JGN II R&D Project : Research and Development of Surrounding Computing Technology

JGN II R&D Project

Research and Development of Surrounding Computing Technology

FUKUMOTO Masahiro*, SHIMAMURA Kazunori, IWATA Makoto
HAMAMURA Masanori, SAKAI Keiichi, MENDORI Takahiko
TSUZUKI Shinji, YAMAGUCHI Takumi and HAYASHI Hideki

Kochi JGN II Research Center 185 Miyanokuchi, Tosayamada-cho,
Kami-gun, Kochi 782-8502, Japan

E-mail : * fukumoto.masahiro@kochi-tech.ac.jp

要約：より快適な情報環境を実現するためには多様なサービスを必要に応じて提供することが重要であり、特に映像など 負荷が非常に大きな情報を処理するためには、ネットワーク上に分散する資源を自由に活用するための技術が求められる。ユビキタス環境では、遠く離れた複数地点からデータを収集しリアルタイムで処理するといったこともできるが、データの流れを考えたときデータを集約する必然性は そもそもなく、分散処理することが自然である。そこで、ネットワークや端末の機能を意識せずに、ネットワーク上の計算資源やDB資源を自由に活用できる、進化するユビキタス環境である、サラウンディング・ コンピューティング環境を確立する。本稿では、サラウンディング・コンピューティング技術の基礎であるユビキタス環境で有用なデータ駆動プロセッサによるファイアウォールと情報再現に適した信号処理方式について述べている。

Abstract : For comfortable information networking, it is necessary to provision variety of services for responding the requirements and to flexibly use of the distributed resources。In the "ubiquitous" environment,the distributed processing is natural to push data for realtime application。The purpose of this research is to establish the "surrounding computing technology",which is evolution of the ubiquitous environment. In this paper,an embedded data-driven firewall processor and a signal processing method that is suitable for an information reproduction are proposed.

1. Introduction

JGN II is an open testbed network environment for research and development, which was previously operated by JGN (Japan Gigabit Network : Gigabit network for R&D) from April 1999 to March 2004, and expanded by the National Institute of Information and Communications Technology (hereinafter NICT) as a new ultra-high-speed test bed networks for

R&D collaboration between industry, academia, government with the aim of promoting a broad spectrum of research and development projects, ranging from fundamental core research and development to advanced experimental testing, in areas including the advancement of network-related technologies for the next generation and diverse range of network application technologies.

Further, seven own research centers for research and development (hereinafter Research Center) are also being provided in collaboration with JGN II.

The seven Research Centers collaborating with JGN II are conducting the research and development in relation to the following four themes regarding Research and Development on Advanced Networks and Application Technologies.

1. Highly Reliable Core Network Technology
2. Access Network Technology
3. Grid Technology
4. Platform and Application Technology

Kochi JGN II Research Center has been established since April 2004, as a center of the ordinary research in the JGNII project, the R&D for "Platform and Application Technology" is being carried out. Specifically, we are focusing on the "Surrounding Computing Technology".

2. Design Concept of An Embedded Data-Driven Firewall Processor

With rapid advancement of information networking technology, various networked systems/ devices are permeating among our daily lives and offices. It can be said that surrounding networking/computing environment will be coming soon around us. In order to keep this comfortable environment robust and safe against malicious or legitimate intruders and viruses, there are developing many security products such as firewall systems, network intrusion detection systems, virus protection systems, and so on [1][2]. Since most of firewall systems monitor packets on the network wire, they cannot completely prevent all accidents and attacks especially among local hosts. On the other hand, software-based security solutions like [3] can be rendered useless if the OS is exploited, compromising the computer and potentially the internal network.

This paper describes the design concept of hardware-based firewall processor embedded inside to local hosts in order to eliminate the possibility of internal attack from behind the perimeter firewall and then it illustrates some experimental results in our feasibility study. This kind of embedded firewall processor is required to be robust, secure and evolutional even against a newly-discovered attack, as well as to be low power consumption and high performance. Thus, our firewall processor was decided to be designed by introducing the self-timed super-pipelined data-driven chip-multiprocessor architecture [4] incorporating a specific instruction set for firewall functions. The data-driven principle provides us natural multiprocessing capability without any process scheduling or complex interrupt handling. Furthermore, the self-timed pipeline scheme serves us flexible pipeline processing capability for high-speed packet stream and flexible power saving feature.

The first prototype system implemented for our feasibility study is equipped with a packet classification, a stateful packet inspection for the layer 4 protocols, and a simple URL filtering. Since every function is realized by super-pipelined algorithms, it can be executed in highly-parallel on the data-driven chip-multiprocessor. Preliminary evaluation results in our feasibility study show that our firewall processor potentially operates at over 100 M b/s in wire-speed, even if it is equipped with a single processor. Furthermore, since scalable performance increasing along with the number of processors is also observed, a single chip incorporating 10 processors could be expected to operate over 1 G b/s packet stream.

