Evaluation and Improvement of IPv6 Protocol Stack by USAGI Project

Yuji Sekiya
Keio University
sekiya@linux-ipv6.org

Hideaki Yoshifuji
The University of Tokyo
yoshfuji@linux-ipv6.org

Mitsuru Kanda
Toshiba Corporation
mk@linux-ipv6.org

Kazunori Miyazawa
Yokogawa Electric Corporation
miyazawa@linux-ipv6.org

Abstract

IPv6 protocol stack has been implemented in Linux kernel since 1996. In spite of the early implementation of IPv6 in the kernel, the stack wasn't maintained for a long time and became out of date. For instance, Linux host couldn't get IPv6 addresses by stateless address autoconfiguration. It was caused by poorly implementation of neighbor discovery protocol. Considering the situation we started USAGI project in October 2000. Our goal is to develop, integrate and provide high quality, RFC compliant, and free IPv6 stack including IPsec and Mobile IPv6. Finally we want to integrate our improvements into the original kernel.

At the beginning we evaluated the original Linux IPv6 stack by TAHI tool. The first evaluation was performed on linux-2.2.15 kernel and the result showed that Linux IPv6 protocol stack wasn't compliant to latest RFCs and didn't have IPsec function which is mandatory for IPv6. As compared with KAME IPv6 protocol stack which passed almost all items of the test, Linux had many problems in kernel. For example, the kernel failed 38 of 58 test items for Neighbor Discovery Protocol and Stateless Address Autoconfiguration failed 55 of 77 items. Then we summarized the results and began to start improving.

Ever since starting the project, we have been continuing to evaluate and improve Linux IPv6 protocol stack. As a result, we have achieved a lot of improvements and released our snapshot code every two weeks and stable code four times. Nowadays the results of evaluation become better and almost all of test item have been passed. Furthermore IPv6 IPsec functions begin to work.

From the experiences, we describe improving methods and evaluation results of USAGI IPv6 protocol stack in this paper. Lastly we describe our future development and merge plans.

1 Introduction

Establishment of IPv6, as a next-generation internet protocol to IPv4, has started since the beginning of the 1990's. The aspect of IPv6 is on providing the solution to the protocol scalability, the greatest problem IPv4 was facing as the Internet grew larger. In detail, IPv6 differ from IPv4 in following ways.

- 128bit address space.

- Forbidding of packet fragmentation in intermediate routers.
- Flexible feature extension using extension headers.
- Supporting security features by default.
- Supporting Plug & Play features by default.

Currently, IPv6 is at the final phase of standardization. Fundamental specifications are almost fixed and commercial products which supports IPv6 has started to show up in the market. International leased lines for IPv6 are out as well. IPv6 has expanded the existing Internet by providing solutions to protocol scalability and beginning to grow as a standard for connecting everything, not just existing computers.

Considering above circumstances, USAGI Project was lunched in October, 2000. USAGI Project is a project which aims to provide improved IPv6 stack on Linux. There are similar organization called KAME, which provides IPv6 stack on BSD Operating systems such as FreeBSD, NetBSD, OpenBSD, and BSD/OS. However, KAME Project does not target their development on Linux. It is important to provide high-quality IPv6 stack on Linux, which is one of the most popular free open-source operating systems in the world, for IPv6 to propagate.

2 Linux IPv6 Implementation

Linux kernel has IPv6 protocol stack by default. However, this IPv6 protocol stack has several problems. In this section, we describe basic IPv6 functions which are needed for IPv6 host and router. Additionally, evaluations on functions mentioned below are done using existing IPv6 protocol stack.

Following utilization patterns are assumed for using Linux, as an IPv6 host, to connect to IPv6 networks.

1. IPv6 host
2. IPv6 gateway inside the house
3. IPv6 mobile host

There, problems on Linux IPv6 protocol stack are described and evaluated in details following the categories mentioned above. Then following evaluations are performed on Linux kernel 2.2.15, 2.2.20 and 2.4.18.

