Effective Hash-Based Filtering Architecture for High-throughput Regular-Expression Matching

Hayato Yamaki, Yasutsugu Nagatomi, and Hiroaki Nishi

***Abstract*—Regular-expression is widely used in various network applications, such as a network intrusion detection system (NIDS). However, existing regular-expression process does not meet required throughput in core routers because most of NIDSs have been implemented by software. In particular, a pattern matching function, a part of regular-expression processing, requires comparatively longer processing time. Huge amounts of patterns are used for this pattern matching in NIDS. Therefore, high-throughput processing modules are strongly required in order to execute a large number of complicated regular-expression. In this paper, a simple but effective hash-based architecture of pattern filtering with special entries is discussed, and it enables to achieve a large number of patterns matching.**

***Index Terms*—Network processor, regular-expression, pattern matching, hash**

I. Introduction

Regular-expression is used in various areas such as Network Intrusion Detection System (NIDS) and content-based analysis. For example of NIDS, Snort [1] and Bro [2] are representative NIDS and they utilize regular-expression stored in pattern files. However, their processing throughput is still very narrow and it deteriorates network service quality. Main reasons of this slow performance is enormous amounts of patterns used in a matching process includes complex regular-expressions.

As an enhancement of this content-based analysis, XML based routing [3]–[5] is studied. This XML-based router uses the information in a payload of a packet for its routing. Service-oriented router [6] is a leading edge project that proposes a special router for enhancing network services and user experience. XML router and Service-oriented router require quick update of regular-expression patterns, especially when it is used in a highly real time service such as action-history analysis and research of real-time popularity [6]. To meet the requirements for the high-speed pattern matching, hardware architecture for processing the enormous amounts of patterns effectively is needed.

Some works have been conducted to propose the accelerating methods of regular-expression processing [7]–[10]. All of them utilize Deterministic Finite Automaton (DFA)-based or Nondeterministic Finite Automaton (NFA)-based device, which has reconfigurable architecture. Aho- Corasick method is a fundamental algorithm of these proposed accelerators. In such DFA/NFA-based systems, circuits for state transition machine are frequently rewritten when the patterns of regular-expression are updated. The DFA/NFA-based processing of regular-expression takes large processing cost if it is used in a system, which requires frequent updates of pattern. In this paper, we propose a mechanism for narrowing down the cost of pattern matching included in regular-expressions processing. This mechanism is composed of three different functions effectively. This mechanism can reduce the area cost because of its simple structure and enhance the regular-expression performance and its updating performance.

Manuscript received May 10, 2012; revised June 12, 2012. This work was supported partially supported by National Institute of Information and Communications Technology (NICT) and a Grant-in-Aid for Scientific Research (C) (22500069).

H. Yamaki, Y. Nagatomi, and H. Nishi are with the Graduate School of Science and Technology, Keio University, Yokohama, Kanagawa, 223-8522, Japan (e-mail: yamaki@west.sd.keio.ac.jp, yasu@west.sd.keio.ac.jp, west@sd.keio.ac.jp)

II. Related Work

Because of the increase in communication traffic over the Internet, a regular-expression, which depends on total throughput of NIDS, is becoming more important. To meet the requirements of the efficient regular-expression processing, several studies have achieved, especially proposes of hardware accelerators are key solution to attain high-throughput regular-expression processing.

Some proposes use NFA-based architecture [8] in order to achieve parallel processing. Although the NFA-based hardware can process many patterns at the same time, it requires frequent reconfiguration of circuits for updating rules of NIDS and it makes a lot of overhead. Titan IC System and LSI corp. have manufactured NFA/DFA–based processor optimized for regular-expression processing [11], [12].

Meanwhile, Z. Baker has proposed DFA-based architecture in order to reduce the overhead caused by frequent update of patterns. The DFA-based hardware can update pattern files only by updating memory. However, Baker's DFA-based architecture has a limitation in its scalability because its next transit state can be set uniquely. This feature increases the number of states exponentially if it is used for complex regular-expression processing. Therefore, hardware architectures which only use such DFA-based architecture are not suited for processing enormous amount of patterns including complex regular-expressions.

As another approach, hash-based pattern matching is proposed. Hash-based pattern matching is appropriate for processing multiple patterns because hash can reduce the number of candidate patterns efficiently. Some of these works utilizes Bloom filter [13]. Bloom filter is a space-efficient probabilistic data structure and it requires many hash functions. There are two methods to prepare memory for hash. One is multiple access method that uses only one hash memory and asserts multiple accesses to the memory. This method costs a lot of processing steps of memory accesses. The other method uses multiple hash memory for permitting pipe-line processing. This method costs the amount of hardware resources. These defects of Bloom filter are pointed out by G. Papadopoulos [14].

