IIR

Internet Infrastructure Review

Feb.2020
Vol. 45

Periodic Observation Report

Internet Trends as Seen from IIJ Infrastructure—2019

Focused Research

Acquiring Forensic Memory Images on Linux

IIJ

Internet Initiative Japan

Internet Infrastructure Review

February 2020 Vol.45

Executive Summary

Here is the final IIR for 2019. Japan suffered many natural disasters in 2019, not least those associated with heavy rainfall events. The heavy downpours in northern Kyushu in August, Typhoon Faxai, which caused widespread damage centering on Kanto in September, and Typhoon Hagibis remain fresh in memory. The term global warming tends to bring to mind rising atmospheric temperatures, but sexperts tell us that rising levels of greenhouse gases in the atmosphere will alter the overall state of the atmosphere, causing rainfall patterns to gradually change.

Companies in the information and communications industry are also taking action by switching to energy-saving equipment and pursuing energy efficiency in the datacenter, for instance. Against this backdrop, it was announced that the prototype of RIKEN's Fugaku supercomputer had taken the top global spot in the Green 500 ranking of the world's most energy-efficient computers.

Information and communications plays a diverse role in building a sustainable world, including through the use of energy-saving equipment and the use of information and communications technology to reduce energy consumption. IIJ will also continue to develop technology with that in mind.

The IIR introduces the wide range of technology that IIJ researches and develops, comprising periodic observation reports that provide an outline of various data IIJ obtains through the daily operation of services, as well as focused research examining specific areas of technology.

The periodic observation report in Chapter 1 presents the 2019 edition of our rundown of Internet trends as viewed from IIJ infrastructure. The report covers data on the number of IPv4 routes on the Internet, an analysis of DNS queries from the full resolver IIJ provides to users, IPv6 usage on the IIJ backbone, traffic on mobile networks and the FLET'S network around the time of natural disasters, and the history of the IIJ backbone.

For the first time, we observed a decline in the number of unique IPv4 addresses in advertised routes, and this will bear watching ahead. Our observations also confirm that the use of IPv6 continues to rise steadily, as evidenced by increasing traffic volumes and increasing use among many service operators. And our analysis of traffic around October 12, when Typhoon Hagibis passed through Japan, showed a clear difference from the usual usage patterns.

The focused research report in Chapter 2 explains Linux memory imaging tools that can be used with the Volatility framework, which we use to analyze memory images when performing incident response and forensics. The report looks at LiME and crash, two tools that can be used to acquire Linux memory images, and describes how to acquire memory images in a way that has minimal impact on disk forensics.

Through activities such as these, IIJ strives to improve and develop its services on a daily basis while maintaining the stability of the Internet. We will continue to provide a variety of services and solutions that our customers can take full advantage of as infrastructure for their corporate activities.

Junichi Shimagami

Mr. Shimagami is a Senior Executive Officer and the CTO of IIJ. His interest in the Internet led to him joining IIJ in September 1996. After engaging in the design and construction of the A-Bone Asia region network spearheaded by IIJ, as well as IIJ's backbone network, he was put in charge of IIJ network services. Since 2015, he has been responsible for network, cloud, and security technology across the board as CTO. In April 2017, he became chairman of the Telecom Services Association of Japan MVNO Council.

Internet Trends as Seen from IIJ Infrastructure —2019

To provide Internet services, IIJ operates some of the largest network and server infrastructure in Japan. Here, we examine and discuss current Internet trends based on information obtained through the operation of this infrastructure.

We cover the topics of network routing information, DNS query information, and IPv6 usage, as well as the impact of natural disasters on mobile and FLET'S connection services. We also report on the current state of the backbone network that supports the bulk of IIJ's traffic.

Topic 1

BGP / Number of Routes

We start by looking at IPv4 full-route information advertised by our network to other organizations (Table 1). This time around, we also show the number of unique IPv4 addresses contained in the IPv4 full-route information (Table 2). During the past year, the maximum size of IPv4 addresses allocated by APNIC (and JPNIC) fell to /23 (512 addresses).

The total number of routes, while seeing a slightly smaller increase than in the previous year, now exceeds 760,000. Together, the /22, /23, and /24 prefixes account for 80.1% of all routes. Meanwhile, the number of unique IPv4 addresses, although accounting for less than 1% of the total, fell for the first time in the past nine years. Whether this is a temporary effect, perhaps due to the removal of unauthorized routes using RPKI, or the first sign that the IPv4 Internet is shrinking is something that will bear close watching ahead.

Table 1: Number of Routes by Prefix Length for Full IPv4 Routes

Date	/8	/9	/10	/11	/12	/13	/14	/15	/16	/17	/18	/19	/20	/21	/22	/23	/24	total
Sep. 2010	20	10	25	67	198	409	718	1308	11225	5389	9225	18532	23267	23380	30451	29811	170701	324736
Sep. 2011	19	12	27	81	233	457	794	1407	11909	5907	9885	19515	26476	26588	35515	34061	190276	363162
Sep. 2012	19	14	29	84	236	471	838	1526	12334	6349	10710	20927	30049	31793	42007	39517	219343	416246
Sep. 2013	16	11	30	93	250	480	903	1613	12748	6652	10971	22588	32202	34900	48915	42440	244822	459634
Sep. 2014	16	12	30	90	261	500	983	1702	13009	7013	11659	24527	35175	37560	54065	47372	268660	502634
Sep. 2015	18	13	36	96	261	500	999	1731	12863	7190	12317	25485	35904	38572	60900	52904	301381	551170
Sep. 2016	16	13	36	101	267	515	1050	1767	13106	7782	12917	25229	38459	40066	67270	58965	335884	603443
Sep. 2017	15	13	36	104	284	552	1047	1861	13391	7619	13385	24672	38704	41630	78779	64549	367474	654115
Sep. 2018	14	11	36	99	292	567	1094	1891	13325	7906	13771	25307	39408	45578	88476	72030	400488	710293
Sep. 2019	10	11	37	98	288	573	1142	1914	13243	7999	13730	25531	40128	47248	95983	77581	438926	764442

Table 2: Total Number of Unique IPv4 Addresses in Full IPv4 Routes

Date	No. of IPv4 addresses
Sep. 2010	2,277,265,152
Sep. 2011	2,470,856,448
Sep. 2012	2,588,775,936
Sep. 2013	2,638,256,384
Sep. 2014	2,705,751,040
Sep. 2015	2,791,345,920
Sep. 2016	2,824,538,880
Sep. 2017	2,852,547,328
Sep. 2018	2,855,087,616
Sep. 2019	2,834,175,488

Next we take a look at IPv6 full-route data (Table 3). The total number of routes increased by more than it did in the previous year and now exceeds 70,000. However, route advertisements for blocks that have been split into smaller fragments still constitute the majority, with the top three year-on-year growth rates coming in the /30–/31, /41–/43, and /45–/47 prefix ranges, sizes that are not all that commonly allocated/assigned.

Lastly, let's also take a look at IPv4/IPv6 full-route Origin AS figures (Table 4). Both the decrease in 16-bit Origin Autonomous System Numbers (ASNs) and the increase in 32-bit-only Origin ASNs were the largest seen in the past nine years. And for the first time, IPv6-enabled ASNs, which advertise IPv6 routes, accounted for over a quarter of the total. It was predicted that RIPE NCC's IPv4 address pool would run out before 2020 arrived, so it will be interesting to see what has happened when we next report on the data.

Topic 2

DNS Query Analysis

IIJ provides a full resolver to enable DNS name resolution for its users. In this section, we discuss the state of name resolution, and analyze and reflect upon data from servers provided mainly for consumer services, based on a day's worth of full resolver observational data obtained on October 25, 2019.

The full resolver starts by looking at the IP address of an authoritative name server for the root zone (the highest level zone), and based on the information available from that server, it then goes through other authoritative name serves to find the records it needs. Queries repeatedly sent to the full resolver can result in increased load and delays, so the information obtained is cached, and when the same query is received again, the response is sent from the cache. Recently, DNS-related functions are also implemented on

Table 3: Number of Routes by Prefix Length for Full IPv6 Routes

Date	/16-/28	/29	/30-/31	/32	/33-/39	/40	/41-/43	/44	/45-/47	/48	total
Sep. 2010	38	3	10	2023	33	2	9	4	17	436	2575
Sep. 2011	68	13	22	3530	406	248	45	87	95	2356	6870
Sep. 2012	102	45	34	4448	757	445	103	246	168	3706	10054
Sep. 2013	117	256	92	5249	1067	660	119	474	266	5442	13742
Sep. 2014	134	481	133	6025	1447	825	248	709	592	7949	18543
Sep. 2015	142	771	168	6846	1808	1150	386	990	648	10570	23479
Sep. 2016	153	1294	216	8110	3092	1445	371	1492	1006	14291	31470
Sep. 2017	158	1757	256	9089	3588	2117	580	1999	1983	18347	39874
Sep. 2018	168	2279	328	10897	4828	2940	906	4015	2270	24616	53247
Sep. 2019	192	2671	606	12664	6914	3870	1566	4590	4165	34224	71462