2.1 Design Considerations on Embedded Firewall

2.1.1 Specific Features of Embedded Firewall

Recently mobile users, telecommuters and business-to-business extranets are significantly increasing. Those are no longer protected by only normal perimeter firewalls because they can directly access to the protected intranet using dial-in, P2P, encrypted application traffic. Thus, those hosts might be potentially unwitting insiders if they are infected with some virus or worm at the outside of the protected network. These trends lead necessity of an embedded firewall attached to the mobile hosts such as mobile PC, PDA, mobile phone, etc. The embedded firewall is placed on a network interface card of a host computer and filters Internet Protocol (IP) traffic to and from the host. The embedded firewall is tamper-resistant because it is independent of the host's operating system. The basic concept of an embedded firewall was originally proposed by C. Payne et al. [5].

The basic functions of the embedded firewall are shown in Fig.1. At first, an incoming packet to the host is stored to the packet buffer and its IP header and TCP/UDP header are transferred to a dynamic packet filtering function. In this case, its destination IP address must be same as the IP address of the host except for multicast addresses. In case of the outgoing packet from the host, its source IP address must be same as that of the host as long as the host is not a willing/ unwilling intruder. This means that filtering cost of one of IP address fields can be reduced. In the dynamic packet filtering function, the TCP header is identified whether it belongs to a new connection or not. If it will establish a new connection, a packet classification module (classifier) checks it using a filtering rule database. If it belongs to the existing connections managed in the firewall, the packet is examined by a stateful packet inspection (SPI) whether it will cause a legitimate state-transition of the connection. Although UDP is a kind of connectionless protocol, a virtual connection for the UDP stream can be assumed by the pair of IP addresses and port numbers. Therefore, even for the UDP packets, a SPI is useful to make the host more secure. After the stateful inspection of the layer-4 packet header, only the acceptable packet can be passed to an application layer filtering (AF) to inspect its contents such as URL or e-mail attached file. Finally, the forwarding packet is determined and forwarded from the packet buffer.

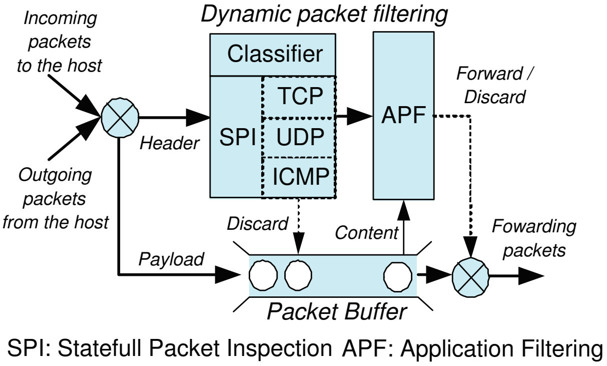


Figure 1 Basic Function of the Embedded Firewall.

In order to accept high-speed packet stream at the embedded firewall, pipelined parallel algorithms for those functions must be investigated and further their efficient hardware platform should be established. So, our final objective is to realize a small, low-power, high-performance embedded firewall processor capable to be employed into the CF card or mobile phone.

2.1.2 Data-Driven Processing Paradigm

The data-driven computation paradigm allows us to represent system functions in natural way, including both software and hardware portions seamlessly. This advantage leads to improve deign productivity of the system-on-chip (SoC) LSI's in contrast to the conventional Von Neumann computation. Furthermore, the data-driven execution control significantly fits the self-timed super-pipeline circuit. It is necessary for realizing high-performance, low-power, and easy-to-design SoC systems even if it is realized by a deep sub-micron semiconductor process. These excellent features have been proved by successfully developing a self-timed super-pipelined data-driven multimedia processor (DDMP) chip in which ten processors are interconnected to each other via an on-chip packet router [4].

Furthermore, in order to apply its high-speed stream processing capability to Internet routers or IP forwarding engines, a multi-protocol data-driven network processor (DDNP) is currently developed. DDNP can accept both IPv4 and IPv6 packet streams without any process scheduling so that it can process both protocols simultaneously at around 2 G b/s. Furthermore, we proposed a super-pipelined IP lookup scheme on DDNP by introducing some compound lookup instructions. This scheme can search a forwarding path from a large routing table (100 K routing entries) at over 50 M IP packets/s [6] and furthermore classify layer 4 packets at around 12 M IP packets/s [7]. In this paper, design concept of an embedded data-driven firewall system is proposed as one of applications of DDNP.

2.2 Architecture of Embedded Data-Driven Firewall

Essential data-dependencies among functions in the embedded firewall can be expressed as shown in Fig.2. If the maximum dataflow based on these data-dependencies is always kept on DDNP, many architectural advantages of DDNP can be fully utilized. That is, the dataflow diagram in Fig.2 shows us the following key points in designing the data-driven firewall system.

