

# Compound Data Oriented Processing in USAGI IPv6 Stack

Hideaki YOSHIFUJI<sup>†a)</sup>, *Student Member*, Kazunori MIYAZAWA<sup>††b)</sup>,  
Masahide NAKAMURA<sup>†††c)</sup>, Yuji SEKIYA<sup>†d)</sup>, *Nonmembers*, Hiroshi ESAKI<sup>†e)</sup>,  
and Jun MURAI<sup>††††f)</sup>, *Members*

**SUMMARY** IPv6 has been attracting much attention as a drastic countermeasure for severe shortage of addresses on the Internet. Linux, one of operating systems, also supports IPv6. However, the quality of the protocol stack was not so good. In this study, we worked out an architectural design for this basic software in Linux. Specifically, we introduce compound data oriented processing in network stack, which simplifies data object management and relationship between object. We show the quantitative research shows that the quality of protocol stack of IPv6 has been greatly improved.

**key words:** *IPv6, Linux, Compound Data Oriented Processing*

## 1. Introduction

The Internet Protocol Version 4 (IPv4)[10] has been the fundamental protocol in the Internet for long time. However, several grave issues have arisen for ten years since it was developed. For instance, severe shortage of address space was expected. Internet Protocol Version 6 (IPv6)[1], so called as “Next Generation Internet Protocol (IPng)”, was designed to solve the problems of the traditional Internet Protocol, Version 4.

The specification of IPv6 has almost fixed at the IETF[13]. After the IPv6 technology being put on its experimental stage, a lot of network appliance and software vendors are now supporting it, and commercial service by Internet Service Providers is also available. Now, IPv6 is reaching practical stage with product quality.

Linux IPv6 protocol stack is not really new; it has been available since early Linux 2.1.x days in 1996. However, once it was integrated into the mainline ker-

nel, it was not so actively developed while situation and specification were changed.

In 2000, USAGI Project[15] was established by WIDE Project[16] and several hackers, who had lead Linux IPv6 Users JP joined the Project. The Project studies and develops on subject from kernel to user space to provide high quality and up-to-date implementation to the world.

According to the research by the Project, we found issues in Linux IPv6 stack. the architecture is too complex to keep them all right.

In this paper, we describes the new architectures to make Linux conform well to specifications by IETF. Specifically, we introduce compound data oriented processing scheme. This scheme is with simple, self-managed data object and compound of them. We describe how our high quality protocol stack is realized in this scheme.

## 2. Neighbor Discovery

Neighbor Discovery (ND) is one of core elements of IPv6 and known that it consists of Router and Prefix Discovery, Address Resolution and Neighbor Unreachability Detection and Redirect [9], [14]. They are essential to keep stable communication, and requires accurate timer and state management by various events on network. For example, Address Resolution and Neighbor Unreachability Detection use Neighbor Cache Entry (NCE) in its concept and it is required to keep the timer accurate to manage reachability of neighbors properly.

In the previous implementation, NCE was managed by periodic polling timer and timer inside the entry. The reachability was managed in the following two ways (Figure 1).

### Periodic timer

Periodic timer (30 seconds) invokes management task, which checks reachability of every node. It also clean-up entries in the table.

### Entry-internal timer

Dynamic timer inside the entry invokes management task for itself in semi-reachable states. The timeout depends on the state.

---

Manuscript received January 1, 2002.

Manuscript revised January 1, 2003.

Final manuscript received January 1, 2004.

<sup>†</sup>The authors are with the University of Tokyo, Tokyo, Japan.

<sup>††</sup>The author is with Yokogawa Electric Corporation, Tokyo, Japan.

<sup>†††</sup>The author is with Hitachi Communication Technologies, Ltd., Tokyo, Japan.