2.1 Linux as an IPv6 Host

Following features are required for using Linux as an IPv6 host.

- IPv6 address autoconfiguration
- Reachability and unreachability detection of neighbor hosts and routers
- IPv6 TCP/UDP socket communication

"IPv6 address autoconfiguration" is a indispensable feature of IPv6 to enable plug & play function. Additionally, "Reachability and unreachability detection of neighbor hosts and routers" are required for a IPv6 host to detect and switch default routers quickly. Furthermore, "IPv6 TCP/UDP socket communication" is a feature required for using IPv6 applications on Linux.

From these perspectives, evaluation on features which Linux IPv6 protocol stack on kernel 2.2.15, 2.2.20 and 2.4.18. The 2.2.15 kernel was released before USAGI Project initialization. The evaluations were done using tools provided by TAHI Project[9].

IPv6 address autoconfiguration First of all, evaluation on IPv6 address autoconfiguration feature is described. IPv6 stateless address autoconfiguration is a function which is defined in RFC2462[10]. It is a function which configures IPv6 address and default router automatically when receiving router advertisements from IPv6 routers. This function is needed for IPv6 Plug & Play feature. The evaluation result of 2.2.15 kernel on this function is shown in Table 2, result of 2.2.20 kernel is shown in Table 3 and result of 2.4.18 kernel is shown in Table 4.

In result of 2.2.15 kernel, there were 30 items that failed and 22 items with warnings of the 54 items in the test. Only 1 of the 54 items passed. There are several causes for this. First, the Duplicate Address Detection (DAD) described in RFC2462[10], which detects duplicated address, may not work correctly. This can be seen from test numbers 2, 20, 21, 23 showing failure in Table 2.

Moreover, the Router Advertisement (RA), which auto-configures the address and the default route, can be one of the causes for the messages may not have processed correctly. The numbers 30, 31 and 40 show this in Table 2.

In result of 2.2.20 kernel, there were 25 failed items, 15 warned items and 13 passed items. The number of passed items are increased from 1 to 13. However, many of DAD function were failed and didn't work correctly.

In result of 2.4.18 kernel, many functions were improved. There were 42 passed items, 10 failed items and 1 warned item.

As compared with 2.2 kernel, DAD functions were improved on Linux 2.4 kernel. However, there was a lack in the number of error processes when receiving irregular messages, which caused the test to failed for abnormal behavior.

Figure 6 summarizes the result of comparing the three kernel results.

Reachability and unreachability detection Second, evaluation on Neighbor Discovery Protocol is mentioned. Neighbor Discovery Protocol is a function defined in RFC2461[7]. It is a function which detects appearance and disappearance of hosts and routers. This function is needed for IPv6 hosts to communicate with neighbor hosts. The evaluation is also performed on Linux kernel 2.2.15, 2.2.20 and 2.4.18. The result on this function is shown in Table 5, Table 6, and Table 7.

Of the 58 items in the test, there were 36 items that failed, 2 items with warnings and 20 items that passed in result of kernel 2.2.15. From failures in test numbers 15, 16, 17, 19, 20, 22, 23 and 24, we see that the state transition in Neighbor Discovery Protocol did not follow the specifications defined in RFC2461.

Same as result of 2.2.15 kernel, there were many failed items in result of 2.2.20 kernel. Regarding NDP functions there wan no difference between 2.2.15 and 2.2.20 kernels.

In result of 2.4.18 kernel, the number of failed items was decreased to 27 items and the number of succeeded items was increased to 29. As a result, there were some improvements between 2.2.20 and 2.4.18 kernels. However, it wasn't good result for working correct IPv6 host.

Figure 7 summarizes the comparison of the three kernel results.

IPv6 TCP/UDP socket communication Last, to enable IPv6 applications, it is necessary to have communication features for both

TCP and UDP in IPv6. GNU libc(glibc)[5] and IPv6 protocol stack in Linux has both TCP and UDP, and applications using Application Program Interface may access both TCP and UDP socket in IPv6.