(a) Efficient dynamic multiprocessing, i.e., process creation, execution, and deletion, including state-transition process.

(b) Parallel implementation of all filtering functions, i.e., classification, stateful packet inspection (SPI), and application layer filtering (APF).

(c) High speed packet buffering mechanism which is capable to access off-chip memory modules, e.g., SDRAM, DDR-RAM, etc.

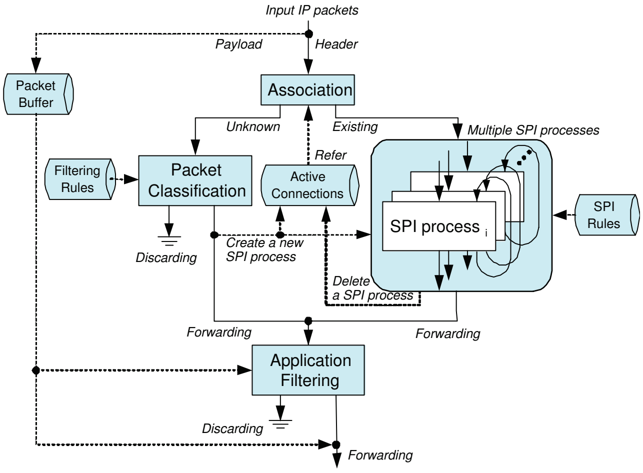


Figure 2 Basic Dataflow of the Embedded Firewall System.

2.2.1 Software Structure

(a) Dynamic multiprocessing based on the data-driven computation

Using the tag identifier of the dynamic data-driven computation principle, active data belonging to each process can be identified by its tag identifier. Thus, the data-driven processor can carry out multiple instances of the same program in the highly-parallel manner, even if they are state-transition processes. The state-transition process such as SPI is basically a sort of history-sensitive process in which the next state and one of selective functions are determined by only the current state and input data. Each current state of the processes can be represented as a tagged token data without storing it into the memory. Therefore, they can be executed in parallel same as normal functional processes under the data-driven firing rule. However, in the case of multiple state-transition processes, the processor must associate every input packet header with one of the existing connections at the firewall and then process it with its corresponding current state. This kind of association is essential for the dynamic data-driven processor to create, execute and delete multiprocessing instances associated with the tag identifiers. There is no existing data-driven processor that supports it.

The association process for the SPI must accept the following queries; *refer* request from every input IP packet and *update* request whenever connections are established or finished. This leads a quick search scheme for an associative memory. In our implementation, the associative memory function is realized by introducing a hybrid scheme with software hashing and small content-addressable memory (CAM) [8]. Since large CAM modules are still expensive due to complex memory cells and WTA circuits, our CAM is used to store the conflicted data by the hashing module so as to reduce the capacity of CAM. Particularly, it is more useful for the embedded firewalls because distribution of IP address of them is smaller rather than that of the perimeter firewalls. In fact, preliminary evaluation result of the proposed scheme indicates that the scheme reduces the search time by 30%–90% in comparison to the software implementation without CAM.

(b) Parallel implementation of filtering functions

As described in **2.1,1**, the embedded firewall is required to support static packet classification, stateful packet inspection, and application layer filtering. As for the static packet filtering, our super-pipelined packet classification scheme can be applied. Although the level compressed (LC) trie structure introduced in our classification needs little higher update cost in place of reducing the search space, the classification rule in the embedded firewall does not often updated usually. Therefore, it is introduced for the static packet filtering for the embedded firewall. As discussed in **2.1.1**, classification fields of the layer 4 packets can be reduced to 3 fields; one IP address and two TCP/UDP port numbers. Other fields can be checked using a simple exact-matching method even if they are needed. Preliminary evaluation estimates that the filtering performance could be from 3.5 M to 5.0 M IP packets/sec. and its required memory space could be 8 k bytes when the classification rule size is assumed as around 2 k entries [9].

Secondly, SPI is required to detect malicious packets by simulating state transition of the connection and checking their TCP flags with the current state of the connection. Normally, SPI for the embedded firewall should operate several thousands connections at the same time. Our SPI implementation works at 329 k IP packets/sec on a single processor [8]. Off course, the performance can be scaleablly improved in proportion to the number of available processors because of the elegant multiprocessing capability of DDNP.

Thirdly, application filtering is required to support various protocols such as http/https, smtp, snmp, and so on to analyze their message contents and find malicious messages. This analysis is a very heavy task and usually needs some decryption engines and syntactic parsing engine. In our feasibility study, a simple URL filtering is implemented in software without specific hardware mechanism. This URL filtering program operated around 9.1 k IP packets/sec. on a single processor when all http packets are "*GET*" messages. It could be improved by introducing some strong string matching instructions in hardware. By the way, in case of embedded firewall, highly functional firewall could be realized by utilizing process information which are running on the local host OS. This means that even if the host might be infected by some viruses, the embedded firewall can prevent malicious network attacks checking the origin process of packets. Even if the host intends to be a malicious intruder using smart VPN software like SoftEther [10], it could be detected using the process information.