<sup>††††</sup>The author is with Keio University, Kanagawa, Japan.

a) E-mail: hideaki@yoshifuji.org

b) E-mail: kazunori@miyazawa.org

c) E-mail: nakam@linux-ipv6.org

d) E-mail: sekiya@wide.ad.jp

e) E-mail: hiroshi@wide.ad.jp

f) E-mail: jun@wide.ad.jp

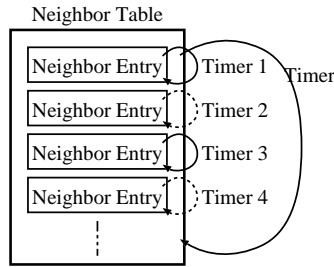


Fig. 1 Linux NDP Table with complex timer

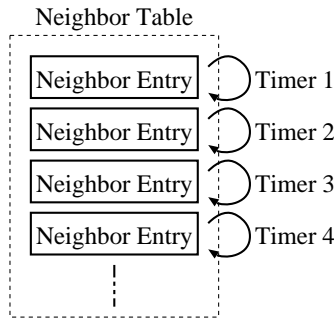


Fig. 2 USAGI NDP Table with Dynamic Timers

Table 1 TAHI Conformance Test Result of NCE Management (PASS Ratio)

Linux 2.4.18	USAGI 2.4	KAME/FreeBSD 4
39%	89%	98%

. Protocol requires accuracy of few seconds for all states, the former results in inaccurate state management. Furthermore, state management was split and resource management was complex.

Therefore, we redesigned NCE management. We split the management task including timers and mutual exclusion (locks), for state management (Figure 2). Each entry has its own dynamic timer and lock for state transition and it is responsible for its own state transition; in short, each NCE is self-managed. On the other hand, periodic timer is not used for state management, and it is only used for garbage collection.

In this architecture, the tasks under the global lock are simplified. Resource managements for neighbors including mutual exclusion are also greatly simplified. As the results of NCE management in Table 1 shows, it becomes possible to exchange messages correspondent to the status of each NDP entry as defined in the NDP specifications.

### 3. Routing Table

Linux IPv6 routing table, as known as “Forwarding Information Base,” is constructed by Radix Tree[12]. A Radix Tree is built with internal nodes and leaves, which are nodes marked with `RTN_RTINFO`. Every node represents a bit position to test. And each leaf has a

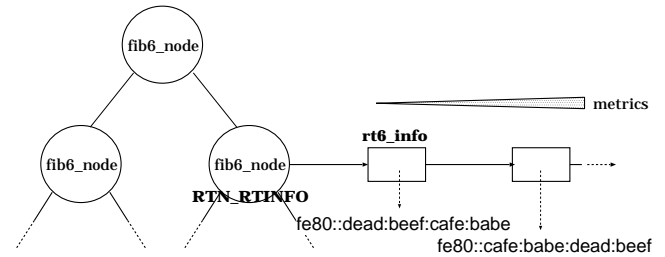


Fig. 3 Linux Routing Table

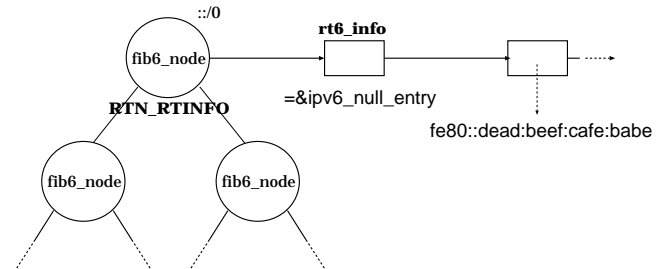


Fig. 4 Linux IPv6 Routing Table Structure

linked list which stores actual routing information, such as nexthop (Figure 3), which is actually represented as NCE. The elements in the linked list are chained in numerical order of metric. Linux IPv6 routing table supports equal-cost multiple paths.

#### 3.1 Default Routers Management

Neighbor Discovery for IP Version 6 (IPv6) [9] introduces conceptual data structure as known as “Default Router List,” which holds information of “default routers” advertised via Router Advertisement messages.

Linux does not have separate “Default Router List” structure. It holds the default routers as a kind of routes to `:::0`, instead of holding the information in separate list.