However, GNU libc and IPv6 protocol stack in Linux made in the beginning of year 2000 before the USAGI Project was founded, were implemented as IPv6 socket API based on RFC2133[3]. The latest IPv6 socket API at that time was RFC2553[4] revised from RFC2133, thus both GNU libc and IPv6 protocol stack in Linux were implemented not based on the latest specifications.

From the evaluation results above, there are number of changes required to use Linux as IPv6 host. Among all, neighbor discovery protocol and IPv6 address autoconfiguration are base features in IPv6 communication. If these features does not function correctly, it will not function as IPV6 host.

Moreover, from the security and error process point of view, protocol stack should be build not cause effects on the function by discarding illegal packets. From this point of view, IPv6 protocol stack in Linux is has not reached the level to use as a stable IPv6 host.

2.2 Linux as an IPv6 Gateway Inside the House

Next, we evaluate Linux IPv6 protocol stack on the features necessary when Linux is used as IPv6 router in homes.

- IPv6 packet forwarding according to routing table
- IPv6 over IPv4 tunneling

The "IPv6 packet forwarding according to routing table" is a feature to forward packets to the next hop according to the routing information in the routing table when IPv6 packets are received. This in a required feature in a IPv6 router. The "IPv6 over IPv4 tunneling" is a technology to structure IPv6 Internet by establishing a virtual link over the IPv4 Internet.

We evaluated the Linux IPv6 protocol stack using Linux kernel 2.2.15 on the above 2 features. We tested if IPv6 packets are correctly forwarded using the topology shown in Figure 1.

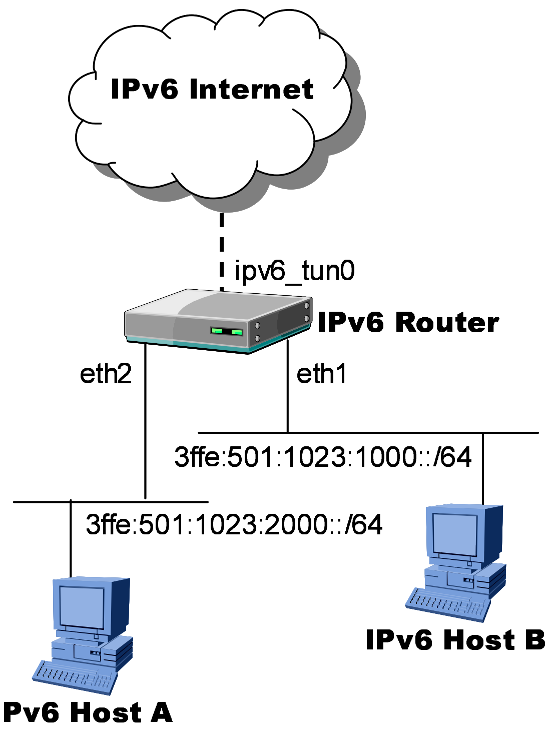

Figure 1: Topology for Routing Evaluation

Also, the routing table on the IPv6 router is shown in the Figure 1. IPv6 hosts A and B receives RA message from IPv6 router and auto-configures IPv6 address and the default route. Additionally, this IPv6 router and the IPv6 router on the other side are connected using IPv6 over IPv4 tunnel. It is assumed that IPv6 router on the other side has correct routing. IPv6 router on the other side uses

Kernel IPv6 routing table		
Destination	Next Hop	Iface
::1/128	::	lo
3ffe:501:1023::/48	::	ipv6_tun0
3ffe:501:1023:1000::1/128	::	lo
3ffe:501:1023:1000::/64	::	eth1
3ffe:501:1023:2000::1/128	::	lo
3ffe:501:1023:2000::/64	::	eth2
fe80::/64	::	eth0
fe80::/64	::	eth1
fe80::/64	::	eth2
fe80::/64	::	ipv6_tun0
ff00::/8	::	eth0
ff00::/8	::	eth1
ff00::/8	::	eth2
ff00::/8	::	ipv6_tun0
::/0	fe80::290:27ff:fe3a:d8	ipv6_tun0

Table 1: IPv6 Routing Table

FreeBSD with KAME snap kit.