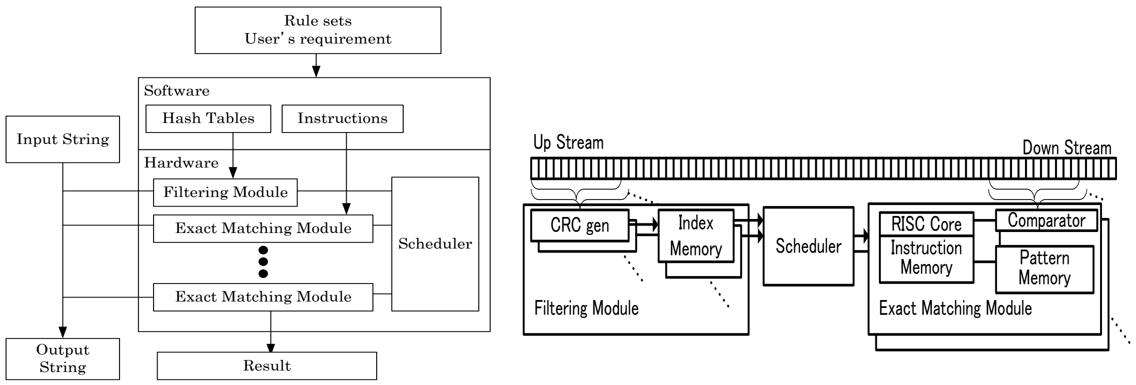

Fig. 1. (a) Structure of regular-expression processor; (b) Hardware architecture

Additionally, perfect hashing function is studied as one of the approaches to hash-based pattern matching [15]. Perfect hash function enables to distribute hash collisions, and ideally it can prevent hash collisions. Although perfect hash function is an efficient method to narrow down the number of candidates, the update cost is higher than other methods. This method cannot be used in the system which requires frequently reconfiguration for rebuilding hash tree. As an enhancement of perfect hashing function, HashMem, a hash-based architecture, is proposed [14]. This architecture costs a lot of hardware because hash table should be prepared for every length of pattern. Another HashMem is also proposed, which reduces hardware cost by dividing long pattern into short one. However, it needs other circuits for aggregating divided hash values and for matching check and could not solve the problem of hardware cost enough.

Furthermore, commercial processor for efficient processing of character strings is manufactured for personal computer. Intel Core i7 has instruction set called as String & Text New Instructions (STTNI) as an extension of SSE (Streaming SIMD Extensions) 4.2. The instruction set is for accelerating analysis of XML and it enables efficient processing of regular-expressions [16]. However, the processor is not suited for processing the enormous amount of patterns because it does not have any special hardware for high-speed processing of patterns.

III. Architecture of Proposed Pre-Filtering Hash Module for Regular-Expression Processor

Our idea is to design special hash-table according to a target application. To achieve effective distribution of hash-memory accesses, preprocess of network traffic and statistical analysis were conducted. As first, proposed architecture of regular-expression processor is shown in Fig. 1(a). User-defined rule sets, described by using a regular-expression, are compiled in software layer. The software outputs entries of Hash tables and the instructions for the exact matching modules. The hash table selects one instruction from instructions memory. Scheduler, filtering module and exact matching modules are implemented by using hardware, because they require high-throughput processing. To accomplish the high-throughput processing, SoC (System on Chip) architecture is indispensable for a network processor. Fig. 1(b) shows the hardware architecture of the regular-expression processor from the viewpoint of its processing order. Exact matching modules are a micro controller based string matching hardware. Scheduler is resource management hardware of the exact matching modules. The following paragraph explains the filtering module.

The filtering module determines potential matches between lexical string of packet stream and target string by using a hash table. This module consists of CRC generator and index memory, as shown in Figure 1(b). CRC generator makes a hash of input string. The hash value is used as an address of index memory. Each entry of the index memory has two types of data. One is a Boolean value that implies that additional process of exact match is required. The other is an address of instruction memory that points the start address of dedicated exact-match-handling program stored in the exact matching module. When the Boolean value indicates that the additional exact match is required, the start address of instruction memory is sent to the exact matching module.