However, as shown in Figure 4, Linux IPv6 protocol stack has a radix tree with fixed node information on the top and it points to `ipv6_null_entry`. Therefore, when default route is added, the information is attached at the next of the `rt6_info{}` structure which contains `ipv6_null_entry`. This causes default route not to be referred.

In USAGI implementation, we replace the `ipv6_null_entry` with the new entry when adding a new routing entry on the top level root of the tree (Figure 5). When the last route is being deleted from the the top level root of the tree, we re-insert `ipv6_null_entry`. This means, we actually treat the “default routes” which hold information regarding “Default Routers” as normal routes. And now, we can naturally insert and remove the “default route” entries properly to/from the routing table as we can do for

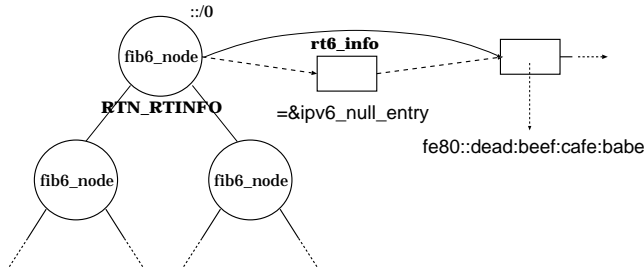


Fig. 5 USAGI IPv6 Routing Table Structure

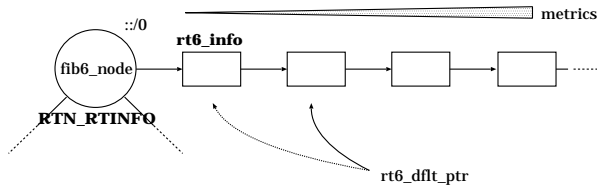


Fig. 6 Default Routers in Linux

other normal routes.

### 3.2 Router Precedence Management

As mentioned before, the default routers are stored on the top level root node of the routing tree (Figure 6). It is called “Default Router Selection” to pick one default router from the default router list. In Default Router Selection, it does round-robin the default router list when it becomes unreachable. To achieve this in Linux, the default router was pointed by `rt6_dflt_pointer`, which is guarded by the global lock named `rt6_dflt_lock`, and updated when router is unreachable (Figure 6).

In this implementation, there were several issues.

#### Unfairness

`rt6_dflt_pointer` is reset when routing is modified; this happens very often and routers are not equally selected.

#### Lack of Generic route selection

`rt6_dflt_pointer` was only for the default routes; The logic could not apply to generic route selections; we need to have generic way to select one from multiple routes to support improved router selection strategy such as “Default Router Preferences, More-Specific Routes, and Load Sharing” [2].

#### Metrics

Linux treated metrics in other way than normal routes. Now we treat default routes like normal routes, it is the time to eliminate special handling of metrics of default routes.

To solve these issues, we introduced a new generic round-robin code for the routes with same metrics. We do round-robin for routes with same metrics when a

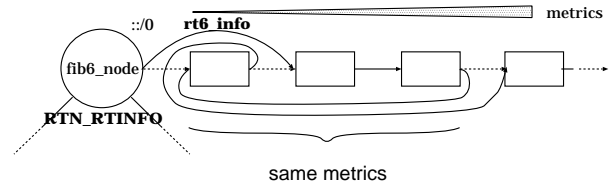


Fig. 7 New Method for Route Round-robin

route in that set is used (Figure 7). When a route is looked up, use the first entry which is in “probably reachable” state in conjunction with NCE in the next-hop entry and relink it at the tail of the entries with same metrics. In this way, the `rt6_dflt_pointer` and `rt6_dflt_lock` are eliminated. This logic can apply to not only the default routes but also other generic routes.

The “Router Selection” draft [2] also introduces the concept of “preference” of routes. The routers advertise 2-bit preferences of routes with Router Advertisement message, and hosts select more preferred routers.

To implement this, we stores preference (2 bit) of routes into the flags of the routing informations instead of reflecting it to the metrics. This is to separate this attribute from metrics to simplify other logic, for example to avoid fixing up routing table when receiving RA.