Table 1 Evaluation Results on a single processor.

		Classification	SPI	URL Filtering
Throughput	[IP packets/sec.]	3.5 M ∼ 5.0 M	329 K	9.1 K ∼ 1.5 M
Program size	[DDNP nodes]	16	443	1027

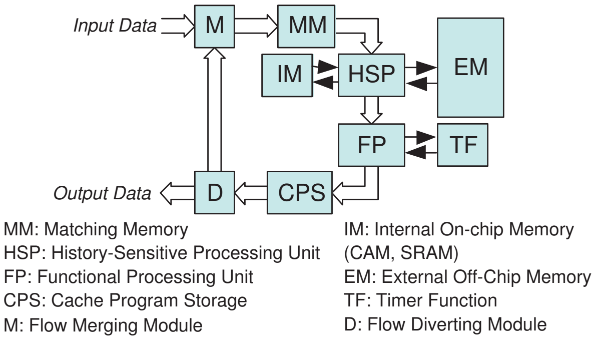

Input Data
M
MM
IM
HSP
EM
FP
TF
Output Data
D
CPS
MM: Matching Memory
HSP: History-Sensitive Processing Unit
FP: Functional Processing Unit
CPS: Cache Program Storage
M: Flow Merging Module
IM: Internal On-chip Memory (CAM, SRAM)
EM: External Off-Chip Memory
TF: Timer Function
D: Flow Diverting Module

Figure 3 Basic Architecture of the Embedded Data-Driven Firewall Processor.

2.2.2 Basic Hardware Architecture

Fig.3 shows the basic hardware architecture of a data-driven firewall processor. An input IP packet is first divided into several data-driven packets which hold a 32 bit data. Each data-driven packet is then fed from outside to the processor to a merging module (M). Passing through M, the packet arrives at a location in the matching memory (MM) and is stored there until its counterpart arrives. In MM, a pair of packets is identified by comparing their tag identifier and destination node number each other. If matching occurs, the two fragments are combined to form an operation packet containing an operation code, a destination node number, a color, and a pair of operands. Then, the operation packet will be delivered to the history-sensitive processing unit (HSP) and the functional processing unit (FP) where operation(s) indicated by the operation code can be performed. After the specified operation is executed, a result packet is generated and sent to a cache program storage (CPS). The next destination of the packet is read from the cache program store. The old destination node number is replaced with a new destination number. After a result packet is generated by CPS, it fed to a diverting module (D) and switched to an output according to the destination named in the header of the packet.

In the firewall processor, two lookup instructions for the high-speed classification are implemented in HSP. Furthermore, HSP allows the operation packet to access three kinds of memory modules; on-chip SRAM module, on-chip CAM module, and off-chip SDRAM module. To access those RAM memory modules flexibly, three modes of memory addressing are supported; i.e., absolute address, relative address by data of the packet, relative address by the tag identifier of the packet. The last one, which is a unique feature of the data-driven processor, helps the operation packets modify data belonging to their program instance (process). It is very useful in multiprocessing. The small CAM module is used for the efficient association as mentioned in section 3.1. As for off-chip SDRAM memory, it is used as an IP packet buffer to store the payload of the IP packet temporally. In order to utilize a burst access mode of SDRAM, a small cache memory is implemented at the interface circuits of off-chip memory module.

Since network processing often requires bit-wise operations like extraction of each field of IP header, FP supports some bit-wise instructions. Furthermore, it supports a timer function to realize several time-out operations.

With the customization of FP and HSP, a specific instruction set suitable for the embedded firewall is realized on DDNP. By utilizing the specific instruction set, the firewall programs on DDNP were developed for our feasibility study. In order to estimate total performance of the embedded data-driven firewall processor, layer 4 packet classification, SPI for TCP packets, and URL filtering were chosen and implemented. Preliminary evaluation results of them are summarlized in Table 1. This table shows that our firewall processor potentially operates at over 100 M b/s in wire-speed, even when using only a single processor. Furthermore, since scalable performance increasing along with the number of processors is also observed, a single chip equipped with 10 processors could be expected to operate over 1 G b/s packet stream.

2.3 Conclusion

This paper presents the design concept of our embedded data-driven firewall processor and describes its software structure and processor architecture. Preliminary evaluation results show that our firewall processor potentially operates at over 100 M b/s in wire-speed, even when using only a single processor. Furthermore, since scalable performance increasing along with the number of processors is also observed, a single chip integrated 10 processors could be expected to operate over 1 G b/s packet stream.