These CRC generators and index memory become big because the set of CRC generator and index memory is separately implemented according to the pattern length of target string. One CRC generator and index memory can handle only one length of string. For reducing the hardware cost of this hash memory, three different methods are proposed as following.

A. Case-Insensitive Approach

When supporting both case sensitive and case insensitive pattern matching, two different CRC generator and hash memory are required. However, 81% of the snort pattern sets specifies the case-insensitive switch as a contents option. For this reason, enough filtering performance can be achieved only when case-insensitive function is implemented in filtering module. In addition, accuracy of detection is not eliminated because the case-sensitive matching can be followed after the case-insensitive matching.

B. Shortening the length of Hashing

Some of ASCII character patterns in Snort exceed over 100 characters. If CRC module in filtering module is implemented individually as different hash modules of 1 to 100 characters, the large size of CRC modules and Index memory may cause a problem when implemented. The main purpose of the filtering module is to reduce the processing cost of following exact matching module, and it is enough to find a possibility of potential match in the filtering module. In this meaning, full-length pattern matching is not required because to check first limited characters is enough to find the possibility of a potential match. As shown in Fig. 2, matching pattern that has longer pattern length has smaller number of matches. This proofs that long matching pattern is not significant in reducing processing cost of the exact matching module. For this reason, the filtering module can distinguish the potential match by using a hash function that only checks first several characters.

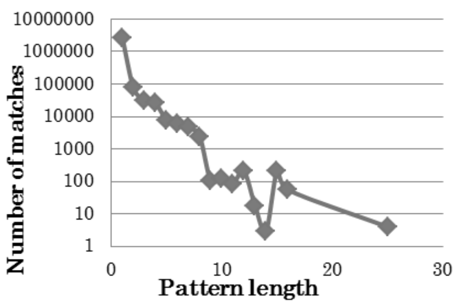

Fig. 2. Relation between pattern length and the number of match patterns

C. Use of Associated Options

Snort allows using several options in its patterns, for example, port number, IP address, and search range of pattern matching, and other parameters derived from the Internet traffic. The filtering module can eliminate the number of matches if the module effectively utilizes these options. Since the number of kicks of processes at an exact matching module can decrease, the number of the exact matching modules can be reduced. To use these options, the entries of index memory should be extended. The architecture of the improved filtering module is shown in Fig. 3. The difference between the original filtering module and the improved filtering module is a behavior of a header analyzer. It extracts information of a destination port, destination IP, etc. In particular, the location of processing should be counted in order to use the search range option that indicates the end of the matching process. By comparing the information including the counter, port number and IP address with the entry information of the corresponding index memory, advanced filtering process on the exact matching module can be achieved according to the previously calculated bit entries. Proposed filtering module also enables effective resource distribution of the exact matching module at the scheduler.

This sophisticated hash memory based filtering module can eliminate the number of processing modules of exact matching. If the hash memory can reduce the cost of processing on the exact matching module up to 80%, the required number of processing module becomes 20% of originally required number. Though the area cost of this hash memory becomes large when the hash table has all the information of the counter, port number, IP address, and other information included in Snort option, the method to reduce the hash table is shown next.

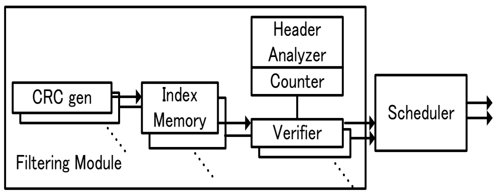

Fig. 3. Improved filtering module block diagram

D. Compression of Managing Options in Hash Memory

Reduction of the size of hash memory, namely the selection of effective set of option and the reduction of managing size of the option, are discussed by using snortrules-snapshot-2.8.

1) Selection of Snort options

In this evaluation, general ASCII code based on 4,881 significant snort patterns without binary search were selected. These pattern sets contain the option of destination port number and IP address. The percentage of a pattern sets that specify destination port was 81%. The percentage of a pattern sets that specify destination address was 51%. These two options were the most dominant options in the pattern sets. In addition, the search range option is also useful because it can terminate the matching process in the middle of processing. We select these three options as the most dominant option in the snort rule sets.

2) Compression of bit-length

Even if the filtering module selects three options to manage in a hash memory, it still requires large memory size. 32bits for IP address, 16bits for port number, and 16bits for length are needed. Totally, additional 64bits entry is required and it enlarges the size of hash memory drastically. We analyzed the most effective compression of the information by analyzing each option of Snort.