Table 4: IPv4/IPv6 Full-Route Origin AS Numbers

ASN	16-bit(1-64495)					32-bit only(131072-4199999999)				
Advertised route	IPv4+IPv6	IPv4 only	IPv6 only	total	(IPv6 -enabled)	IPv4+IPv6	IPv4 only	IPv6 only	total	(IPv6 -enabled)
Sep. 2010	2083	32399	67	34549	(6.2%)	17	478	3	498	(4.0%)
Sep. 2011	4258	32756	115	37129	(11.8%)	90	1278	13	1381	(7.5%)
Sep. 2012	5467	33434	125	39026	(14.3%)	264	2565	17	2846	(9.9%)
Sep. 2013	6579	34108	131	40818	(16.4%)	496	3390	28	3914	(13.4%)
Sep. 2014	7405	34555	128	42088	(17.9%)	868	4749	55	5672	(16.3%)
Sep. 2015	8228	34544	137	42909	(19.5%)	1424	6801	78	8303	(18.1%)
Sep. 2016	9116	33555	158	42829	(21.7%)	2406	9391	146	11943	(21.4%)
Sep. 2017	9603	32731	181	42515	(23.0%)	3214	12379	207	15800	(21.7%)
Sep. 2018	10199	31960	176	42335	(24.5%)	4379	14874	308	19561	(24.0%)
Sep. 2019	10642	31164	206	42012	(25.8%)	5790	17409	432	23631	(26.3%)

devices that lie on route paths, such as broadband routers and firewalls, and these devices are sometimes involved in relaying DNS queries and applying control policies.

ISPs notify users of the IP address of full resolvers via various protocols, including PPP, DHCP, RA, and PCO, depending on the connection type, and they enable users to automatically configure which full resolver to use for name resolution on their devices. ISPs can notify users of multiple full resolvers, and users can specify which full resolver to use, and add full resolvers, by altering settings in their OS, browser, or elsewhere. When more than one full resolver is configured on a device, which one ends up being used depends on the device's implementation or the application, so any given full resolver is not aware of how many queries a user is sending in total. When running full resolvers, therefore, this means that you need to keep track of query trends and always keep some processing power in reserve.

Observational data on the full resolver provided by IIJ show fluctuations in user query volume throughout the day, with volume hitting a daily trough of about 0.05 queries/sec per source IP address at around 4:30 a.m., and a peak of about 0.23 queries/sec per source IP address at around 12:30 p.m. These values are almost the same as last year, with a slight increase in the peak of 0.01 points. Broken down by protocol (IPv4 and IPv6), the trends in query volume are virtually the same (no major differences) in the middle of the night, whereas IPv6 queries per IP address show a tendency to rise when people are active during the daytime and particularly after 8:00 p.m. This suggests that the computing environment needed to allow use of IPv6 in the home is coming into place. And looking at total query count, both the number of source IPs and the number of actual queries are higher for IPv6 than for IPv4. The number of IPv6-based queries is on the rise, accounting for around 60% of the total, up by more than 4 points from 55% in the previous year.

Recent years have seen a tendency for queries to rise briefly at certain round-number times, such as on the hour marks in the morning. The number of query sources also increases, which tells us that this is possibly due to tasks scheduled on user devices and increases in automated network access that occur when devices are activated by, for example, an alarm clock function. In the previous year, we noted an increase in queries 14 seconds before every hour mark, and the 2019 results also show another increase 10 seconds before every hour. The increase in queries that occurs on the hour tapers off gradually, but with the spikes that occurs 14 and 10 seconds before the hour, query volume immediately returns to about where it had been. Hence, because a large number of devices are sending queries in almost perfect sync, it seems like some sort of lightweight, quickly completed tasks are being executed.

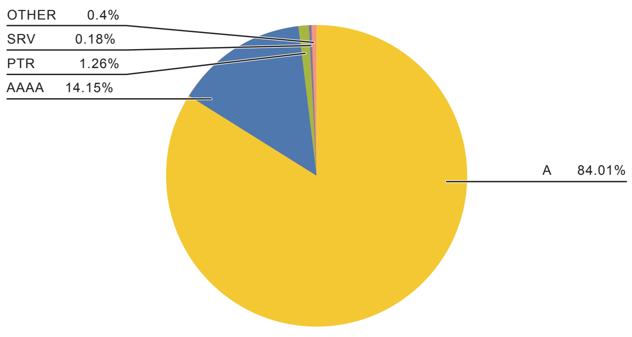

Figure 1: IPv4-based Queries from Clients

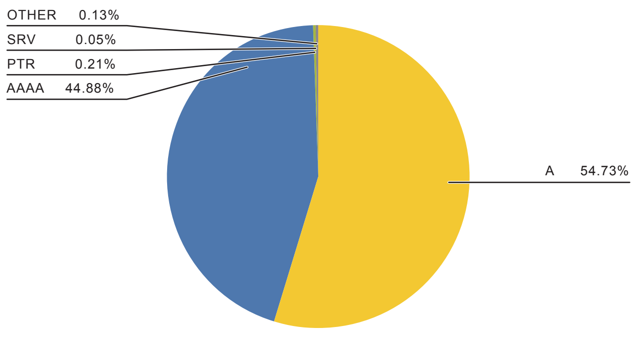

Figure 2: IPv6-based Queries from Clients

For example, there are mechanisms for completing basic tasks, such as connectivity tests or time synchronization, before bringing a device fully out of sleep mode, and we posit that the queries used for these tasks are behind the spikes.

Looking at the query record types, most are A records that query the IPv4 address corresponding to the host name and AAAA records that query IPv6 addresses. The trends in A and AAAA queries differ by IP protocol, with more AAAA record queries being seen for IPv6-based queries. Of IPv4-based queries, around 84% are A record queries and 14% AAAA record queries (Figure 1). With IPv6-based queries, meanwhile, AAAA record queries account for a higher share of the total, with around 54% being A record and 44% being AAAA record queries (Figure 2). Compared with the previous year, the levels are virtually the same for IPv6, while for IPv4, we observe a drop of 3 percentage points or so in A record queries and a rise of 3 points or so in AAAA record queries.

Topic 3

IPv6

In this section, we report on the volume of IPv6 traffic on the IIJ backbone, the sources of that traffic, and the main protocols used. And for a new perspective on IPv6 in the mobile space, we look at the state of IPv6 connections according to differences in device OS (Apple iOS / Android).

■ Traffic

As before, we again present IPv4 and IPv6 traffic measured using IIJ backbone routers at core POPs (points of presence—Tokyo, Osaka, Nagoya), shown in Figure 3. The data span the year from October 1, 2018 to September 30, 2019.

Over the year, IPv4 traffic increased by around 8% while IPv6 traffic rose by around 85%. IPv6 accounts for around 10% of overall traffic (Figure 4), a 4-point increase from around 6% last year. Further, IPv6 traffic hit a peak of

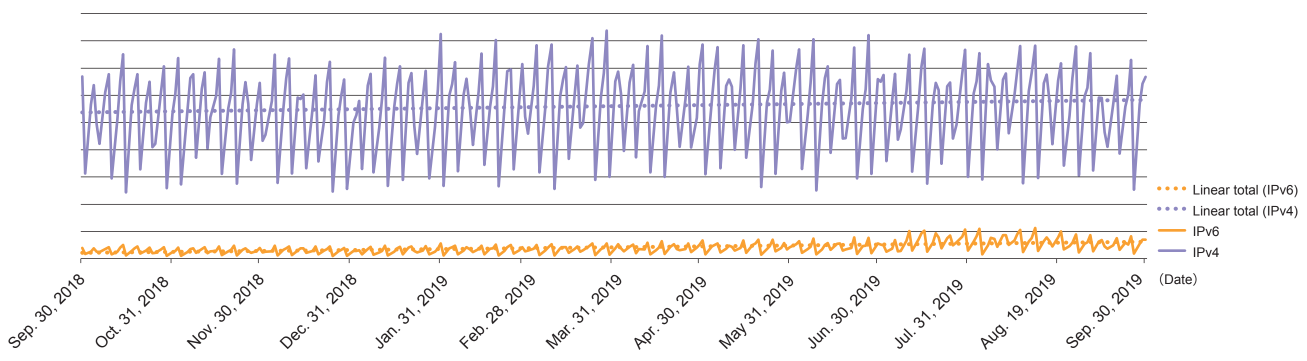

Figure 3: IPv4/IPv6 Traffic Measured via IIJ Backbone Routers at Core Points of Presence (Tokyo, Osaka, and Nagoya)

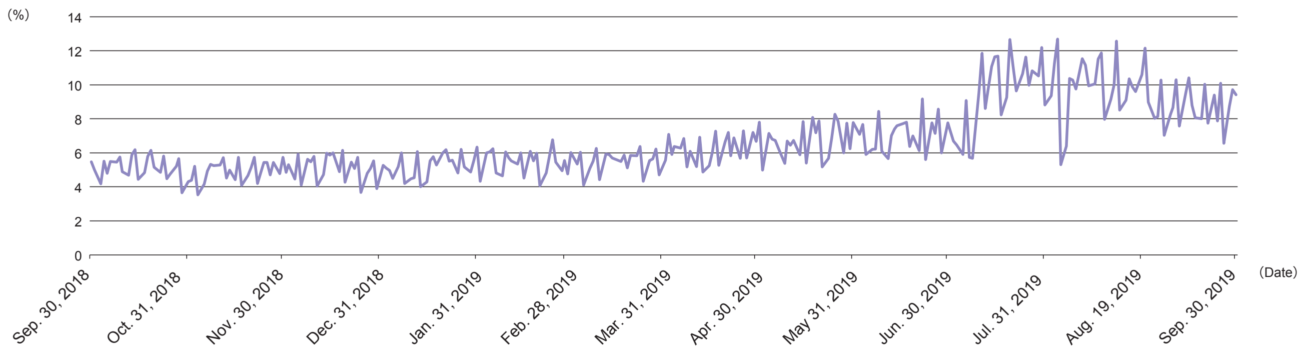

Figure 4: IPv6 Traffic as a Proportion of Total

around 12% of total, and it continues to increase steadily as a proportion of overall traffic.