Following four kinds of tests are done using above network environment.

TEST #1 Communication from IPv6 router on the other side to IPv6 hosts A and B.

TEST #2 Communication between IPv6 hosts A and B.

TEST #3 Communication between IPv6 router and IPv6 router on the other side.

TEST #4 Communication from IPv6 router to any IPv6 hosts on the Internet.

As a result, test 1 was proven to be successful. Test2, passing through IPv6 router, host A and B was able to communicate each other. On test 3, with the use of IPv6 tunnel, communication using IPv6 was proven to be possible.

However, test 4 failed. The cause was IPv6 router discarding the packet sent from A to fe80::290:27ff:fe3a:d8, which is the next hop of the default route(::/0) without forwarding.

This results from the IPv6 protocol stack specification on Linux. From the point of routing control mechanism, Linux IPv6 protocol stack differs from IPv4 protocol stack with no similarity. On IPv4 protocol stack, routing table is kept in the hash table[1], on the other hand, IPv6 routing table is kept using radix tree[8].

In routing table using radix tree, the top of the tree is the host which possesses the information regarding default route. However, as shown in Figure 2, Linux IPv6 protocol stack has a radix tree with fixed node information on top and it points to ipv6_null_entry. Therefore, when default route is added, the information is attached next to the rt6_info structure which contains ipv6_null_entry. This causes default route not to be referred.

In conclusion, "IPv6 packet forwarding according to routing table" works correctly except for the default route part. Also, "IPv6 over IPv4 tunneling" is proven to work correctly as well. For IPv6 routers inside the houses, it is critical that default route is not functioning. It is hard to say that IPv6 router inside the house possess full routes and it is recom-

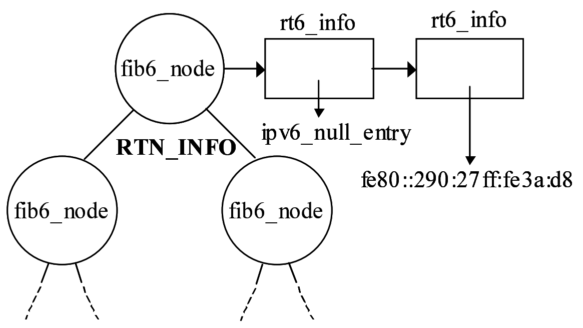

Figure 2: Linux IPv6 Routing Table Structure

mended that routes outside the house is held as default route. Therefore, it is hard to use Linux as a IPv6 gateway inside the house without the change in protocol specification.

2.3 Summary of Linux IPv6 Protocol Stack

From the evaluations mentioned above, Linux IPv6 protocol stack is not suitable in its features as an IPv6 node. It is hard to believe Linux IPv6 protocol to function as an IPv6 host and router using the current Linux IPv6 protocol stack.

3 Activities of USAGI Project

As mentioned in Section 2, IPv6 protocol stack in Linux carry several fatal problems. Thus, USAGI Project was launched, as mentioned in Section 1, to improve the IPv6 protocol stack in Linux. Improvement made by USAGI Project is mentioned in this section.

3.1 Improvement by USAGI Project

Improvements by USAGI Project are mentioned below based on the evaluation and analysis in Section 2.

- Reinforcing illegal NDP message check
- Improving control timer for NDP state
- Following Latest API
- Improving IPv6 routing table structure
- Imprementing IPsec for IPv6

Each improvements is mentioned in detail.

Reinforce illegal NDP message check First of all, reinforced illegal message check for Neighbor Advertisement(NA), Neighbor Solicitation(NS), Router Advertisement(RA), and Router Solicitation(RS) as defined in NDP specifications. The checks are mentioned below in detail.

- Make an unified management function for above messages
- Discarding messages with Hop Limit other than 255
- Discarding NA messages with solicited flag and multicast address as source address
- Selecting proper source address when sending DAD messages
- Improving router renewal algorithm for RA messages with link-local unicast address as source address
- Discarding RA messages with source address other than link-local unicast address
- Discarding NA messages with destination address as multicast address
- Improving the number to send RS message

These simple improvement prevented abnormal process.