The proposed embedded data-driven firewall processor is one of applications for our data-driven network processor, DDNP. DDNP scheme has a lot of excellent advantages such as elegant parallel multiprocessing capability, natural power-saving capability, easy-to-design for SoC. Furthermore, the self-timed pipeline scheme introduced for the development of DDNP is the most promising scheme to realize highly-functional hardware modules for larger SoC's. For example, autonomous priority-based queuing chip utilizing a self-timed folded pipeline has been developed for Diffserv queuing [11]. This queuing hardware could be expected to be integrated into DDNP to realize a traffic engineering processor. However, this kind of large self-timed system needs efficient development tools such as high-speed simulator [12] or emulator [13]. Those powerful tools as well as DDNP applications have to be investigated in our further work.

3. Upper Limit of Step Gain for NLMS Algorithm in Noisy Environments

With advancements in LSI technology, adaptive filtering has recently been put to practical use, and applied to noise cancellers, automatic equalizers, echo cancellers and so forth. In such applications, a fast adaptive algorithm is necessary to actualize a real-time processing. For example, it is required thousands of adaptive filter's coefficients to realize an acoustic echo canceller. Therefore, complex algorithms are unsuitable for such systems. Although many algorithms have proposed and it has been improved the convergence speed and estimation accuracy, useful adaptive algorithms are only simple ones as the LMS algorithm or the normalized LMS algorithm. Since the normalized LMS (NLMS) algorithm requires few operations, it is widely used. However, this algorithm behaves unstably when a norm of an input vector becomes close to zero. The division in the procedure of the NLMS algorithm causes this property. One solution for this problem is addition of a positive constant to a square norm of an input vector. On the other hand, it is known that to interrupt adaptive filter's coefficients update when a norm of an input vector is smaller than a threshold is one of another ways to stabilize behavior of the NLMS algorithm. Though the stability is improved by using this method, the threshold must be decided to obtain a desired property. It has been shown the effects of interruption of adaptive filter's coefficients update and indicated a guarantee value (least upper bound of the convergence value) and the stochastic fastest convergence step gain (SF-NLMS algorithm). However, coefficients update interruption causes reduction of convergence speed. In this paper, the upper limit of the step gain that satisfies the guarantee value without interrupt coefficients update is shown.

Now, in this paper, $(L \times N)$ matrix A and $(N \times 1)$ vector b are described by A_{LN} and b_N

respectively.

3.1 Normalized LMS Algorithm

We define the notation for the sake of convenience and review the NLMS algorithm.

Let us define an input vector

$$\boldsymbol{x}_N(i) = [x(i), x(i-1), \cdots, x(i-N+1)]^T, \tag{1}$$

and the coefficient vector of the adaptive filter

$$\boldsymbol{h}_N(i) = [h(1), h(2), \cdots, h(N)]^T, \tag{2}$$

where N and ^{T} denote the number of the filter's coefficients and the transpose of a vector respectively. The output signal of the adaptive filter is expressed as

$$y(i) = \boldsymbol{h}_N^T(i)\boldsymbol{x}_N(i). \tag{3}$$

Assuming that $\boldsymbol{w}_N$ represents the coefficient vector of the unknown system, the desired signal is given by

$$d(i) = \boldsymbol{w}_N^T\boldsymbol{x}_N(i), \tag{4}$$

and the output error signal is defined by

$$e(i) = d(i) - y(i). \tag{5}$$

The NLMS algorithm is shown by the following:

$$\boldsymbol{h}_N(i+1) = \boldsymbol{h}_N(i) + \alpha\frac{\boldsymbol{x}_N(i)}{\|\boldsymbol{x}_N(i)\|^2}e(i), \tag{6}$$

where α is the step gain.

3.2 SF-NLMS Algorithm

In this section, we indicate the SF-NLMS algorithm.

Now, we define the observed output signal as

$$d'(i) = d(i) + v(i), \tag{7}$$

where $v(i)$ is observation noise.

3.2.1 Guarantee Value

The criterion for interrupting coefficients update to stabilize behavior of the NLMS algorithm is shown as

$$\|\boldsymbol{x}_N(i)\|^2 \leq \varepsilon, \tag{8}$$

where ε is the threshold. If coefficients update is interrupted according to the above criterion, the worst condition is to continue the state $\|\boldsymbol{x}_N(i)\|^2 \simeq \varepsilon$. Thus the converged norm of weight error vector ($\|\boldsymbol{\theta}'_N(i+1)\|^2 = \|\boldsymbol{h}'_N(i+1) - \boldsymbol{w}_N\|^2$) is smaller than the norm of weight error vector under the following condition:

$$E\left[\|\boldsymbol{x}_N(i)\|^2\right] = \varepsilon. \tag{9}$$

The converged norm of weight error vector is shown as

$$E\left[\|\boldsymbol{\theta}'_N(\infty)\|^2\right] \approx \frac{\alpha^2\sigma_v^2}{N\sigma_x^2}\frac{e^\mu}{e^\mu - 1}, \tag{10}$$