Figure 5 plots the data for the same period on a log scale. It is evident that IPv6 traffic is growing steadily while growth in IPv4 traffic is slowing.

Traffic Source Organization (BGP AS)

Next, Figures 6 and 7 show the top annual average IPv6 and IPv4 traffic source organizations (BGP AS Number) for the year from October 2018 through September 2019.

Company A retains the top spot, but the traffic volume gap between it and No. 2 downward has narrowed further, and its share of the pie is only about 60% of what it was last time. This reflects the increasing use of IPv6 among many operators as well as an increase in IIJ's IPv6 traffic due to IPv6 being enabled on video streaming services that use JOCDN's platform (JOCDN is an IIJ affiliate that provides a video streaming platform).

Protocols Used

Figure 8 plots IPv6 traffic according to protocol number (Next Header) and source port number, and Figure 9 plots IPv4 traffic according to protocol number and source port number (for the week starting September 30, 2019).

In the IPv6 space, TCP 80 (HTTP) moved from No. 3 last time to No. 2, and UUD 443 (QUIC) came in at No. 3. This mirrors the IPv4 ranking, so we can now say that IPv6

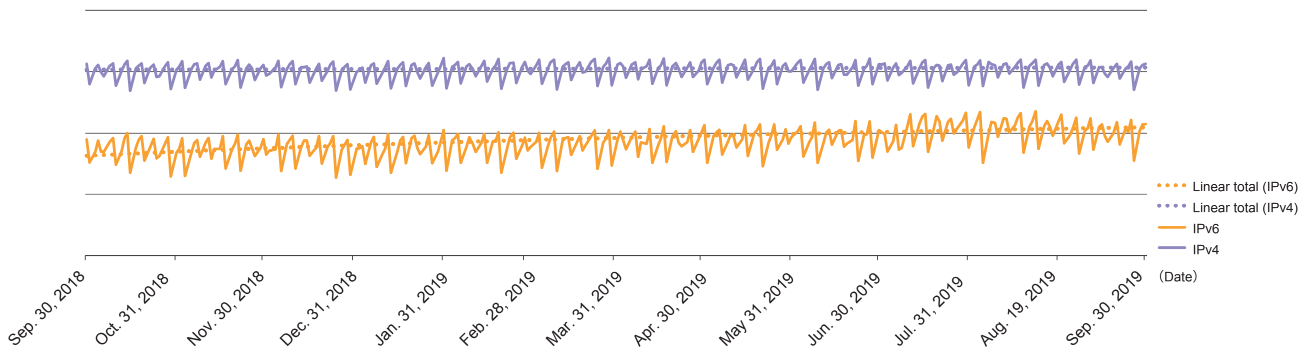

Figure 5: IPv4/IPv6 Traffic Measured via IIJ Backbone Routers at Core Points of Presence (Tokyo, Osaka, and Nagoya)—Log Scale

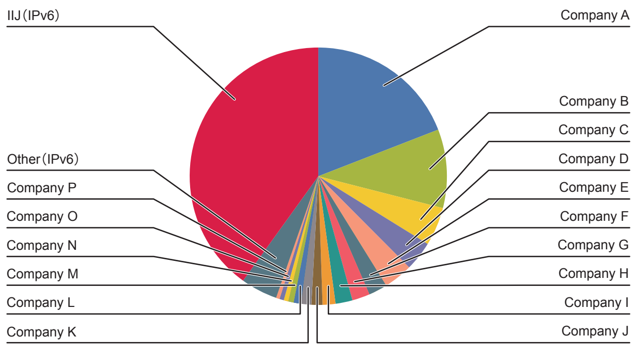

Figure 6: Top Annual Average IPv6 Traffic Source Organizations (BGP AS Number) from October 2018 to September 2019

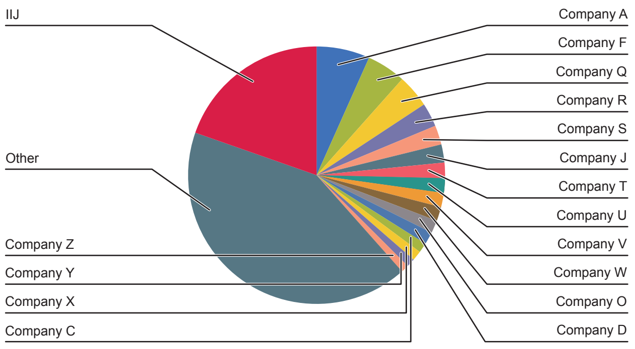

Figure 7: Top Annual Average IPv4 Traffic Source Organizations (BGP AS Number) from October 2018 to September 2019

usage is similar to IPv4 usage or, in other words, that it has become mainstream.

UDP 4500, which was outside the rankings last time, came in at No. 5. NAT is basically not used with IPv6, and it is curious to note that UDP 4500, which is generally for IPSec NAT traversal, ranks toward the top.

IPv6 Across Different Device OSs

Last time, we noted that IPv6 is enabled by default in Apple iOS from version 11, and that mobile IPv6 traffic had thus increased.

This time around, we look at mobile device IMEI (International Mobile Equipment Identity) numbers to facilitate an analysis of OS type (iOS or Android) and whether the device is connecting via IPv6 or not (whether an IPv6 address is assigned).

The analysis covers around 1.07 million mobile phones lines on a personal mobile service (IIJmio Mobile Service, number of lines subscribed to as of end-June 2019) that were connected on a particular weekday in October 2019 (MVNE lines and business lines are not included).

First, IPv6 was enabled on 48% and disabled on 52% of connections, a fairly even split, as shown in FIgure 10. While the connections are split fairly evenly, the traffic is about 80% IPv4 and 20% IPv6.

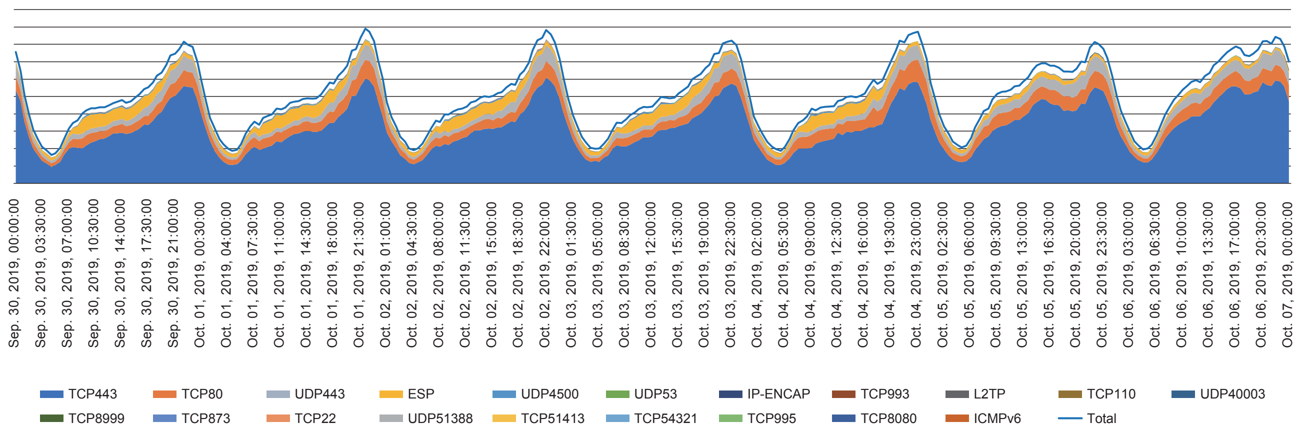

Figure 8: Breakdown of IPv6 Traffic by Protocol Number (Next Header) and Source Port Number