Improving control timer for NDP state As mentioned before, NDP is a feature to control status of neighbor nodes. NDP checks on regular intervals if neighbor nodes are reachable or not. The specification require to allocate NDP entry in memory for every node in neighbor, and control status for each neighbor node.

However, the existing Linux IPv6 protocol stack checks reachability of neighbor nodes with a single kernel timer, as shown in Figure 3. Consequently, reachability were checked in constant intervals, regardless of the status for each node.

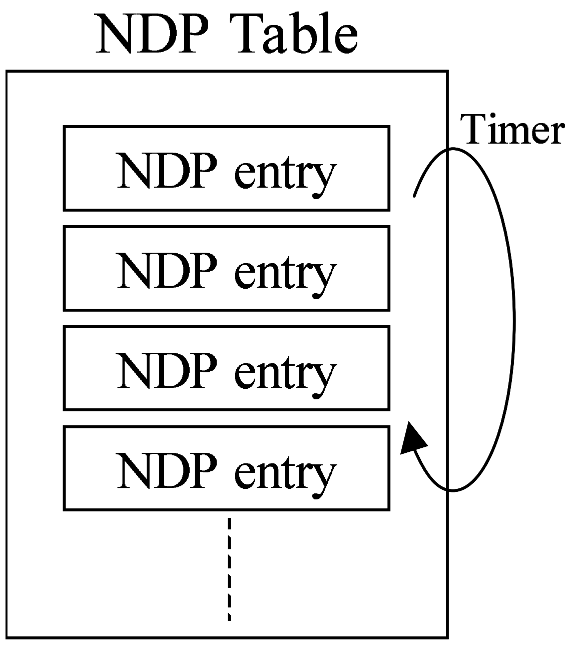

Figure 3: Linux NDP Table

Therefore, USAGI Project improved this kernel timer to check each NDP entry independently as shown in Figure 4. Thus, it is possible to enable and disable timer separately for each NDP entry, and prevent check made to unnecessary NDP entries. Moreover, it is possible to exchange messages correspondent to the status of each NDP entry as defined in the NDP specifications.

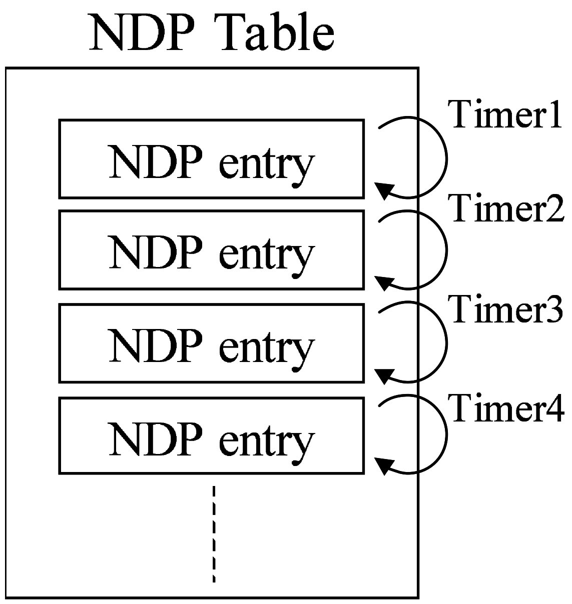

Figure 4: USAGI NDP Table

Following Latest API Linux IPv6 protocol stack and GNU libc, which corresponds to RFC2133, was updated to comply with RFC2533. Change in kernel is adding member sin6_scope_id to sockaddr_in6 structure.

Improving IPv6 routing table structure Problem on recognizing default route is fixed. As shown in Figure 2, modification has made it possible to change pointer from fib6_node structure to rt6_info structure and also made it to adapt routing table lookup functions to new tree structure. With all the improvements, Linux IPv6 protocol stack can recognize IPv6 default route and forward IPv6 packet properly.

