366–370, IEEE, 2011.

Biographies

Wenliang Fu, is currently a Ph. D. student of School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He has been an IEEE student member since 2012. He was a visiting student to the Department of Computer Science and Engineering at Washington University at St. Louis from 2012 to 2013. His research interests include network security, energy efficient networking, future Internet architecture, etc.

Xin Xin, is currently an assistant professor in School of Computer Science, Beijing Institute of Technology. He received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, in 2006 and 2008, respectively, and the Ph.D degree in computer science and engineering from the Chinese University of Hong Kong in 2011. His research interests include data mining, machine learning, etc. *The corresponding author, E-mail: xxin@bit.edu.cn.

Ping Guo, received the master degree from Beijing University, China, in 1983. Then, he received the doctor degree from Chinese University of Hong Kong, China, in 2001. He is currently a professor in School of Computer Science and Technology, Beijing Institute of Technology, China. His research interests include computing intelligence and its application.

Zhou Zhou, received the master degree and doctor degree from Beijing Institute of Technology, China, in 2012. He is currently an associate professor in Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. His research interests include network security, high performance string matching, multi-core processing system architecture, etc.