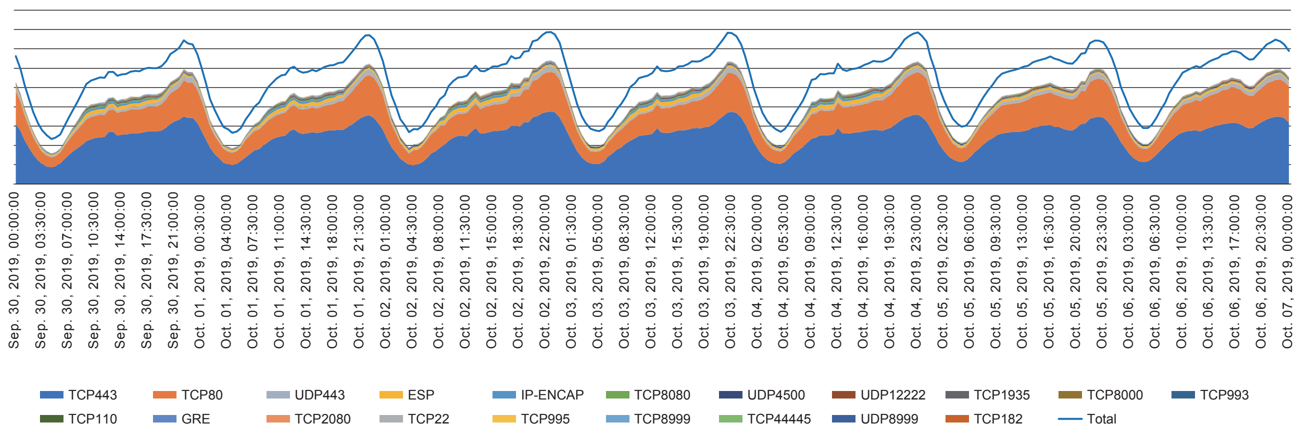

Figure 9: Breakdown of IPv4 Traffic by Protocol Number and Source Port Number

We first look at which OS the IPv6-enabled devices are using. As Figure 11 shows, over 80% are on Apple iOS while 14% are on Android. And next we look at the OS on devices where IPv6 is disabled. As Figure 12 shows, the proportions here are inverted, with Android on 82% and iOS on 8%.

Android has supported IPv6 since the release of Android 5 in 2014, much earlier than Apple enabled IPv6 support with iOS 11, but IPv6 is often disabled by default on Android per device manufacturer and MNO policies. So from an IPv6 viewpoint, a stark difference has arisen between Android and iOS, which is exclusively controlled by Apple.

Nonetheless, IPv6 is disabled on some Apple iOS devices; 9.21% of all iOS devices to be precise. We would guess that these are older devices that do not support iOS 11 or higher. IPv6 is enabled on 14.08% of all Android devices, and it appears that many of the recent SIM-lock-free devices have IPv6 enabled.

Summary

In this issue, we examined IPv6 traffic volume and protocols used, as well as IPv6 connection rates based on mobile device OS. While IPv4 traffic growth is slowing, IPv6 traffic exhibited similar levels of growth to last time, indicating that use of IPv6 continues to advance. Many recently available home Wi-Fi routers offer support for IPv6 IPoE, and there are moves to enable IPv6 on video streaming services as well, so IPv6 traffic looks set to rise even further.

IPv6 is available in the mobile space more than we had imagined, being enabled on almost half of connections. It still only accounts for around 20% of all traffic, so we hope that services continue to provide even more IPv6 support ahead.

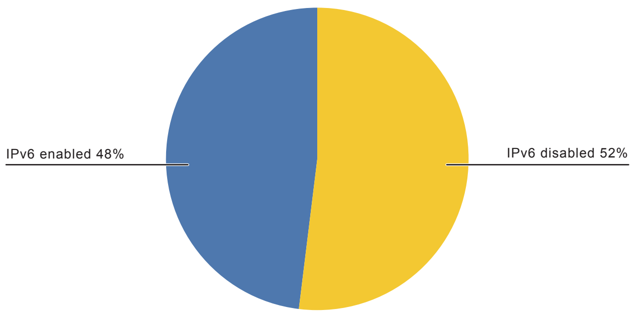

Figure 10: Proportion of Connections with IPv6 Enabled

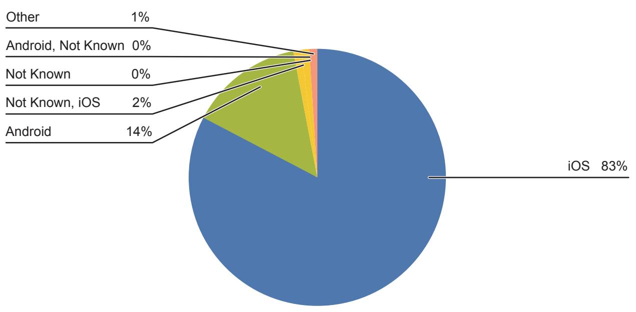

Figure 11: OS Breakdown for IPv6-enabled Devices

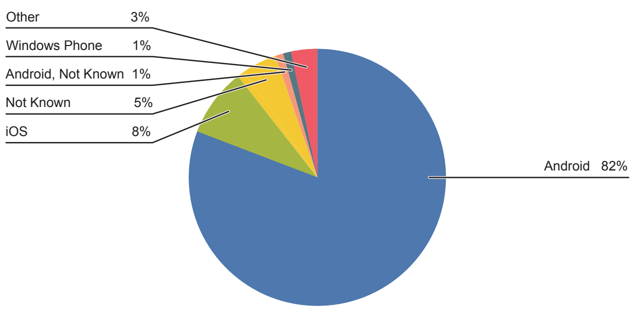

Figure 12: OS Breakdown for IPv6-disabled Devices

Topic 4

Mobile FLET'S and Natural Disasters

Usually, we would report on changes in the trends in FLET'S and mobile traffic based on comparable observations, but virtually the same content was presented in our periodic observation report in Vol. 44 (https://www.iij.ad.jp/en/dev/iir/044.html), so here we look at the effects of natural disasters.

Many natural disasters occurred in 2019. Typhoon Hagibis, in particular, wrought serious damage across many areas. On the IIJ backbone, several lines connecting Tokyo and Osaka experienced lengthy disconnections. A road was washed away in Nagano Prefecture (between Tokyo and Osaka), and buried optical fiber cables were physically damaged. The damage spanned a long section of the route, and the recovery took weeks. Fortunately, IIJ's backbone network is designed to handle faults caused by events such as natural disasters like this. The Tokyo–Osaka leg is distributed among Pacific Ocean, Sea of Japan, and inland routes, so there was no real impact on user communications.

Meanwhile, Typhoon Hagibis had a clear impact on access services linked directly users' use of mobile and FLET'S services. Let's examine this with reference to the data. Note that FLET'S here refers only to PPPoE.

Figures 13 to 24 graph number of connections and traffic volume around October 12, 2019, when Hagibis hit Japan's main island, along with the corresponding data for a week earlier around October 5, 2019 for comparison (upload and download). All of the series have been indexed to a value of 1 at 12 a.m. on the Fridays (October 4 and 11, 2019).

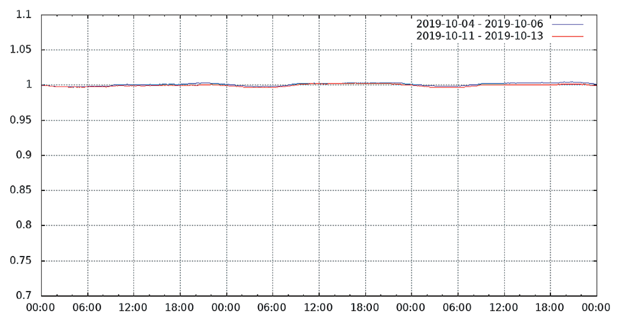

Figure 13: Number of FLET'S Connections in the Osaka Area

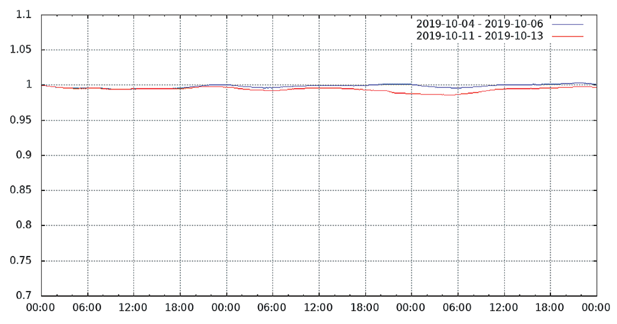

Figure 15: Number of FLET'S Connections in the Tokyo Area

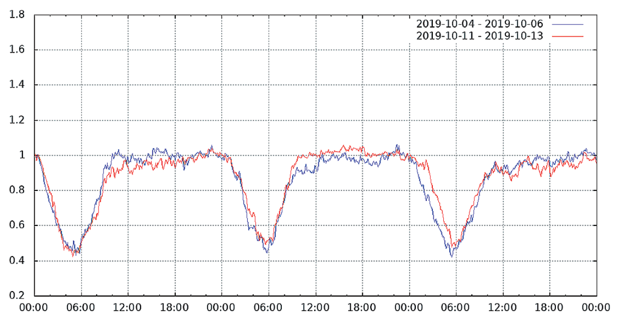

Figure 14: FLET'S Traffic Volume in the Osaka Area

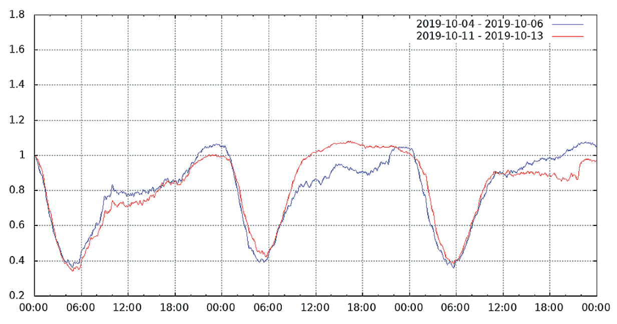

Figure 16: FLET'S Traffic Volume in the Tokyo Area

First, we focus on the number of connections. Currently, most FLET'S users have always-on connections via broadband routers and home gateways, so connection volume remains fairly constant throughout the day. Indeed, connection volume changed only slightly during the period in the Osaka area, which was largely unaffected by Typhoon Hagibis.