Implementing IPsec for IPv6 We implement IPsec for IPv6 in USAGI kernel and our IPsec stack was derived from IABG IPsec[6]. We have improved and changed based on IABG's IPsec and currently our IPsec stack is independent from IABG's IPsec.

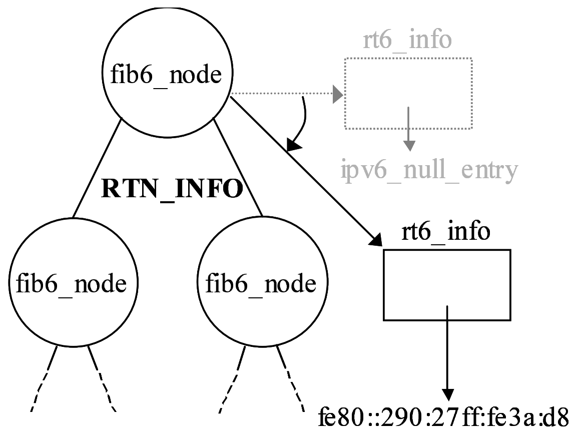

Figure 5: USAGI IPv6 Routing Table Structure

Our stack supports AH/ESP transport mode and manual keying, key negotiation by IKE which is ported from FreeS/WAN[2]. We have already attended some interoperability test events and our stack could communicate with other IPsec implementations. Because our implementation is in still progress, we will continue to work with IPsec for IPv6.

3.2 Evaluation of USAGI Improvement

The same TAHI IPv6 conformance test was done on USAGI snapshot release at the point of Feb. 18th, 2002, which contains five improvements mentioned earlier. The kernel is based on Linux kernel 2.4.17.

The test result of IPv6 address autoconfiguration on USAGI kernel is shown in Table 8.

The result showed that it passed almost all tests regarding IPv6 stateless address autoconfiguration. There were 52 passed items and only 1 warned item.

Then the test result of NDP on USAGI kernel is shown in Table 9.

It reduced the number of failed items. There were 45 succeeded items, 3 warned items and 10 failed items.

Summary of the IPv6 conformance tests on Linux kernels and USAGI IPv6 kernel regarding IPv6 stateless address autoconfiguration and neighbor discovery is shown in Figure 8 and Figure 9.

4 Availability

The outcome of the USAGI Project can be obtained from http://www.linux-ipv6.org/. We release snapshot and stable USAGI kit. Stable USAGI kits are released several times a year and snapshot USAGI kits are released once in two weeks. We have already released stable kit four times. The first was on Nov. 1st, 2000 and the second was on Feb. 5th, 2001. The third was on Jan. 1st, 2002 and The forth, it was a bug fix release of third stable release, was released on Apr. 8th, 2002.

5 Summary

We evaluated Linux IPv6 protocol stack and analyzed the problems in detail. As a result, we were able to pointed out the points which needs improvements. The results enabled to pass many items listed in IPv6 conformance test, and use the features required for IPv6 host.

From the evaluation results, it is obviously that the quality of Linux IPv6 protocol stack has been improved by USAGI Project. We are now working for making patches in order to contribute our improvements to original Linux kernels. However, we have to divide our improvements into small patches which are splitted by each improvement. It takes somewhat span to complete making patches and contribute them.

6 Future Works

Finally, we list our future plans in USAGI Project in this section.

The analysis and improvements mentioned in this paper, were those only in IPv6 protocol specification in IPv6 protocol stack.

Concerning IPv6 protocol specifications, we plan to continue activity especially focused on passing TAHI IPv6 conformance Test, implementing IPv6 features not yet implemented, IPsec Protocol, and IPv6 performance tuning.

For our future plan, we have the below developing items.

- IPsec tunnel mode
- Generic tunnel device for IPv4 and IPv6
- Introduce scope semantics
- DHCPv6
- Prefix Delegation Protocol
- IPv4/IPv6 Translator

References

[1] Jon Crowcroft and Iain Phillips. *TCP/IP and Linux Protocol Implementation: Systems Code for the Linux Internet*. WILEY Publishers, October 2001.