Finally, when a route is looked up, use the first entry with highest preference is in “probably reachable” state in conjunction with NCE in the next-hop entry and relink it at the tail of the entries with same metrics.

This is the way how we improved routing operation including default routes and router selections using compound data structure oriented scheme in conjunction with NCE; objects are self-managed and data compound represents data relationship.

## 4. IP Packet Transformer

The role of IP is to transfer datagrams from source to destination. In traditional manner, it is enough to carry it in as-is way. Today, however, keeping security becomes important.

In order to support these requirements with minimum impact against users, IP layers are required to be changed. Now, they not only transfer datagrams from one to another, but also flexibly “transforms” its format. For example, IP layer will encrypt / decrypt packets to keep security. It also appends authentication (or integrity check) data etc. These are all kinds of “transformations.”

### 4.1 Stackable Destination and XFRM

A new framework for IP packet processing has been introduced into linux-2.5.x to implement IPsec, e.g.

Authentication Header[6] and Encapsulating Security Payload[7]. It is called XFRM and “Stackable Destination.”

XFRM stands for transformer. Its fundamental data structures are `xfrm_policy{}` and `xfrm_state{}`. Each represents IPsec Policy (SP) and Security Association (SA) respectively. Thus, Security Policy Database (SPD) consists of `xfrm_policy{}`. Similarly, Security Association Database (SAD) consists of `xfrm_state{}`. An `xfrm_state{}` is associated with `xfrm_policy{}` via `xfrm_tmpl{}`, which represents template for packet transformation.

Stackable destination is the infrastructure for packet transformation in the output path. It looks like a kind of linked list of `dst{}`, the protocol independent destination cache. This list is created temporarily and cached according to the policy. `dst{}` has its own output method `output` and it transforms the packet in conjunction with `xfrm_state{}`, which represents state of transformer.

The netlink[11] infrastructure is used as fundamental user interface to maintain SAD and SPD. In addition to this native interface, the standardized `PF_KEY`[8] interface, which is for SAD, is supported, and for SPD, `PF_KEY KAME` [5] extension is supported.

## 4.2 AF Independent XFRM Infrastructure

Since core functionality of the XFRM engine is common among address families (e.g. IPv6 or IPv4). AF independent XFRM infrastructure has been introduced.

Instance of AF specific XFRM engine is instantiated by registering AF specific information tables, e.g. `xfrm_policy_afinfo{}` and `xfrm_state_afinfo{}`. to the core XFRM engine. e.g. `xfrm_policy_afinfo{}` and `xfrm_state_afinfo{}`. Common variables are also passed via the tables.

## 4.3 Packet Processing Details

In this section, we describe details of packet processing.

### 4.3.1 Output Path

The output process of IPsec fully uses this architecture. The sequence of calling functions are `xfrm_lookup()`, `xfrm_tmpl_resolve()`, `xfrm_bundle_create()` and `dst_output()`.

First, `xfrm_lookup()` looks up `xfrm_policy{}` in SPD after routing resolution. At the moment the parameter `dst{}` in the stack points `original_dst{}` structure. `xfrm_tmpl_resolve()` is called in `xfrm_lookup()` to resolve `xfrm_tmpl{}` in `xfrm_policy{}` which represents how the packet is processed and finds set of `xfrm_state{}` for it. This process corresponds to looking up IPsec SA or IPsec SA bundle matched with IPsec policy.