However, the number of FLET'S connections in the heavily impacted Chiba, Fukushima, and Miyagi areas declined over October 12 and 13, 2019. It dropped sharply in the FLET'S Chiba area in particular. These areas experienced many power outages, so the drops were likely due to household broadband routers and home gateways going offline when the power dropped out. For mobile, meanwhile, the nature of MVNO equipment precludes the ability to determine device location, so we use nationwide totals. The data show that the number of connections fell over October 12 and 13, 2019. Possible explanations are that MNO equipment

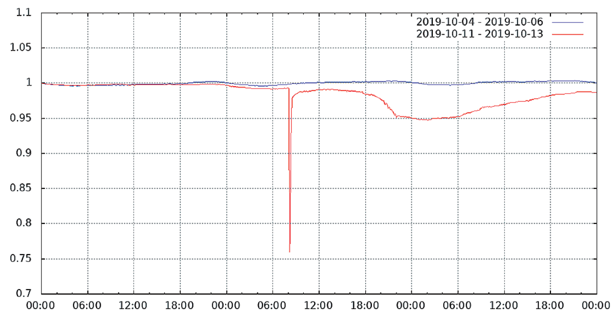

Figure 17: Number of FLET'S Connections in the Chiba Area

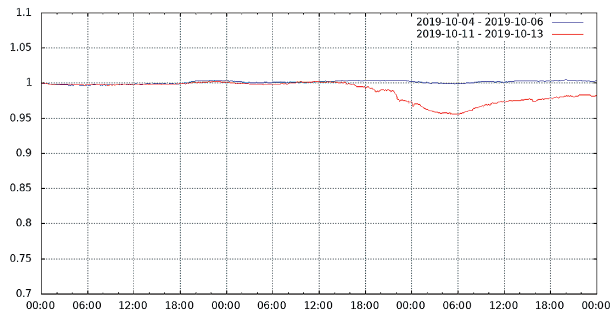

Figure 19: Number of FLET'S Connections in the Fukushima Area

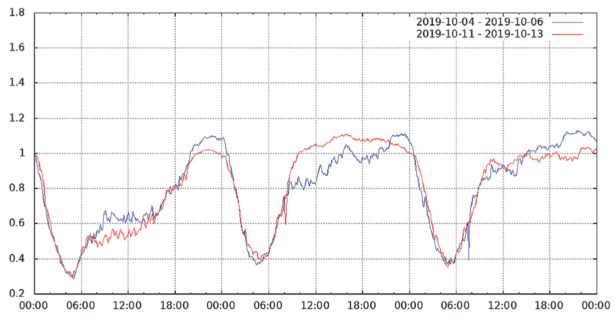

Figure 18: FLET'S Traffic Volume in the Chiba Area

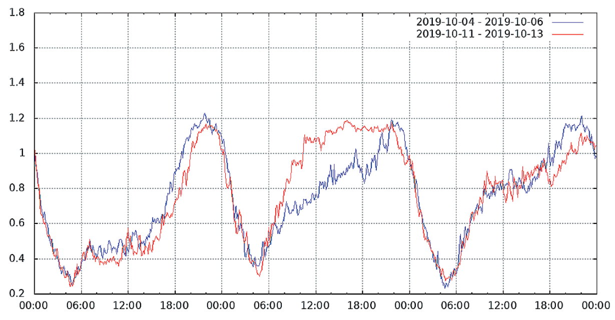

Figure 20: FLET'S Traffic Volume in the Fukushima Area

was also damaged or that people refrained from going out and thus did not use their mobile connection.

Next we look at traffic volume. In Tokyo and areas to the north, FLET'S traffic volume increased during the daytime on October 12, 2019, while mobile traffic fell heavily. It was a weekend, but many people probably stayed indoors to avoid the storm, using their FLET'S broadband at home instead of mobile data.

The Internet is now an important part of our infrastructure alongside electricity, gas, and water, and one that is all the more valuable during natural disasters. IIJ will continue to design its backbone with due consideration to the potential impact of natural disasters and provide Internet infrastructure that users can rest assured will be available at all times.

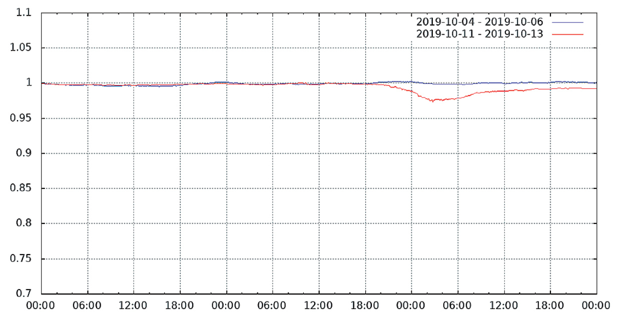

Figure 21: Number of FLET'S Connections in the Miyagi Area

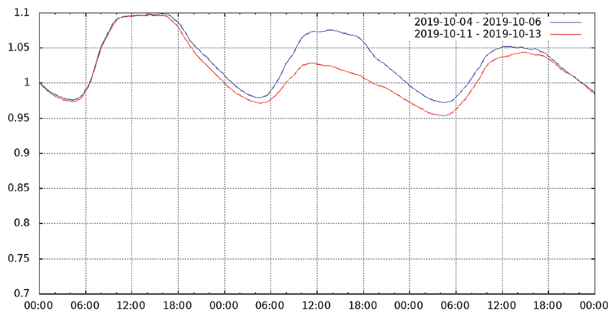

Figure 23: Number of Mobile Connections

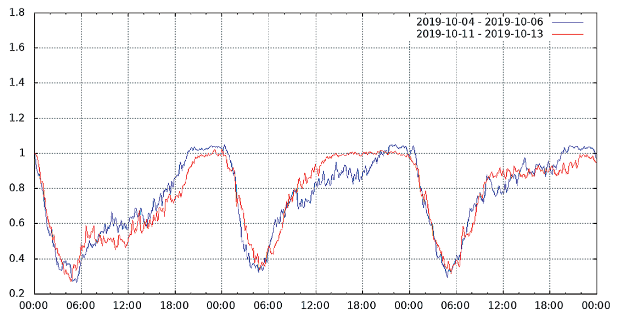

Figure 22: FLET'S Traffic Volume in the Miyagi Area

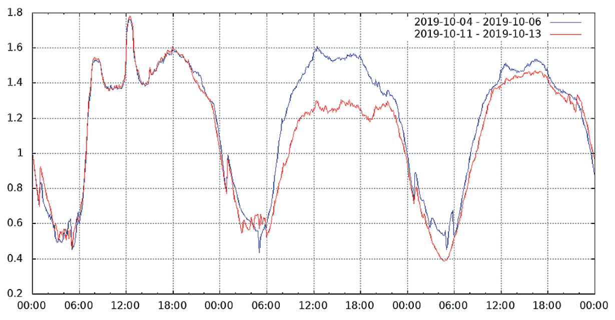

Figure 24: Mobile Traffic Volume

Topic 5

The Transition of the Bandwidth of IIJ Backbone Circuits

In the 2018 edition (Vol. 41) of this report, we noted that total traffic on IIJ's backbone had grown more than tenfold in the past 10 years. Here, we look back on the transition of IIJ's backbone, particularly the bandwidth of the backbone circuits. IIJ began its ISP operations as a Type 2 Telecommunication Carrier; License for the carrier providing telecommunication services using facilities or circuits leased from Type 1 Telecommunication Carriers. So from the outset, we have leased circuits installed by Type 1 Telecommunication Carriers to build a network and provide our services. And this situation is unchanged even after the repeal of the distinction between Type 1 and Type 2 carriers.

We still have the backbone map from 1998. Most of the circuits' bandwidth used at that time were 45Mbps (DS-3), with some being 155Mbps (STM-1). The map shows how the backbone circuit has changed out from 1998 gradually.

For a time, it was the case that circuits and router interfaces with quadruple the bandwidth were released roughly every two years. Bandwidth continued to expand to 600Mbps (STM-4) in 2000, 2.4Gbps (STM-16) in 2002, and 9.6Gbps (STM-64) in 2004. It was a happy era in which circuit capacity expanded as traffic grew. While the circumstances may have been different for larger ISPs, IIJ was mostly able to keep up with the growing traffic by upgrading circuits without significantly changing its network topology.