To reduce the bit length of port number option, the port number of Snort option was analyzed. The analysis result in Table I indicates that the information of destination port number can be compressed to 2-bit information which expresses four groups of port number, such as port 80, other port, any port and false. The false group means mismatch and such mismatched information is excluded from the target pattern. Though it is desirable to divide the probability into 33% for each when considering the distribution of hash memory, port 80 (www) is dominant and this division is understandable in this situation.

To reduce the bit length of destination IP, the existing probability of destination IP in Snort pattern was analyzed in Table II. It is enough to use only one-bit information that indicates destination address is included in internal network or in any other network and is almost perfectly divides the probability into half and half.

Finally, to reduce the bit length of starting point of a search in a data stream, we analyzed the frequency of usage of the starting point option in Snort. To eliminate the number of candidate that kicks the process of the exact matching module, a pattern with comparatively shorter length should be filtered. It means small number of starting point should be remained in this reduction process. This is also shown in Fig. 3 and it proofs that a pattern with shorter length is frequently accessed. According to this fact, the dominant starting point in such a sort patterns is shown in Table III. This result indicates that the bit length of starting point can be compressed to 2-bit information. The existing probability of Start from 1st character or start from 2-11th character is small but is dominant in network traffic as described. Totally, Snort option can be expressed by using 5-bit information in an entry of hash table.

TABLE I: DOMINANT DESTINATION PORT NUMBER

Destination Port	Ratio
80	57%
Other	24%
Any	19%

TABLE II: DOMINANT DESTINATION IP

Destination IP	Ratio
Internal network	51%
Any	49%

TABLE III: DOMINANT STARTING POINT IN SHORT PATTERNS

Starting point of search scope	Ratio
Any	50%
Start from 40th character	40%
Start from 1st character	8%
Start from 2-11th character	1%

IV. EVALUATION OF PROCESSING COST

In this section, we evaluate the number of matches in order to validate the efficiency of the proposed method and confirm the result of hardware cost reduction. As a packet stream, we used 500 streams (13MB) captured on the gateway router of our laboratory. The specifications of destination port number and destination address of the stream are shown in Table IV and Table VII.

TABLE IV: SPECIFICATION OF A DESTINATION IP

Destination IP	Ratio
Internal network	30%
External network	70%

TABLE V: SPECIFICATION OF A DESTINATION PORT

Destination Port	Ratio
80	97%
The other port	3%

A. Case-Insensitive Approach

Result of case-insensitive approach is shown in Fig. 4. In figure, y-axis means the improving rate of the proposed method compared of case-insensitive CRC module with the original method that has both case-sensitive and case-insensitive CRC module. The rate of improvement is limited at most about 20%. The overall incensement of the number of matches is about 1.6%.

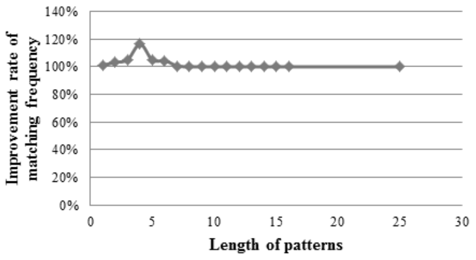

Fig. 4. Improvement rate of match frequency

B. Shortening the length of Hashing

The relation between the length of hashing pattern and the number of process kicks in exact matching module is shown in Fig. 5. Y-axis denotes the improving rate of the proposed method compared with the original method in the number of kicks. If the hashing length is restricted to five characters, the number of matches will increase about 1.4% of the cases, which did not limit the hashing length.