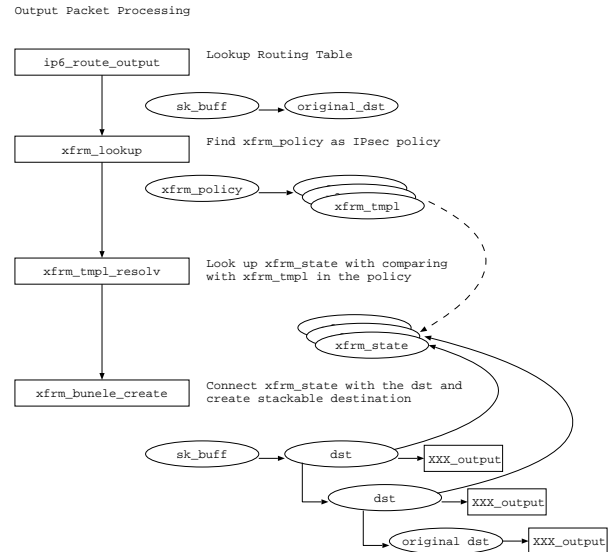


Fig. 8 IPsec output process

Then, `xfrm_bundle_create()` creates the stackable destination. This corresponds to creating IPsec SA (or SA bundle if multiple SA are needed).

Finally, `dst_output()` is called after building up the packet. Each output routine specified by the function pointer in the `dst{}` is called along with the chain of `dst{}` by popping up the `dst{}`. This pointer points `esp6_output()` etc. The output function is able to use `xfrm_state{}` from `dst{}` pointer in `sk_buff{}`. And at last, the original `dst{}`'s output function is called.

## 4.4 Input Path

The input process for IPsec is more simple than output.

As all extension header handlers and protocol handlers are registered with `inet6_protos[]` at its initialization phase, processing routines for AH and ESP are also registered to `inet6_protos[]` at initiation.

When a packet is reached input handler, The kernel parse it from the head, and call the handler according to the registration table.

Each handler of IPsec protocol looks up `xfrm_state{}` and processes it. If it succeeds, used `xfrm_state{}` pointer is kept in `sec_path{}` in `sk_buff{}` which contains the packet.

Finally, `xfrm_policy_check()` is called at the entrance of upper layer process. it compares `xfrm_tmpl{}` in `xfrm_policy{}` and `xfrm_state{}` kept in `sec_path{}` to determine if it is allowed to be delivered.

## 4.5 Test Results

On 24th April, 2003, Tom Lendacky reported to netdev mailing list that Test results of Linux-2.5 IPsec are very excellent(Table 2).

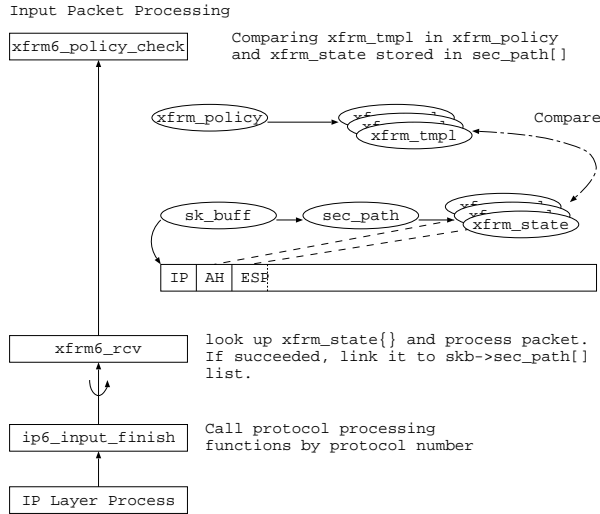


Fig. 9 IPsec input process

We have tried to fix the bugs in IPv6 IPsec fragmentation, and they should be fixed for now.