The following graph, based on the backbone map, plots total backbone bandwidth between Japan and the US along with the highest-capacity circuits that were available in the backbone. The circuit bandwidth steadily increased from 1998 up until when STM-64 rolled out in 2004. But after 2005 through 2014, over 9 years, circuit bandwidth above

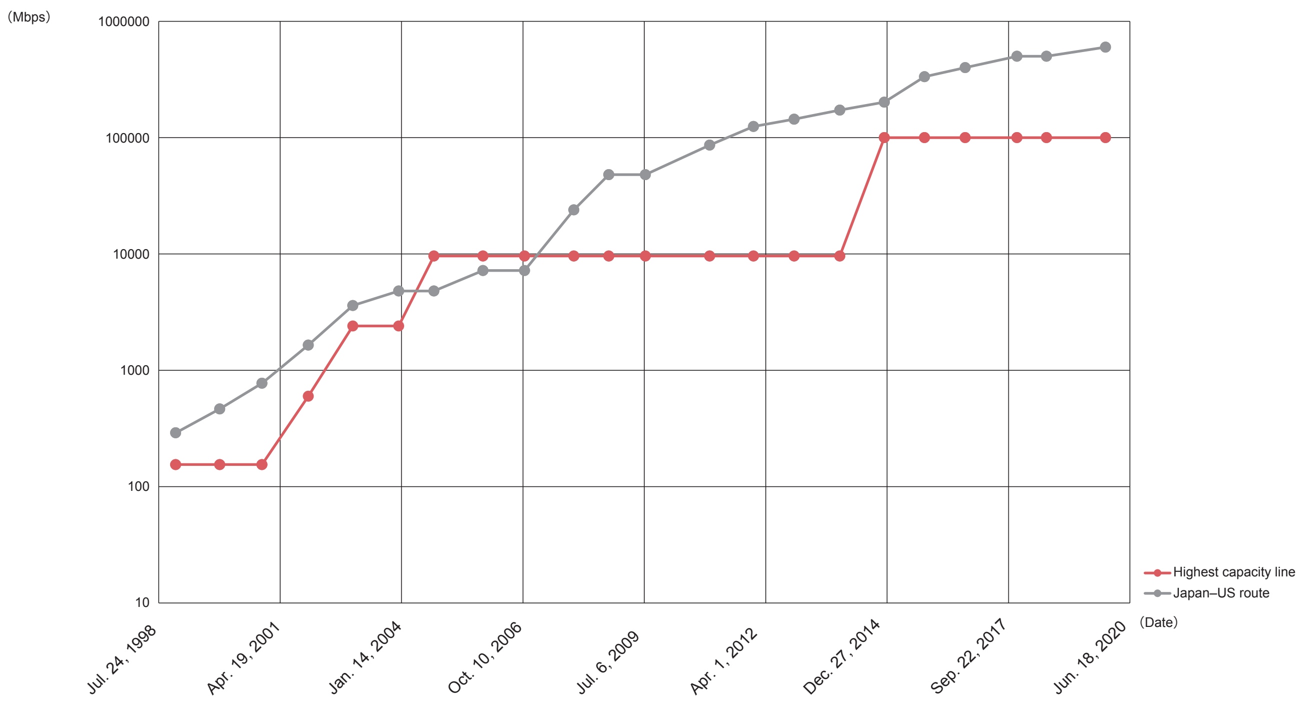

Figure 25: Line Capacity and Japan–US Bandwidth

STM-64 did not become standard. Various factors were behind this. STM-256 was not all that widely adopted, and technical difficulties associated with 100G, the next step up, caused development and standardization delays. Yet traffic continued to grow over this period, eventually reaching the point that we needed bandwidth of 200Gbps or 20 STM-64 circuits between Japan and the US. (Note that the graph's y-axis is logarithmic!)

Hence, the design of the IIJ backbone changed considerably over this period. To change our backbone topology freely, we adopted the MPLS abstract layer, and to boost circuit utilization efficiency, we altered the network topology to make it more amenable to ECMP (Equal Cost Multi-Path).

Things settled somewhat once 100G ethernet became standard in 2014. IIJ also finished converting core backbone circuits to 100G in 2018 and is in the process of upgrading the other network to 100G as traffic grows.

Meanwhile, a sixth 100G circuit between Japan and the US is in the schedule to go up in December 2019. With five years having passed since 100G went mainstream, we're starting to search for the next step up in capacity. The footsteps of 400G are drawing gradually closer.

IIJ will continue to follow the latest technological developments as it expands our backbone to ensure it remains fast and efficient with high availability.

1.BGP / Number of Route
Tomohiko Kurahashi
Technology Development Section, Operation Technology Department, Infrastructure Engineering Division, IIJ.

2.DNS Query Analysis
Yoshinobu Matsuzaki
Technology Development Section, Operation Technology Department, Infrastructure Engineering Division, IIJ

3.IPv6
Taisuke Sasaki
Deputy General Manager, Network Technology Department, Infrastructure Engineering Division, IIJ

4.Mobile FLET'S and Natural Disasters
Takafusa Hori
Manager, Network Technology Section, Network Technology Department, Infrastructure Engineering Division, IIJ

5.The Transition of the Bandwidth of IIJ Backbone Circuits
Takanori Sasai
Manager, Backbone Technology Section, Network Technology Department, Infrastructure Engineering Division, IIJ

Acquiring Forensic Memory Images on Linux

2.1 Linux Memory Dump Toolsn

We use the Volatility Framework when performing memory analysis for incident response and forensics. In IIR Vol. 32, “1.4.1 Creating Profiles for the Volatility Framework,” we explained the procedure for creating a Volatility profile for Linux[*1].

Here, we explain how to acquire Linux memory images for analysis in Volatility. Several tools for acquiring Linux memory images exist, but here we will look at LiME[*2] and crash[*3]. We will also describe a memory image acquisition method that minimizes the impact on disk forensics. Note that this article assumes your system is running CentOS 7.7-1908. Another tool for acquiring memory images is Linpmem[*4], but it did not work well in our test environment, so we decided to leave it out.

Note that in this article we refer to the machine being examined as the “target host” and the machine on which the examination is performed as the “examination host”. To avoid changing data on the target host as much as possible, compilation of the analysis tools, etc., should be done on a separate host.

2.2 What is LiME?

LiME is short for Linux Memory Extractor, and is the tool that Volatility recommends[*5] for acquiring memory images. Since LiME operates as a Linux kernel module, it must be compiled against the same kernel version as that running on the target host.

2.3 Compiling LiME

To compile LiME, you first obtain the LiME source code by running the git command in Figure 1 or by downloading the zip file from LiME’s GitHub page and extracting it into a directory. The src subdirectory inside the source code directory contains a Makefile, so running the make command from within that subdirectory will generate the LiME module

```
$ git clone https://github.com/504ensicsLabs/LiME.git
```

Figure 1: Cloning the LiME Git Repository

```
$ cd LiME/src
$ make
$ ls
disk.c  lime-3.10.0-1062.el7.x86_64.ko  lime.o    Makefile.sample  tcp.o
disk.o  lime.h                          main.c    modules.order
hash.c  lime.mod.c                      main.o    Module.symvers
hash.o  lime.mod.o                      Makefile  tcp.c
```

Figure 2: Compiling the LiME Module

```
$ sudo yum install kernel-devel gcc
```

Figure 3: Installing the kernel-devel Package

*1 Internet Infrastructure Review (IIR) Vol. 32, 1.4.1 Creating Profiles for the Volatility Framework (https://www.iij.ad.jp/en/dev/iir/032.html).

*2 LiME (https://github.com/504ensicsLabs/LiME).

*3 crash (https://people.redhat.com/anderson/).

*4 Velocidex/c-aff4 (https://github.com/Velocidex/c-aff4/releases).

*5 Linux · volatilityfoundation/volatility Wiki (https://github.com/volatilityfoundation/volatility/wiki/Linux#acquiring-memory).

(see the red outline in Figure 2). Certain packages are required to compile LiME (kernel-devel, gcc, etc.), so if the make command produces an error, try installing the corresponding packages as shown in Figure 3.

Note that unless you specify a version at install time, the latest version of the kernel-devel package will be installed. If a different version of the kernel is running on the target host, run a search using the yum command, install the appropriate kernel-devel package version, and then compile the LiME module, as shown in Figure 4. You need to pass the KVER option to the make command when doing this.

If the OS version differs, you can generate a LiME module for that version by downloading the corresponding kernel-devel package separately, extracting it using the cpio command, and then running make with the KVER and KDIR options specified, as shown in Figure 5. KVER specifies the kernel version, and KDIR specifies the directory where the kernel-devel package was extracted. You also need to install any other required packages (on our machine, we needed to install elfutils-libelf-devel). We confirmed that the LiME module generated in this manner works on CentOS 8 (1905).

2.4 Dumping Memory to an External Drive

When acquiring memory images, you need to minimize disk write operations on the target host. In particular, the memory image is almost always several GB or more is size, so writing it to the target host's disk would cause many unused areas to be overwritten, which could greatly affect forensic analysis of the disk.