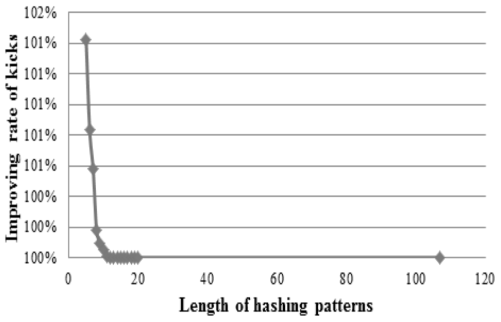

Fig. 5. Relation between hashing pattern length and the number of kicks

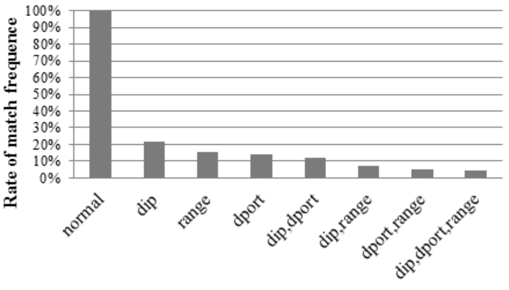

Fig. 6. Relation between match frequency and combination of options

C. Use of Associated Options

The relation between the number of process kicks in exact matching module and combination of options is shown in Fig. 6. Y-axis denotes the improving rate of proposed the method compared with the original method in the number of kicks as well as Fig. 5. Normal means the case when an option is not used. Dip specifies a destination IP address. Range specifies the starting point option of Snort. Dport specifies a destination port. The number of kicks is improved and it can be eliminated to the 20% to 4% of original method.

TABLE VI: SELECTION OF BIT WIDTH OF CRC

# of patterns	1	2	3-5	6-11	12-23	24-47	48-95	96-191	192-383	384-767	768-1535	1536-3071	3072-
Bit widths	4	5	6	7	8	9	10	11	12	13	14	15	16

TABLE VII: IMPLEMENTATION RESULT

Design	Fluctuations in the number of kicks	CRC generator [μm^2]	Index Memory Size [kbyte]	Area Cost [μm^2]	Performance
Original	100%	86,373	184	161,216	1
+insensitive	102%	43,187	92	80,609	2.0
+insensitive +prefix (5)	103%	1,128	100	41,804	3.7
+insensitive +option	4%	43,187	129	95,659	42
+insensitive +prefix (5) +option	5%	1,128	140	58,074	56

$$\begin{aligned} &Performance = 1/\left(Area\ cost \times \#\ of\ kicks \times Performance\ of\ original\ design\right) \\ &Performance\ of\ original\ design = 1/\left(Area\ cost \times \#\ of\ kicks\right) \end{aligned} \qquad (1)$$

V. EVALUATION OF HARDWARE COST

This section focuses on the hardware cost of CRC generator and Index memory. In this evaluation, we designed the proposed method by using the verilog Hardware Description Language, and we use Synopsys Design Compiler 2008.09 as a logic synthesis tool. In our design, we used the 45-nm process technology by synthesis library called freePDK [17]. As a pattern length, more than 2-byte length of pattern is used. In this evaluation, the hardware cost of CRC generator and index memory in each technique is compared. The word length of an index memory is set to 10 bits (1bit Boolean + 9bit for the address of instruction) in the original method that does not use Snort rule option. By contrast, It becomes 14 bits (5bit for Boolean and options + 9bit for instruction memory address) in proposed method that use the rule option. As a condition of implementation of CRC generator in filtering module, the bit length of CRC pattern is altered according to the number of pattern as shown in Table VII. This is because the effectiveness of hash memory depends on the hit rate of hash memory and the hit rate is affected by the size of addressing bus calculated by the CRC.

Table VIII shows the performance and area cost of CRC generator and index memory. Area cost is evaluated on both CRC generator and index memory. The performance is evaluated by using efficiency per area cost, and it is calculated by using the equation (1).

We compared five different configurations of hash memory designs and the number of CRC generators. Original configuration is case sensitive. Any hardware reduction techniques are not given. In +insensitive configuration, the number of CRC generator and index memories is reduced by ignoring the case-sensitive operation in the filtering module. In +prefix (5) configuration, the number of CRC generator and index memories is reduced by limiting the length of pattern up to five characters. In +option configuration, three options of Snort patterns are considered and stored in the index memory. Filter performance is improved dramatically.

The performance of proposed methods is 2.0–56 times better than the performance of original method as shown in Table 7.

VI. CONCLUSION

In this paper, we described and compared three proposed techniques to reduce the hardware cost of string searching function in a regular-expression processing. Case-insensitive approach contributes to reduce the low area cost of hash memory, and it does not degrade the rate of miss filtering much. Shortening of hashing pattern length can reduce the number of CRC generator and index memory. Use of associated options improves filter performance dramatically and prohibits the incensement of memory size. By using proposed sophisticated hash memory architecture, it can reduce the processing cost and resource cost of processor based exact matching.

REFERENCES

[1] SNORT. [Online]. Available: http://www.snort.org/

[2] BRO. [Online]. Available: http://www.bro-ids.org/

[3] A. C. Snoeren, K. Conley and D. K. Gifford, "Mesh based content routing using xml," In *Proc. 18th ACM Symposium on Operating System Principles*, pp. 160-173, 2001.

