Developing CPE Routers based on NetBSD: Fifteen Years of SEIL

Masanobu SAITOH(msaitoh@netbsd.org)[*] Hiroki SUENAGA(hsuenaga@iij.ad.jp)[†]

March 2014

Abstract

Typical ISPs use customized small routers to connect their network from customer's local networks. Such routers are called CPE, Customer Premises Equipment. We, Internet Initiative Japan (IIJ), also have own CPE named 'SEIL.' SEIL is a German word of rope. SEIL ropes IIJ and its customers together.

The firmware of SEIL is a customized NetBSD. IIJ has self-manufactured the firmware for 15 years, since March 1999. We describe about some implementation and enhancement for NetBSD during SEIL's 15 years history.

1 The Target environment of our CPE

Customer Premises Equipments (CPE) are communication devices, such as Internet access gateways and routers, deployed in customer's homes and offices. There are various customers, so we need to describe the target customers and environments before entering detailed discussion.

IIJ is an ISP company and most of its customers are corporations. Typical corporations use the Internet to communicate with their partners, satellite offices, and shops such as convenience stores.

Internal communications of corporations definitely include a lot of confidential information. So our CPE must have cryptographic functionalities such as IPsec and SSL, and their accelerators as much as possible. Supporting various secure tunneling protocols are also important. Our CPE supports PPTP, L2TP, L2TPv3, IPsec, SSTP, and so on, to satisfy many different requirements of our customers.

Most corporations don't have enough IP addresses and use NAPT to connect to the Internet. They also use IP filters to ensure minimum security. Since there are a lot of persons and computers in a office, performances of NAPT and IP filters are most important requirements of our CPE.

Such complicated requirements make CPE's configurations so difficult. IIJ have put efforts to simplify configuration syntax, but there are limitations to do so. Engineers of IIJ can write configurations, but most engineers in various corporations can't do enough. Thus, IIJ has found one more important requirement, simple and easy management of a number of CPEs. The word 'management' includes some concepts, configurations, operations, and monitoring. The name SEIL was selected to show this principle of management, it is abbreviation of 'Simple and Easy Internet Life.'

Table 1 shows past CPEs which made to achieve the mentioned requirements. Each of hardware architecture has been changed by the era of network environment, but core concepts of the CPEs are not changed. In this paper, we focus on the concepts and software implementations to realize it. We discuss about our management framework at first. It is an important start point of us, and an important difference between ISP's genuine CPE and other CPEs. In technical point of view, the management framework is not a BSD specific topic. But it is important

[*]The NetBSD Foundation
[†]Internet Initiative Japan Inc.

Table 1: Hardware architecture of SEILs

WAN Interfaces	LAN Interfaces	CPU(Model)	Released
128Kbps BRI	10Mbps Ethernet	Hitachi SH2(SH7604)@20MHz	Aug 1998
1.5Mbps PRI	10Mbps Ethernet	Hitachi SH3(SH7709A)@133MHz	Dec 1999
128Kbps BRI	100Mbps Ethernet	Hitachi SH4(SH7750)@200MHz	Oct 2001
1.5Mbps PRI	100Mbps Ethernet	Hitachi SH4(SH7750)@200MHz	Oct 2001
100Mbps Ethernet	100Mbps Ethernet	Hitachi SH4(SH7750)@200MHz	Nov 2001
25Mbps ATM	100Mbps Ethernet	Hitachi SH4(SH7750)@200MHz	Oct 2002
1Gbps Ethernet	1Gbps Ethernet	Freescale PowerPC G4(MPC7445)@600MHz	Jun 2003
100Mbps Ethernet	100Mbps Ethernet	Intel XScale(IXP425)@400MHz	Dec 2003
1Gbps Ethernet USB 3G/LTE Modem	1Gbps Ethernet	Cavium Octeon(CN3010)@300MHz	Feb 2008
1Gbps Ethernet USB 3G/LTE Modem	1Gbps Ethernet	Cavium Octeon(CN3120)@500Mhz	Feb 2008
100Mbps Ethernet USB 3G/LTE Modem 128Kbps BRI	100Mbps Ethernet	Intel XScale(IXP432)@400MHz	Oct 2008
1Gbps Ethernet USB 3G/LTE Modem	1Gbps Ethernet 802.11n Wireless LAN	Marvell Kirkwood(88F6281)@1.2GHz	Feb 2013

to understand why ISPs have made own CPEs from scratch. We discuss about extensions and modifications for NetBSD at second. And we discuss some knowledge and tweaks to make our daily development work flows easy and efficient.

2 Management of SEIL

2.1 Central management

Most important motivation of self-manufacturing CPE is to manage CPEs from ISP politely. The quality of managing CPEs is one of the quality of Internet connections. Most of CPEs are designed to be managed by local network managers of a customer. Of course, we can make a template configuration for them, we can advice what to do, and so on. But IIJ thinks that the work of customer should be almost nothing. Only work is to check reports from ISP and confirm there is no problem.