therefore, the guarantee value of the norm of weight error vector is given by

$$q = \frac{\alpha^2\sigma_v^2}{\varepsilon}\frac{e^\mu}{e^\mu - 1}, \tag{11}$$

where σ_v^2 is the variance of the observation noise and

$$\mu = \frac{2\alpha - \alpha^2}{N}. \tag{12}$$

3.2.2 Threshold of Interrupting Coefficients Update

From (11), the threshold to ensure the guarantee value is given by

$$\varepsilon = \frac{\alpha^2\sigma_v^2}{q}\frac{e^\mu}{e^\mu - 1}. \tag{13}$$

3.2.3 Stochastic Fastest Convergence Step Gain

The probability of executing coefficients update is expressed as

$$P = \int_\varepsilon^\infty \frac{1}{\sqrt{4\pi N\sigma_x^4}}\exp\left[-\frac{\left(\|\boldsymbol{x}_N(i)\|^2 - N\sigma_x^2\right)^2}{4N\sigma_x^4}\right] d\|\boldsymbol{x}_N(i)\|^2. \tag{14}$$

The time constant τ is expressed as

$$\tau = -\frac{1}{P\log_e(1-\mu)}. \tag{15}$$

The time constant is the number of samples to decay to $1/e$ (e is the base of natural logarithm). Now, we define the convergence speed as

$$c = \frac{1}{\tau}. \tag{16}$$

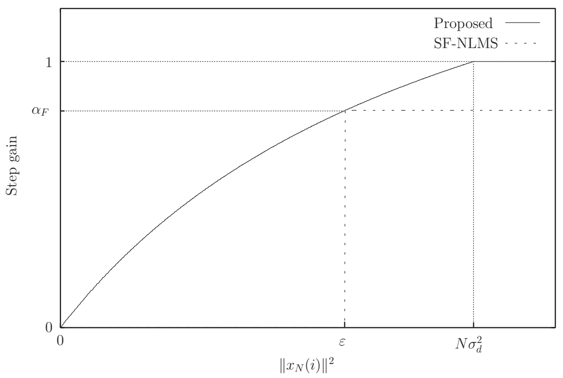


Figure 4 Comparison of step gain.

Since the convergence speed c is concave at $0 < \alpha \leq 1$, the stochastic fastest convergence step gain sets the differential $\frac{\partial c}{\partial \alpha}$ to zero. The differential $\frac{\partial c}{\partial \alpha}$ is given by

$$\frac{\partial c}{\partial \alpha} = -\frac{2\exp(\lambda)}{N(\alpha-2)} \left\{ \frac{\left[\alpha^2 - \left(3+\frac{\iota}{\sigma_x^2}\right)\alpha + 2\right]}{\left[\exp\left(-\frac{\iota\alpha}{\sigma_x^2(\alpha-2)}\right)+\exp(\lambda)\right]^2} \exp\left(-\frac{\iota\alpha}{\sigma_x^2(\alpha-2)}\right) + \frac{\exp(\lambda)(\alpha-1)(\alpha-2)}{\left[\exp\left(-\frac{\iota\alpha}{\sigma_x^2(\alpha-2)}\right)+\exp(\lambda)\right]^2} \right\}. \quad (17)$$

The stochastic fastest convergence step gain α_F is shown as

$$\alpha_F = \frac{2\left[\lambda - \log_e\left(\frac{\iota}{\sigma_x^2}\right)\right]}{\lambda\left(1+\frac{\kappa}{\sigma_x^2}\right) - \log_e\left(\frac{\iota}{\sigma_x^2}\right)}, \quad (18)$$

where

$$\iota = \frac{0.85\sqrt{2N}\sigma_v^2}{q}, \quad (19)$$

$$\kappa = \frac{\sigma_v^2}{q} \quad (20)$$

and

$$\lambda = 0.85\sqrt{2N}. \quad (21)$$

3.3 Upper Limit of Step Gain for Guarantee Value

This section modifies the threshold for the guarantee value, and shows the upper limit of the step gain for guarantee value without coefficients update interruption.

3.3.1 Upper Limit of Step Gain

By eq.(11), the guarantee value when coefficients are updated is given as

$$q = \frac{\alpha^2\sigma_v^2}{\varepsilon}\frac{e^\mu}{e^\mu - 1}. \quad (22)$$

If $N \gg 1$, then the guarantee value q is represented by

$$q = \frac{N\sigma_v^2}{\varepsilon}\frac{\alpha}{2-\alpha}. \quad (23)$$

To give assurance the guarantee value without coefficients update interruption, $\|\boldsymbol{x}_N(i)\|^2 \geq \varepsilon$ should be satisfied constantly. So, the threshold ε is set

$$\varepsilon = \|\boldsymbol{x}_N(i)\|^2. \quad (24)$$

From eq.(23), the guarantee value q is shown as

$$q = \frac{N\sigma_v^2}{\|\boldsymbol{x}_N(i)\|^2}\frac{\alpha}{2-\alpha}, \quad (25)$$

and the step gain α to satisfy this guarantee value is given by

$$\alpha = \frac{2q\|\boldsymbol{x}_N(i)\|^2}{q\|\boldsymbol{x}_N(i)\|^2 + N\sigma_v^2}. \quad (26)$$