So if you have physical access to the target host, you can connect a USB stick or mobile SSD containing the LiME module to it and load the LiME module into the kernel from there via the insmod command, as shown in Figure 6. This will let you acquire a memory image while barely writing anything to the target host's disk. The double quotes in Figure

```
$ yum --showduplicates search kernel-devel
$ sudo yum install kernel-devel-3.10.0-1062.1.2.el7.x86_64
$ make KVER=3.10.0-1062.1.2.el7.x86_64
```

Figure 4: Compiling LiME Against a Specific Kernel Version (1)

```
$ curl -OL http://ftp.iij.ad.jp/pub/linux/centos/8.0.1905/BaseOS/x86_64/os/Packages/kernel-devel-4.18.0-80.11.2.el8_0.x86_64.rpm
$ rpm2cpio ./kernel-devel-4.18.0-80.11.2.el8_0.x86_64.rpm | cpio -id
$ sudo yum install elfutils-libelf-devel
$ make KVER=4.18.0-80.11.2.el8_0.x86_64 KDIR=~/src/usr/src/kernels/4.18.0-80.11.2.el8_0.x86_64/
```

Figure 5: Compiling LiME Against a Specific Kernel Version (2)

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=/media/centos77.mem format=lime"
```

Figure 6: Loading the LiME Module

6 contain the LiME module options. In this example, a lime format dump file will be saved to the USB drive mounted at /media under the filename /media/centos77.mem.

The memory dump starts when the LiME module is loaded, but the command prompt does not return until the memory dump is completed. So don't worry if you're unable to enter commands after loading the module; just wait for the dump to complete. More recent servers, in particular, can be expected to take a while because of their large memory capacity. Note that the LiME module remains loaded even after the memory dump finishes, so unload it using the rmmod command, as shown in Figure 7.

2.5 Dumping Memory via the Network

If you do not have physical access to the target host as covered above (for example, if the target host is in a remote location that makes physical access difficult), or if you don't have a large-capacity USB stick, you can combine LiME with other tools to acquire a memory image via the network. Here, we show how to combine it with Netcat, NFS, and SSH. The examination host's IP address is 192.168.232.131, and target host's is 192.168.232.132.

■ Netcat (1)

The LiME module has functionality that enables memory dumps over the network, and we combine this functionality with the Netcat command to acquire a memory image via the network. Once the Netcat command is installed on the examination host, the commands in Figures 8 and 9 will connect the examination host to the port on which the LiME module is listening (4444/tcp) to acquire the memory image data.

■ Netcat (2)

The above procedure will result in data equivalent to the capacity of memory being received. If the target host is a server, that data may take up several dozen GB or more, so

```
$ lsmod | grep lime
$ sudo rmmod lime
```

Figure 7: Unloading the LiME Module

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
```

Figure 8: Listening on 4444/tcp with LiME (Target Host)

```
$ nc 192.168.232.132 4444 > centos77.mem
```

Figure 9: Acquiring a Memory Image via the Network Using Netcat (Examination Host)

```
$ nc -l 5555 > memorydump.lime.gz
```

Figure 10: Command Executed on the Examination Host

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
Switch to another login session to execute the following command
$ nc localhost 4444 | gzip -c | nc 192.168.232.131 5555
```

Figure 11: Commands Executed on the Target Host

```
$ sudo yum install nfs-utils
$ sudo mkdir /mnt/nfsserv/
$ chown -R nfsnobody:nfsnobody /mnt/nfsserv/
$ sudo vi /etc/exports
$ sudo systemctl start nfs.service
$ sudo systemctl status nfs.service
```

Figure 12: Setting the NFS Server (Examination Host)

```
/mnt/nfsserv/ 192.168.232.132(rw,all_squash)
```

Figure 13: NFS Export Settings on the Examination Host (/etc/exports)

we want to compress it as much as possible. If Netcat is also installed on the target host, executing the commands in Figures 10 and 11 will result in the memory image being compressed using gzip when it is transferred to the examination host. As mentioned, the command prompt does not return when the LiME module is loaded, so the command following the nc command needs to be run from a separate login session.

In this example, we first use Netcat on the examination host to listen on 5555/tcp. Next, we use Netcat on the target host to connect to 4444/tcp, where the LiME module is listening, acquire a memory image, compress it using gzip, and then transfer it to the examination host via Netcat.

■ NFS

If the target host can mount an NFS volume, set up the examination host on a network that the target host has access to. Then export the NFS volume on the examination host with read/write capability. By copying LiME to this NFS volume and mounting it as an NFS from the target host, you can acquire a memory image without creating any files on the target host (Figures 12, 13, 14, and 15).

■ SSH

If you cannot use Netcat or NFS, you can use SSH instead. By executing the commands in Figure 16, you can transfer a memory image to the examination host via SSH. However, this method can only be used in bash. As with the Netcat (2) instructions, the exec command onward must be run from a separate login session.

2.6 What is Crash?

The crash command is a tool for analyzing Linux memory images. The primary purpose of the tool is to perform analysis, but a module can also be used to perform memory dumps. But as there is no RPM package of the crash memory dump module, we have to compile it from source

```
$ sudo firewall-cmd --permanent --add-service=nfs
$ sudo firewall-cmd --reload
$ sudo exportfs -v
```

Figure 14: Configuring the Examination Host's Firewall and Checking the NFS Export

```
$ sudo mount -t nfs 192.168.232.131:/mnt/nfsserv/ /mnt/
$ sudo insmod /mnt/lime-3.10.0-1062.el7.x86_64.ko "path=/mnt/centos77.mem format=lime"
```

Figure 15: Mounting the NFS and Acquiring a Memory Image (Target Host)

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
Switch to another login session to execute the following command
$ exec 5<>/dev/tcp/127.0.0.1/4444; cat <&5 | ssh -c user@192.168.232.131 'cat > centos77.mem'
```

Figure 16: Commands Executed on the Target Host

(Figure 17). The crash command also requires a kernel with debug symbols, so we install the kernel debugging package (Figure 18). Next, we copy the three files shown in Figure 19 to the same directory on a USB stick. On the target host, we execute crash as shown in Figure 20 to acquire a memory image.

If the kernel version of the target and examination hosts differ, then search for the appropriate version of the kernel-debuginfo package as demonstrated in Figure 4 (yum --showduplicates search). Download and extract the package as shown in FIgure 5 (curl, rpm2cpio, cpio commands), and copy the vmlinux file. We also recommend using a crash version that corresponds to your OS version (for example, crash-7.2.3-18 is provided in CentOS 8.0).

2.7 Analyzing the Memory Image

To analyze a Linux memory image using Volatility[*5], you need a profile corresponding to the Linux kernel version. As noted at the beginning, the procedure for creating a Volatility profile is described in IIR Vol. 32[*1]. Please refer there if you are unsure.

While we were writing this article, Lorenzo Martínez opened a Bitbucket repository that automatically generates and publishes LiME modules and Linux profiles for Volatility[*7]. This repository is updated whenever new Linux kernel packages are released. Note, though, that the OSs that it automatically generates for are CentOS 5, 6, 7, 8 and Ubuntu 14.04 LTS, 16.04 LTS, 18.04 LTS. You can find and download the appropriate LiME module and

```
$ sudo yum install crash crash-devel
$ yumdownloader --source crash
$ rpm -ivh crash-7.2.3-10.el7.src.rpm
$ cd rpmbuild/SPECS
$ rpmbuild -bp crash.spec
$ cd ../BUILD/crash-7.2.3
$ make extensions
```

Figure 17: Installing the crash Source Package and Compiling the Module

```
・/usr/bin/crash
・rpmbuild/BUILD/crash-7.2.3/extensions/snap.so
・/usr/lib/debug/usr/lib/modules/3.10.0-1062.el7.x86_64/vmlinux
```

Figure 19: Files Required for a Memory Dump

```
$ sudo yum install --enablerepo=base-debuginfo kernel-debuginfo-3.10.0-1062.el7.x86_64
```

Figure 18: Installing a Kernel with Debug Symbols

*6 The Volatility Foundation - Open Source Memory Forensics (https://www.volatilityfoundation.org/).

*7 Lorenzo Martínez' tweet (https://twitter.com/lawwait/status/1181469996821700609).

Volatility profile by filtering the repository webpage by kernel version.

Also, it does not support architectures other than x86_64, so if you are using a Linux distribution or architecture other than those covered by the autogenerated files, you will need to prepare the files yourself. The same applies if you are using a customized Linux kernel. If using crash, you will also need your own customized kernel with debug symbols.

2.8 Tips for Acquiring Memory Images

In Linux kernel version 2.4 upward, the tmpfs file system is available. Data on a tmpfs file system resides only in memory and is lost if the host is shut down or restarted, so tmpfs is usually only used for temporary directories and the like with files one is not concerned about losing. It has been observed, however, that attackers take advantage of these properties by using tmpfs as an anti-disk forensics haven for files.

For this reason, Volatility provides the Linux-specific linux_tmpfs command. This command is used to restore files held in a tmpfs. Figure 21 illustrates tmpfs files mounted on the /home/user/tmp directory being restored. The command results in the file called “tmpfs_example.txt” being restored, and evidently it contains the string “hello!!”