**Table 2** Summary of TAHI Conformance Test (linux-2.5.58, %)

Test Series	Pass	Warn	Fail
ipsec	95	2	3
ipsec4	98	2	0
ipsec4-udp	96	4	0

#### 4.6 Future Evolution

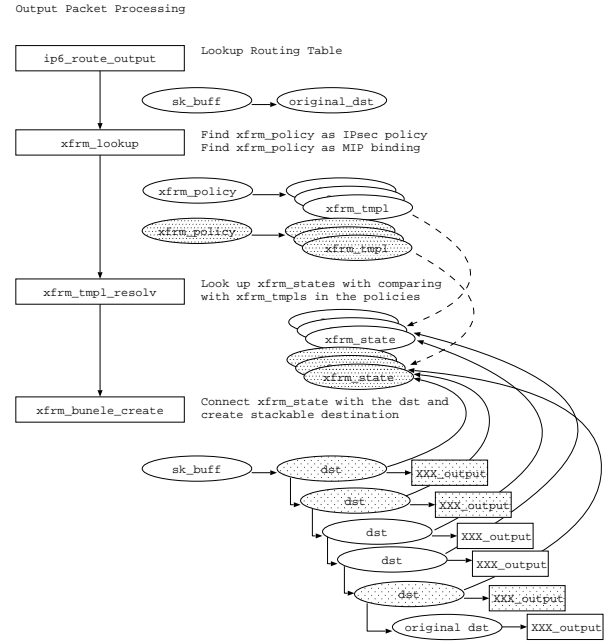
The XFRM infrastructure and the Stackable Destination are the promising schemes. For example, Mobile IPv6[4] is feasible.

Figure 10 shows the prototype diagram for Mobility support using XFRM / Stackable Destination infrastructure.

In this prototype, we introduce another set of policy for Mobile IP. This policy controlled by mobility daemon in user space based on Binding Update. It describes the type of transformation (e.g. appending Home Address Option in the destination option header etc.) and its corresponding binding data (i.e. Care-of Address and Home Address).

The `xfrm_bundle_create()` compiles multiple templates into the single stackable destination, taking account of the flow of the process. Assuming co-existence of AH, ESP and Mobile IP, AH requires full packet. However, source address in the packet should be home address. Thus, the Stackable Destination should be constructed as follows.

- Mobile-IP Dest1, which inserts Care-of Address into the packet.
- Mobile-IP Rthdr, which inserts routing header (type 2) into the packet.



**Fig. 10** Mobile IP using XFRM / Stackable Destination Scheme

- ESP, which encrypts the packet after the destination header for Home-Address option.
- AH, which generates authentication data for packet and insert it after the destination header.
- Mobile-IP Dest2, which swaps Care-of Address and Home Address in the packet.

In this way, complex packet will be constructed in these schemes.

## 5. Conclusion

In this paper, we describes the architecture of USAGI IPv6 networking stack. While USAGI software was developed based upon original Linux, we introduced new compound data oriented architecture to various area. This idea is to make some object simple and self-managed, and the compound of the object represents complexed processing. NCE in Neighbor Discovery manages itself. In routing area, Routing Entry is corresponded with the NCE, and the reachability is managed by NCE itself. Preference of router is organized itself within the compound of the routing objects. On the other hand, each IP packet transformer is represented by a object, and the chain of object called “stackable destination” enables us flexible transformation. This idea totally enables us to achieve natural and high quality protocol stack.

Our product is widely and freely available under open source license, and which can be used with various kinds of Linux systems.

Finally, we list future items.

- Policy routing, which realizes routing not only by

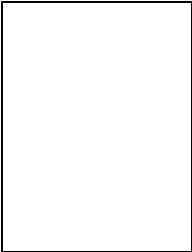
destination address but also by the source address, traffic class, etc. This also affects the source address selection [3] algorithm.

- New security architecture for real end-to-end communication.
- More contribution to mainline kernel based on wide knowledge, as one of co-maintainer of Linux kernel networking stuff.

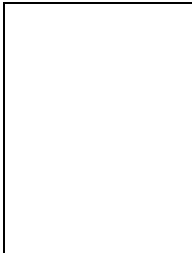
We will continue studying on architecture from basic to application. We continue providing high quality implementation widely to deploy IPv6, and contributing IETF standardization process.