We create a management framework to achieve it. The framework is named SMF, SEIL Management Framework(Figure 1). It was released in 2003. The framework has the following behaviors:

1. Zero Configuration. Just power the CPE on, that's all.

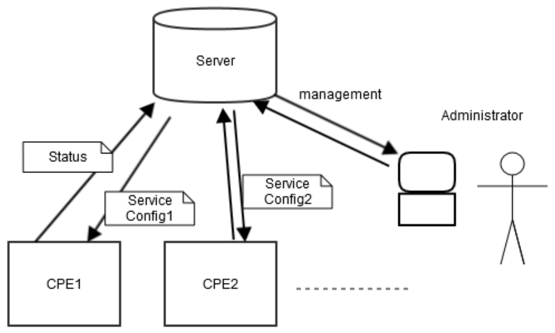

Figure 1: The SMF system

2. Watch the running status, logs of CPE by resources in ISP.

A lot of our customers use this system to built and manage complex networks they designed.

The SMF consists of server side system and CPE side system. IIJ uses NetBSD to create the intelligent CPE side system. The system is named 'recipe framework', and is developed using C language, libraries and scripting languages. We mainly have used C languages for many years. In 2013, we began to use mruby[1], a variant of Ruby scripting language, to control NetBSD based CPE itself. It's little difficult for C language to avoid buffer overflows. Using scripting language can avoid the problem, so we use it for performance independent part.

2.2 Manageability of UNIX like OS based CPE

For end users, this is not free UNIX like systems but CPE. So the following things are important:

easy to understand
Easily and uniformly understandable without any knowledge of based OS. Easy to understand what happened. Easy to understand it's problem or not. Easy to understand what is a problem.

stability
Not do panic. Strong against attack from others.

automation
Automatically change setting if it can. Usually, some changes are done by editing files under /etc on UNIX like OS, and some functions don't consider changes of other functions. If a change is deterministic in system wide, it should be done automatically.

3 Development of SEIL

3.1 Extending device drivers

CPE has a number of network devices that is not common in desktop operating systems. CPE also has a naming scheme of the devices that is different from UNIX culture. IIJ has added some new H/W device drivers and special *if_net.if_xname* handling code.

Some CPE support networking port other than Ethernet. For example, SEIL supports ISDN BRI(IPAC-X on a simple device BUS), 3G MODEM(emulates USB serial port), LTE MDOEM(emulates USB Ethernet device). Most peoples weren't interested in IPAC-X, so we wrote the driver from scratch. Most people want the drivers for 3G/LTE modems, but there are no specification document. IIJ haven't had enough document in the fact, but we have tried to write the device driver and hold down the ugly BUGs of the 3G/LTE modems. The modems are managed by userland daemon 'connmgrd'. The daemon manipulates various type of P2P connections such as ISDN, 3G modem, L2TP, and so on. 802.11 wireless networking device that supports AP-mode is also a topic on CPE. NetBSD has its own 802.11 stack, but IIJ has ported vendor genuine, but buggy, 802.11 wireless support to NetBSD.

IIJ also has modified basic functionalities of NetBSD's network device. We can change the device name via ifconfig. Because port name of the CPE such as lan1, lan2, are far different from BSD's device name such as *wm*0, *em*0, *bge*0, and so on. Of course, we can translate CPE's configuration name to NetBSD device name. If we think about logging, using existing daemons, to change the device name is most cost effective way.

There are Ethernet switching devices in CPE. NetBSD has no framework to manage Ethernet switching functions such as Port based VLAN, per-port link status detection, learning table separation. IIJ wrote simple framework to do this and configuration command named *swconfig* [2].

We also change queuing strategy of network devices to work with link status detection. For example, to queue packets to link-downed network device wastes mbuf resources though such old packets are useless. And, it's sometime dangerous because some old packet might cause the network trouble. So our implementation drops the packet as soon as possible if there were some problems on link state or protocol state.

There are some pseudo networking device implemented by IIJ. For example, IPsec tunneling device[2], paravirtualized Ethernet driver for Hyper-V. FreeBSD also has a Hyper-V driver. There is no functional difference between FreeBSD's one and IIJ's one. The reason why we implemented a driver is simple, there was no FreeBSD driver when IIJ needed it. These are such duplicated implementation, and IIJ's implementation is not so special, but some of these can be useful.

3.2 Extending IP Networking stack

The IP networking stack is the most important part of CPEs. We need both of the routing speed and additional functionality. CPE is placed on a border between the ISP and the customer's LAN. CPE doesn't require very high performance such as 10G Ethernet, but require to absorb characteristics of each customer and to maximize the benefits of the ISP services.