If available memory is running low, however, the contents of the tmpfs are swapped out, making it impossible to restore the data in the tmpfs via a memory dump (Figure 22). A potential countermeasure in this case is to temporarily

```
Move to the directory where the files were copied and then execute these
commands.

$ sudo ./crash ./vmlinux
(Then from within the crash command prompt)
extend ./snap.so
snap centos77.mem
```

Figure 20: Acquiring a Memory Image

```
$ hexdump -C ~/vol_output/swapout/tmpfs_example.txt
00000000  00 00 00 00 00 00 00 00                          |........|
00000008
```

Figure 22: Example of a Failure to Restore tmpfs Data due to a Swap-Out

```
$ python2 ./vol.py --profile=LinuxCentOS77x64 -f ~/tmpfs_swapoff.mem linux_tmpfs -L
Volatility Foundation Volatility Framework 2.6.1
1 -> /sys/fs/cgroup
2 -> /run
3 -> /home/user/tmp
4 -> /dev/shm

$ python2 ./vol.py --profile=LinuxCentOS77x64 -f ~/tmpfs_swapoff.mem linux_tmpfs -S 3 -D ~/vol_output/
$ hexdump -C ~/vol_output/tmpfs_example.txt
00000000  68 65 6c 6c 6f 21 21 0a                          |hello!!.|
00000008
```

Figure 21: Restoring and Examining the Contents of Files Stored in a tmpfs

disable the swap. This means forcing data that has been swapped out to be swapped in. Whether or not this can be done, however, depends on memory usage on the target host. If data has been swapped out because of a temporary increase in memory usage, you may be able to subsequently disable the swap if memory has since been freed up. This means a situation like that illustrated in Figure 23, for example, where the value of Swap used is lower than free Mem. Analysis of the memory image acquired after disabling the swap confirms that the tmpfs data can be restored, as shown in Figure 24.

This method, however, does overwrite data in unused areas of memory. Because unused memory areas may still contain useful data, a better way to perform Linux memory forensics is to dump memory once before disabling the swap and once again after.

2.9 Volatility 3

Version 2 of Volatility has been in use for a long time, but during the writing of this article, a public beta version of Volatility 3 was released[*8]. Of course we did not check it with all types of memory images, but in our setup, we were able to analyze Windows RAW memory images and CentOS memory images acquired using LiME. A big change is that the option to specify a profile, which was required in Volatility 2, is no longer present. Instead, Volatility 3 infers OS type and version from the memory image being analyzed and refers to the corresponding symbol table. For example, when a Windows memory image is read in, Volatility 3 automatically downloads the PDB file from Microsoft and analyzes it to construct a symbol table that it can reference. However, you still need to prepare symbol tables for macOS and Linux in advance, as in Volatility 2. You can use the symbol tables provided by the Volatility developers, but the symbol tables for Linux lack information relative to those for Windows and macOS, so in most cases, you'll need to provide the table yourself.

Symbol tables are created using a tool called dwarf2json[*9]. As dwarf2json is written in Go, we first install the golang package and then build dwarf2json. We also install the kernel-debuginfo package because we need a Linux kernel with symbols. Executing dwarf2json with a Linux kernel with

```
$ free
              total        used        free      shared  buff/cache   available
Mem:        1863248       75920      307192         768     1480136     1591860
Swap:       2097148       56776     2040372
$ sudo swapoff -a
$ free
              total        used        free      shared  buff/cache   available
Mem:        1863248      118804      247652        9800     1496792     1539936
Swap:             0           0           0
(After dumping memory, reenable the swap)
$ sudo swapon -a
```

Figure 23: Checking Memory Usage, and Disabling and Enabling the Swap

```
$ hexdump -C ~/vol_output/swapoff/tmpfs_example.txt
00000000  68 65 6c 6c 6f 21 21 0a                           |hello!!.|
00000008
```

Figure 24: tmpfs Data Restored from a Memory Image After Disabling the Swap

```
$ sudo yum install epel-release
$ sudo yum install golang
$ git clone https://github.com/volatilityfoundation/dwarf2json.git
$ cd dwarf2json
$ go build
$ sudo yum install --enablerepo=base-debuginfo kernel-debuginfo-3.10.0-1062.1.2.el7.x86_64
$ ./dwarf2json linux --elf /usr/lib/debug/usr/lib/modules/3.10.0-1062.1.2.el7.x86_64/vmlinux > centos77-3.10.0-1062.1.2.el7.x86_64.json
$ xz -z centos77-3.10.0-1062.1.2.el7.x86_64.json
$ wget https://downloads.volatilityfoundation.org/volatility3/symbols/linux.zip
$ zip ./linux.zip ./centos77-3.10.0-1062.1.2.el7.x86_64.json.xz
```

Figure 25: Building dwarf2json and Generating a Symbol Table

*8 Volatility Labs: Announcing the Volatility 3 Public Beta! (https://volatility-labs.blogspot.com/2019/10/announcing-volatility-3-public-beta.html).

*9 dwarf2json (https://github.com/volatilityfoundation/dwarf2json).

symbols specified will generate a symbol table, so we add this to the file (linux.zip) that contains the symbol table distributed by the developers. See Figure 25 for the specific commands. Copy the generated symbol table to the specified Volatility 3 subdirectory (Figure 26).

Volatility 3 is executed using this command line format: “python3 vol.py -f ＜memory image file＞ ＜plugin＞”. Figure 27 shows the results of analyzing a CentOS 7.7 memory image using the pstree plugin. The format has changed slightly from that of Volatility 2, with the * character now used to denote process nesting. The method for specifying the plugin to be executed has also changed. For the pstree plugin, you specify “linux_pstree” in Volatility 2, but in Volatility 3, you specify “linux.pstree.PsTree”. A list of available plugins can be found by running “python3 vol.py -h”.

While we did get it to work properly, we also identified some bugs. As noted above, OS type and version are inferred from the contents of the memory image specified on the command line, but it can fail to recognize the correct Linux kernel version with some memory images, causing the analysis to fail. And with Windows memory images, sometimes analysis of the downloaded PDB fails, preventing you from advancing to the memory image analysis stage of the process.

The Volatility development team has announced an official Volatility 3 version will be released in August 2020. Support for Volatility 2 will continue for one year after that through August 2021, but with Volatility 3 set to become mainstream ahead, it’s probably a good idea to get accustomed to the new usage and configuration methods before the official release hits.

```
$ cd ..
$ sudo yum install python3
$ git clone https://github.com/volatilityfoundation/volatility3.git
$ pip3 install --user pefile yara-python capstone
$ cp ./dwarf2json/linux.zip ./volatility3/volatility/symbols/
```

Figure 26: Installing Volatility 3 and Copying the Symbol Tables

```
$ cd volatility3
$ python3 vol.py -f ~/centos77.mem linux.pstree.PsTree
Volatility 3 Framework 1.0.0-beta.1
Progress:    23.13               Scanning LimeLayer using RegExScanner
PID     PPID    COMM

1       0       systemd
* 833   1       login
** 1695 833     bash
* 841   1       firewalld
* 843   1       NetworkManager
** 992  843     dhclient
* 558   1       systemd-journal
* 593   1       systemd-udevd
* 818   1       dbus-daemon
* 1171  1       sshd
** 1720 1171    sshd
*** 1724        1720    sshd
**** 1725       1724    bash
***** 1751      1725    sudo
****** 1753     1751    insmod
* 1172  1       tuned
* 821   1       systemd-logind
* 822   1       irqbalance
* 823   1       polkitd
* 1174  1       rsyslogd
* 1397  1       master
** 1402 1397    pickup
** 1405 1397    qmgr
(Subsequent output omitted)
```

Figure 27: Running the pstree Plugin on a Linux Memory Image

Minoru Kobayashi

Forensic Investigator, Office of Emergency Response and Clearinghouse for Security Information, Advanced Security Division, IIJ
Mr. Kobayashi is a member of IIJ-SECT, mainly dealing with digital forensics. He works to improve incident response capabilities and in-house technical capabilities.
He gives lectures and training sessions at security events both in Japan and abroad, including Black Hat, FIRST TC, JSAC, and Security Camp events.

February 2020 Vol.45

Internet Initiative Japan

About Internet Initiative Japan Inc. (IIJ)

IIJ was established in 1992, mainly by a group of engineers who had been involved in research and development activities related to the Internet, under the concept of promoting the widespread use of the Internet in Japan.
IIJ currently operates one of the largest Internet backbones in Japan, manages Internet infrastructures, and provides comprehensive high-quality system environments (including Internet access, systems integration, and outsourcing services, etc.) to high-end business users including the government and other public offices and financial institutions.
In addition, IIJ actively shares knowledge accumulated through service development and Internet backbone operation, and is making efforts to expand the Internet used as a social infrastructure.

Internet Initiative Japan Inc.

Address: Iidabashi Grand Bloom, 2-10-2 Fujimi, Chiyoda-ku,
Tokyo 102-0071, Japan
Email: info@iij.ad.jp URL: https://www.iij.ad.jp/en/