[4] A. Carzaniga, M. J. Rutherford, and A. L. Wolf, "A routing scheme for content-based networking," In *Proc. 23rd Annu. Joint Conf. Computer and Communications Societies*, pp. 918-928, 2004.

[5] J. Moscola, Y. H. Cho, and J. W. Lockwood, "A reconfigurable architecture for multi-gigabit speed content-based routing," In *Proc High-Performance Interconnects (HOTI)* 2006, pp. 61-66, 2006.

[6] K. Inoue, D. Akashi, M. Koibuchi, H. Kawashima, and H. Nishi, "Semantic router using data stream to enrich services," In *Proc. 3rdInternational Conference on Future Internet CFI 2008 Seoul*, pp. 20-23, 2008.

[7] Z. Baker, H. J. Jung, and V. Prasanna, "Regular-expression Software Deceleration for Intrusion Detection Systems," In *Proc. International Conference on Field-Programmable Logic and its Applications (FPL) 2006*, pp. 418-425, 2006.

[8] I. Sourdis, J. Bispo, J. M. Cardoso, and S. Vassiliadis, "Regular expression matching in reconfigurable Hard-ware," *Journal of Signal Processing Systems*, vol. 51, no. 1, 2007.

[9] B. C. Brodie, D. E. Taylor, and R. K. Cytron, "A scalable architecture for high-throughput regular-expression pattern matching," In *Proc. International Symposium on Computer Architecture (ISCA) 2006*, pp. 191-202, 2006.

[10] F. Yu, Z. Chen, Y. Diao, T. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," *Technical Report UCB/EECS-2006-76*, EECS Department, University of California, Berkeley, 2006.

[11] Regular Expression Processor, titanicsystems [Online]. Available: http://www.titanicsystems.com/products/item/1/regular-expression-processor-rxp/

[12] Tarari Content Processors, [Online]. Available: http://www.lsi.com/networking_home/networking_products/tarari_content_processors/index.html

[13] M. Attig, S. Dharmapurikar, and J. Lockwood, "Implementation results of bloom filters for string matching," In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machined (FCCM)*, 2004.

[14] G. Papadopoulos and D. Pnevmatikatos, "Hashing + Memory = Low Cost, exact pattern matching," In *Proc. International Conference on Field Progrramable Logic and Applications*, pp. 39-44, 2005.

[15] I. Sourdis, D. Pnevmatikatos, S. Wong, and S. Vassiliadis, "A reconfigurable perfect-hashing scheme for packet inspection," In *Proc. International Conference on Field Programmable Logic andApplications*, pp. 644-647, 2005

[16] String Text New Instructions, .cs. uml [Online]. Available: http://www.cs.uml.edu/~bill/cs515/Intel_Nehalem_Processor.pdf

[17] Free PDK, [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK

Hayato Yamaki received his B.E. degrees from Keio University, Japan, in 2011, respectively. He researches about cache architecture of network processor and is interested in hash.

Yasutsugu Nagatomi received his B.E., and M.E. degrees from Keio University, Japan, in 2009 and 2011, respectively. He researched about network architecture of router and was interested in hash.

Hiroaki Nishi received his B.E., M.E., and Ph.D. degrees from Keio University, Japan, in 1994, 1996, and 1999, respectively. He has been selected as fellowships from Japan Society for Promotion of Science from 1996 to 1999. He was a researcher in Real World Computing Partnership from in 1999 and researched high performance network for cluster computers. Since 2002, he was a researcher in the Central Research Laboratory, Network Platform Department, Hitachi Ltd., and he led new generation backbone router project. He was Lecturer and Assistant Professor in Department of System Design Engineering, Keio University in 2003 and 2006, respectively. Now, he is Associate Professor of Keio University since 2007 and Visiting Associate Professor of National Institute of Informatics (NII), Japan since 2010. Prof. Nishi is a member of IEEE, Institute of Electrical Engineers of Japan, Architectural Institute of Japan, Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineers and the Society of Instrument and Control Engineers. He received Network System Research Award in 2003 from IEICE, Award for the outstanding contribution IEEE ICS Advanced Motion Control Workshop in 2004, Best Paper Award of FANAC FA Robot financial group in 1007 and 2008, Best Paper Award of IPSJ Ubiquitous Computing Society in 2010, Best Presentation Award of IPSJ SLDM society in 2010 and Computer System Award of IPSJ in 2011. The main theme of his current research is in building of the total network system including development of hardware and software architecture.