Several years ago, IIJ implemented own IP filter and NAT/NAPT functions named *iipf* and *iipfnat*. NetBSD had an IP filter implementation *ipf*, however it didn't satisfy our requirements. *pf* and *npf* wasn't born yet. *iipf* had implemented some ideas to improve throughput. It will be described in another paper[2].

Our IPsec stack also has a caching layer on Security Policy Database (SPD) and Security Association Database (SAD). IPsec tunneling is also important for VPN; many customers prefer Route-based VPN to Policy-based VPN. This topic will be described in another paper[2].

A CPE typically uses a very cheap CPU, thus cryptographic accelerators are very important components. IIJ has done many efforts to use the accelerators effectively, and implemented a framework to use the accelerators. Using accelerators in a C function (i.e., single kernel context) was possible, but the resulting performance was very slow. So IIJ decided to separate IP stack into two parts, "before IPsec" and "after IPsec". This strategy is same as *opencrypt*(9) subsystem and *FAST_IPSEC*(4) stack that stems from OpenBSD. This framework works fine today, so IIJ has decided to use it. (Though there were some minor problems fixed by IIJ, the performance of the framework is fairly good now.)

A number of network interfaces can be causes of many problems. For a desktop machine, there are just a few network interfaces. But for a CPE, there can be many pseudo network interfaces which provide various tunnel connections. If some code uses a simple list to manage the interfaces, it becomes very slow, and consumes a large amount of memory. For example, *getifaddrs*() function uses a large memory footprint in both of the kernel and userland processes if there are a lot of interfaces. IIJ has added selector and cache layers on *getifaddrs*(). We can get a list of interfaces which link-state is up by using *getifaddrs_up*() for example.

There are some common hacks on CPEs, such as TCP-MSS clumping to avoid the Path MTU Discovery problem and Session Hijacking to create transparent proxy. IIJ has own implementations against them to satisfy requirements from customers.

3.3 Implementing New Network Protocols

IIJ has implemented some network tunneling protocols on NetBSD. PPTP and L2TP protocols are implemented in NetBSD userland. There is also an in-kernel cut-through forwarding mechanism named PIPEX. These functions are already merged to OpenBSD [1].

We has an implementation of L2TPv3. The L2TPv3 is a kind of Ethernet encapsulation and tunneling protocol described in RFC3931. The L2TPv3 network device acts as a kind of Ethernet device, and can be added to an Ethernet bridging group. Our CPE can bridge separated Ethernet segments via an L2TPv3 network device. If multiple L2TPv3 network devices are added to one bridging group, the CPE acts as a virtual Ethernet HUB.

There are also some experimental implementations of new Internet Drafts. For example, IIJ has a MAP (`draft-ietf-softwire-map-xx`) implementation. Because IIJ is an ISP company, so we are so interested in new protocols. They are not a standard protocol yet, but experimental implementations are important to standardize good protocols. The development of L2TPv3 is one of successful efforts. It had been started with a project that develops Internet Draft of L2TPv3 (`draft-ietf-l2tpext-l2tp-base-xx`).

Most CPEs support the UPnP protocol. IIJ implements UPnP services from scratch. They are completely different from *libupnp* based implementations. They are designed to cooperate with *iipfnat* and control *iipfnat* rules by the UPnP protocol.

[1]The merge has been done by `yasuoka@openbsd.org`

IIJ implements '*sockfromto*' which is a collection of extended socket API used in some operating systems. The typical functions of sockfromto are *sendfromto*() and *recvfromto*(). These functions enable to reduce a complicated usage of *bind*(). A sending socket which is bound the source port can cause a unexpected packet arrival to the sending socket. If you used *INADDR_ANY* to receiving socket, and forgot that the sending socket can receive packets, some arrival packets may lost during sending packets. *sendfromto*() can send packet with specified source port without calling *bind*().

3.4 Standing for heavy Ethernet rx interrupts

Traditional BSD system used simple *spl* mechanism. A high priority event always obstructs lower priority events. In CPE case, Ethernet rx interrupt always obstructs IP Networking stack, routing daemons, user interface, and so on. Especially, live-lock of IP Networking stack is serious problem for CPE. IIJ did some efforts to reduce such live-lock. It was serious problem for IIJ, because the live-lock can break our centralized management framework.

At first we tried to control interrupt enable/disable bits, rate control bit of Ethernet devices. What is the trigger to throttle the interrupts? We tried to add some probes that detect stall of IP Networking stack. Checking IP input queue(*ipintrq*) length, checking system load(*kern.cp_time*), checking callout timers, etc, etc..

OpenBSD tell us to control rx buffer works fine, instead of to control the interrupts directly. The idea is implemented as *MCLGETI* API of OpenBSD [2]. IIJ has ported the *MCLGETI* API to NetBSD and does some performance test. We confirm the *MCLGETI* works fine enough by various inspection.