In general the step gain $\alpha \leq 1$,

$$\frac{2q\|\boldsymbol{x}_N(i)\|^2}{q\|\boldsymbol{x}_N(i)\|^2 + N\sigma_v^2} < 1, \quad (27)$$

or

$$\|\boldsymbol{x}_N(i)\|^2 < \frac{N\sigma_v^2}{q}, \quad (28)$$

if $\|\boldsymbol{x}_N(i)\|^2 \geq \frac{N\sigma_v^2}{q}$ then the step gain $\alpha = 1$.

Consequently, coefficients update method to satisfy the guarantee value is shown as

$$\boldsymbol{h}_N(i+1) = \boldsymbol{h}_N(i) + \gamma(i)\frac{\boldsymbol{x}_N(i)}{\|\boldsymbol{x}_N(i)\|^2}e(i), \quad (29)$$

where $\gamma(i)$ is the upper limit of the step gain:

$$\gamma(i) = \begin{cases} \dfrac{2q\|\boldsymbol{x}_N(i)\|^2}{q\|\boldsymbol{x}_N(i)\|^2+N\sigma_v^2}, \text{ if } \|\boldsymbol{x}_N(i)\|^2 < \dfrac{N\sigma_v^2}{q} \\ 1, \text{ otherwise} \end{cases} \tag{30}$$

If the guarantee value q is assumed

$$q = \frac{\sigma_v^2}{\sigma_d^2}, \tag{31}$$

the step gain $\gamma(i)$ when $\|\boldsymbol{x}_N(i)\|^2 < \dfrac{N\sigma_v^2}{q}$ is given by

$$\begin{aligned} \gamma(i) &= \frac{2q\|\boldsymbol{x}_N(i)\|^2}{q\|\boldsymbol{x}_N(i)\|^2 + N\sigma_v^2} \\ &= \frac{2\|\boldsymbol{x}_N(i)\|^2}{\|\boldsymbol{x}_N(i)\|^2 + N\sigma_d^2}, \end{aligned} \tag{32}$$

where σ_d^2 is the variance of the output signal $d(i)$. In such case,

$$\|\boldsymbol{x}_N(i)\|^2 < \frac{N\sigma_v^2}{q} = N\sigma_d^2, \tag{33}$$

therefore, the step gain $\gamma(i)$ when $q = \sigma_v^2/\sigma_d^2$ is given as

$$\gamma(i) = \begin{cases} \dfrac{2\|\boldsymbol{x}_N(i)\|^2}{\|\boldsymbol{x}_N(i)\|^2 + N\sigma_d^2}, \text{ if } \|\boldsymbol{x}_N(i)\|^2 < N\sigma_d^2 \\ 1, \text{ otherwise} \end{cases} \tag{34}$$

Figure 4 shows a comparison of the step gain of the proposed method (eq.(34)) and the SF-NLMS algorithm[16].

3.4 Computer Simulation

In this section, the result of computer simulation is shown.

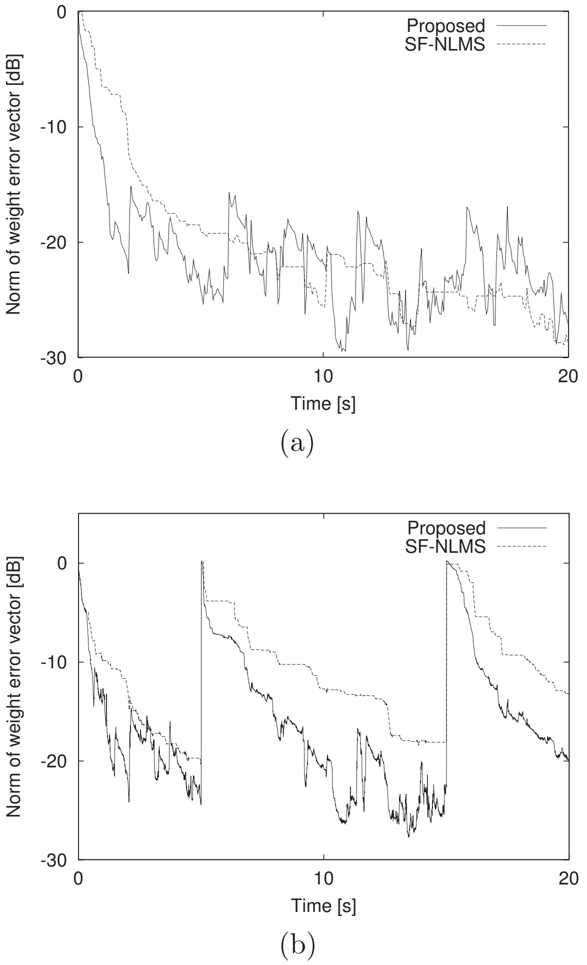
The input signals and the observation noise are speech signals that are voiced speech segments sampled at 8 kHz. S/N is set as 20 dB. The impulse response length of adaptive filter is 100 ($N = 100$).