#### References

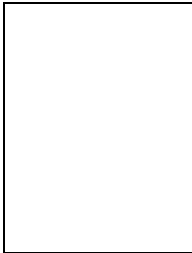
- [1] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC2460, Devenber 1998.
- [2] R. Drave and R. Hinden. Default router preferences, more-specific routes, and load sharing. Work in Progress, June 2002.
- [3] R. Draves. Default Address Selection for Internet Protocol version 6 (IPv6). RFC3484, February 2003.
- [4] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. Work in Progress, June 2003.
- [5] KAME Project. KAME Project Web Page. <http://www.kame.net>.
- [6] S. Kent and R. Atkinson. IP Authentication Header. RFC2402, November 1998.
- [7] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). RFC2406, November 1998.
- [8] D. McDonald, C. Metz, and B. Phan. PF\_KEY Key Management API, Version 2. RFC2367, July 1998.
- [9] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC2461, December 1998.
- [10] J. Postel. Internet Protocol. STD0005, September 1981.
- [11] J. H. Salim, H. Khosravim, A. Kleen, and A. Kuznetsov. Netlink as an IP Services Protocol, December 2002.
- [12] Keith Sklower. A tree-based packet routing table for berkeley unix. In *USENIX Winter*, pages 93–104, 1991.
- [13] The Internet Society. Internet Engineering Task Force. <http://www.ietf.org>.
- [14] S. Thomson and T. Narten. IPv6 Stateless Address Auto-configuration. RFC2462, December 1998.
- [15] USAGI Project. USAGI Project Web Page. <http://www.linux-ipv6.org>.
- [16] WIDE Project. WIDE Project Web Page. <http://www.wide.ad.jp>.



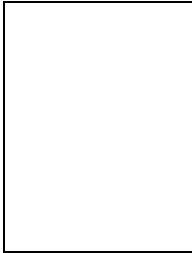
**Hideaki YOSHIFUJI** was born in Tokyo, Japan. He received the B.Eng., and M. of Information Sciences from Tohoku University, Sendai, Japan, in 1999 and 2001, respectively. He works for USAGI Project as core member since its establishment in 2000. Now he is aiming for Ph.D. in the University of Tokyo. He is one of Linux co-maintainers of networking area, since 2003.



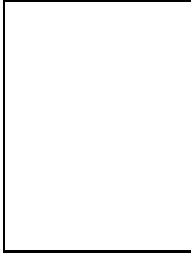
**Kazunori MIYAZAWA** He joined Yokogawa Electric Corporation. He joined USAGI Project since 2000.



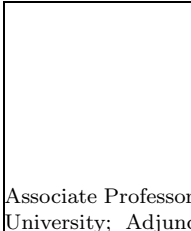
**Masahide NAKAMURA** He joined Hitachi Corporation. He joined USAGI Project since 2002.



**Yuji SEKIYA** received B. from Kyoto University, received M. from Keio University. The University of Tokyo. He works for USAGI Project as core member since its establishment.



**Hiroshi ESAKI** He received the B.E. and M.E. degrees from Kyushu University, Fukuoka, Japan, in 1985 and 1987, respectively. And, he received Ph.D from University of Tokyo, Japan, in 1998. In 1987, he joined Research and Development Center, Toshiba Corporation, where he engaged in the research of ATM systems. From 1998, he works for University of Tokyo as an associate professor, and works for WIDE project as a board member. He has been at Bellcore in New Jersey (USA) as a residential researcher from 1990 to 1991, and has engaged in the research on high-speed computer communications. From 1994 to 1996, he has been at CTR (Center for Telecommunications Research) of Columbia University in New York (USA) as a visiting scholar. He is currently interested in a high-speed Internet architecture, including MPLS technology, a mobile computing, and IPv6.



**Jun MURAI** Born in March 1955 in Tokyo. Graduated Keio University in 1979, Department of Mathematics, Faculty of Science and Technology, MS for Computer Science from Keio University in 1981, Received Ph.D in Computer Science from Keio University in 1987. Associate Professor, Faculty of Environmental Information, Keio University; Adjunct Professor at Institute of Advanced Stud-

ies, United Nations University; Instructor at Tokyo University of Art and Music; President of Japan Network Information Centre (JPNIC). General chair person of WIDE Project, a Internet research consortium in Japan. Vice chair person of Internet Society, Japanese chapter; Vice president of Internet Association, Japan.