[2]The API has other motivation that reduce memory usage on supporting jumbo frames.

4 Daily workflow

4.1 Creating new products

IIJ has created many products. Here is a list of our common works to create a new product.

- Create plain new port of NetBSD like evbxxx.
- Create customized ramdisk of the product like install kernel.
- Launch an NTP daemon and measure clock jitter/drifts, and tune a clock parameter if needed.
- Send/Receive various sizes of Ethernet frames. Frame with 802.1q VLAN tag often reveals MTU handling problem of Ethernet drivers.
- Check if dmesg buffer (*kern.msgbuf*) is cleared after software reboot. If it is cleared on reboot, fix it not to clear. The buffer is important for debugging.
- Measure primitive performances such as memory access. CPU benchmark(INT, FLOAT), cryptographic benchmark(DES, AES, DH, ..), system calls benchmark. The performance of system calls tell us performance of VM(large copyin/copyout), performance of exception handlers. We often reveal a architecture dependent problem by system call benchmarks.
- Measure IP routing performances using various major commercial measuring equipments. Such measuring equipments are also useful to apply high network load to the product. The load often reveals *spl* bugs.
- Check counters. If an value isn't visible, add it. If an counter is not incremented on some cases, fix it.
- Modify log facility, level and the message. Some logs's level are inadequate for users, so change it. Some log messages might be misunderstood by users, so modify it or remove it. Some event is important. If not log message is generated by the event, add it.

- Throttling log. Some logs might be frequently generated. If it occurred, stability of the system will be bad.

Some of the works are hard to be done by non-commercial hackers due to lack of environments, equipments, and time. If bugs are found, BSD hackers in IIJ sometimes merge the fix for the bugs.

4.2 Debugging NetBSD on small embedded hardware

On developing commercial product, debugging is very important, and we pay very much costs for it. To minimize the costs, IIJ has implemented debugging functionalities for small, embedded devices such as CPE.

IIJ has customized the syslog daemon. The log rotation mechanism on filesystem is works fine in desktop, but it is not always useful in restricted CPE. To minimize memory usage, our syslogd has internal ring buffer to remember logs, and user interface process can get logs via IPC. There are multiple ring buffer per facilities, and user can configure the size of each ring buffer. Most important facility is different for each customers.

A CPE often has no storage device, so there is no room to dump core files. So our kernel report the core information of the core files to dmesg buffer. For example, process name, program counter that causes an exception, back-trace of the userland process. The back-trace doesn't include symbol information, but is useful enough. MIPS processor has no frame pointer so the back-trace is not so trusted.

IIJ extended ddb to debugging networking stack. To print or list socket structure in kernel, To print the last function who touches a mbuf structure. Due to NetBSD-current modifies pool framework to support cache-coloring, some of these function are not working now. We need to re-design these.

Watch dog timer(wdog) is very important component in commercial product. IIJ has implemented wdog framework, and there are many point to kick the wdog. There is genuine wdog framework recent NetBSD, we are surveying it. Configuring wdog is not difficult, but kick the wdog is difficult. Especially to live-lock situation requires very sensitive design. *panic*() is also difficult situation. We do want to see the information from *panic*() and ddb stack dump, but we do avoid the infinite loop in dump. We kick the wdog during dump, but there is limit in the depth of stack. The dump can cause a exception and start new stack dump. We force cpu reboot in such situation.

4.3 Following the evolution of NetBSD

IIJ currently uses NetBSD-6.x as it's base system. Past products used NetBSD-1.x and NetBSD-3.x. Because the evolution of NetBSD is faster than life cycle of our product lines, leaping into a new NetBSD become a hard work for us. Though it's easy to generate a diffs of our implementation, it's sometimes difficult to apply the diffs to a new NetBSD.

Unfortunately the last fifteen years was so tough years, IIJ has not contributed to BSD community enough. Few BSD developers in IIJ have contributed to the community. For example, yasuoka@openbsd.org contributed and has developed PPP implementation '*npppd*' and in kernel PPP cut through mechanism '$PIPEX$' now.

As we wrote above, we have implemented some new functions and have enhanced some current functions, but a lot of have not merged yet.

5 Conclusion

IIJ has developed own CPE named 'SEIL' for long years. The name SEIL was often appeared in NetBSD developers community in the past, but IIJ didn't say much about it. This and [2] are 1st public articles about IIJ's past works and knowledges. IIJ hopes that the articles become some good lessons for BSD communities.

References

[1] mruby https://github.com/mruby/mruby

[2] Masanobu SAITOH and Hiroki SUENAGA, “Implementation and modification for CPE: filter rule optimization, IPsec interface and Ethernet switch” In proceedings of AsiaBSDCon2014, March 2014.