The stochastic fastest convergence step gain of the SF-NLMS algorithm is set to 0.816. The variances of input signals and desired signals are shown, respectively, as

$$\sigma_x^2 = (1-b)\sum_{k=0}^{\infty} b^k x^2(i-k), \tag{35}$$

$$\sigma_d^2 = (1-b)\sum_{k=0}^{\infty} b^k d^2(i-k), \tag{36}$$

where b is set to 0.9999.



(a)

(b)

Figure 5 Comparison of convergence properties.

Figure 5 shows the convergence properties. The proposed algorithm surpasses the SF-NLMS algorithm in tracking ability.

3.5 Conclusion

This paper shows the upper limit of the step gain that satisfies the guarantee value without coefficients update interruption. The proposed method improves convergence speed on the SF-NLMS algorithm. It is clear that the proposed method is very useful for applying in acoustical systems, case of unsteady observation noise is inflicted in particular.

Acknowledgements

This research project is supported by the National Institute of Information and Communications Technology (NICT).

Reference

[1] R. Zalenski, "Firewall technologies," *IEEE Potentials*, Vol. 21, No. 1, pp. 24–29, Feb. 2002.

[2] J. McHugh, A. Christie, J. Allen, "Defending Yourself: The Role of Intrusion Detection Systems," *IEEE Software*, Vol.17, No.5, pp. 42–51, Sep. 2000.

[3] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith, "Implementing a Distributed Firewall," *ACM Conf. on Computer and Communications Security*, pp. 190–199, Nov. 2000.

[4] H. Terada, S. Miyata, and M. Iwata, "DDMP's: Self-Timed Super-Pipelined Data-Driven Processors," *Proceedings of the IEEE*, Vol. 87, No. 2, pp. 282–296, Feb. 1999.

[5] C. Payne and T. Markham, "Architecture and Applications for a Distributed Embedded Firewall," *17th Computer Security Applications Conf. (ACSAC)*, pp. 329–336, Dec. 2001.

[6] D. Morikawa, M. Iwata, H. Hayashi, and H. Terada, "Superpipelined IP-Address Lookups on a Data-Driven Network Processor," *Int. Conf. on Parallel and Distributed Computing and Systems*, pp.431–436, Aug. 2001.

[7] D. Morikawa, M. Iwata, and H. Terada, "Super-Pipelined Implementation of IP Packet Classification," *Journal of Intelligent Automation and Soft Computing*, Vol. 10, No. 2, pp. 175–184, Aug. 2004.

[8] R. Zhang, M. Iwata, Y. Shirane, T. Asahiyama, W. Su, and Y. Zheng, "High Speed Stateful Packet Inspection in Embedded Data-Driven Firewall," *NEINE'04*, Sep. 2004 (to be presented).

[9] D. Morikawa, T. Matsumoto, and M. Iwata, "Fast Packet Filtering in Data-Driven Embedded Firewall," *NEINE'04*, Sep. 2004 (to be presented).

[10] SoftEther Corp., "SoftEther VPN System," *www.softether.com*, ver. 1.01, Sep. 2004.

[11] M. Iwata, M. Ogura, Y. Ohishi, H. Hayashi, and H. Terada, "100MPacket/s Fully Self-Timed Priority Queue: FQ," *Int. Solid-State Circuits Conf. (ISSCC) 2004*, Session 8, No.1, San Francisco, CA. U.S.A., Feb. 2004.

[12] S. Sannomiya, Y. Ohmori, and M. Iwata, "A Macroscopic Behavior Model for Self-Timed Pipeline Systems," *17th Workshop on Parallel and Distributed Simulation (PADS 2003)*, pp. 133–140, SanDiego, CA, U.S.A., June 2003.

[13] S. Ogasawara, S. Sannomiya, Y. Omori, M. Iwata, "An On-Chip Trace-Driven Emulation for Self-Timed Data-Driven Processors," *NEINE'04*, Sep. 2004 (to be presented).

[14] J. Nagumo and A. Noda, "A learning identification method for system identification," *IEEE Trans. Automat. Contr.*, vol.63, pp.1692–1716, Dec. 1975.

[15] M. Fukumoto, H. Kubota and S. Tsujii, "Improvement in stability and convergence speed on normalized LMS algorithm," *roc. IEEE ISCAS '95*, vol.2, pp.1243–1246, Seattle, WA, May 1995.

[16] M. Fukumoto, H. Kubota and S. Tsujii, "Simplification of stochastic fastest NLMS algorithm," *Proc. IEEE ISCAS '99*, 38.7, Orlando, FL, June 1999.