[2] FreeS/WAN Project. Linux FreeS/WAN Project. `http://www.freeswan.org/`.

[3] R. Gilligan, S. Thomson, J. Bound, and W. Stevens. Basic Socket Interface Extensions for IPv6. RFC2133, April 1997.

[4] R. Gilligan, S. Thomson, J. Bound, and W. Stevens. Basic Socket Interface Extensions for IPv6. RFC2553, March 1999.

[5] GNU Project. GNU C library. `http://www.gnu.org/software/libc/libc.html`.

[6] IABG. IPv6 at IABG. `http://www.ipv6.iabg.de/`.

[7] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC2461, December 1998.

[8] Keith Sklower. A tree-based packet routing table for berkeley unix. In *USENIX Winter*, pages 93–104, 1991.

[9] TAHI Project. Test and Verification for IPv6. `http://www.tahi.org/`.

[10] S. Thomson and T. Narten. IPv6 Stateless Address Autoconfiguration. RFC2462, December 1998.

7 Tables & Figures

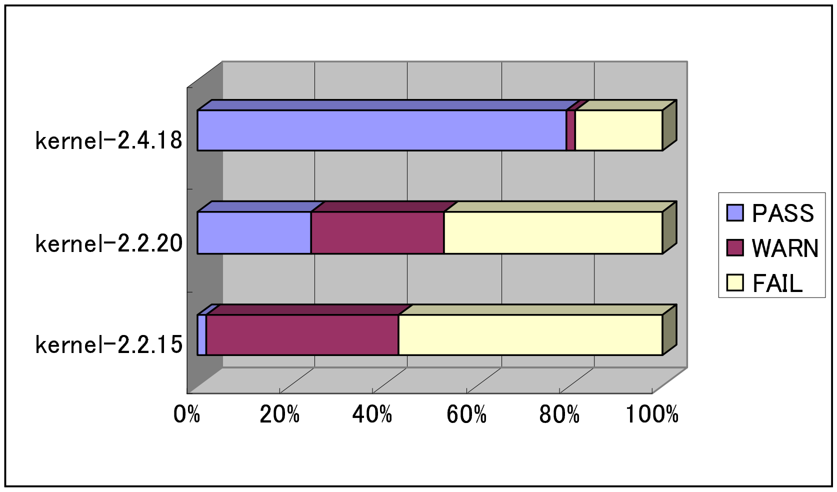

Figure 6: Result of IPv6 Address Autoconfiguration Test on Linux Kernel

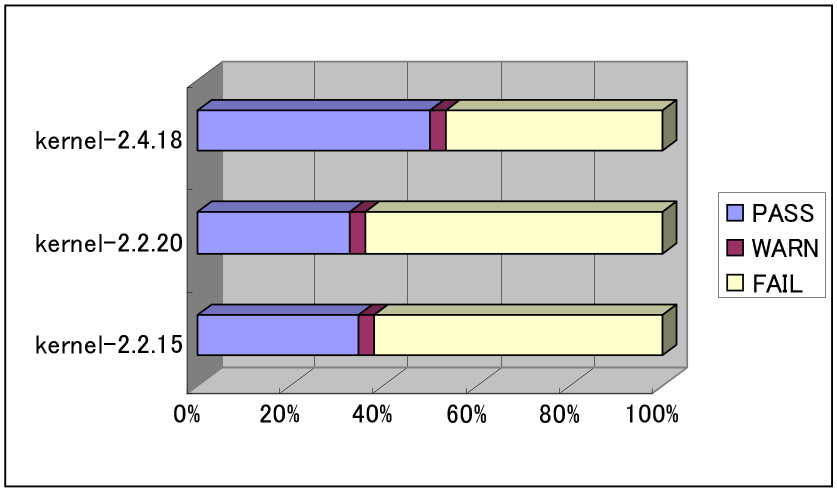

Figure 7: Result of IPv6 NDP Test on Linux Kernel

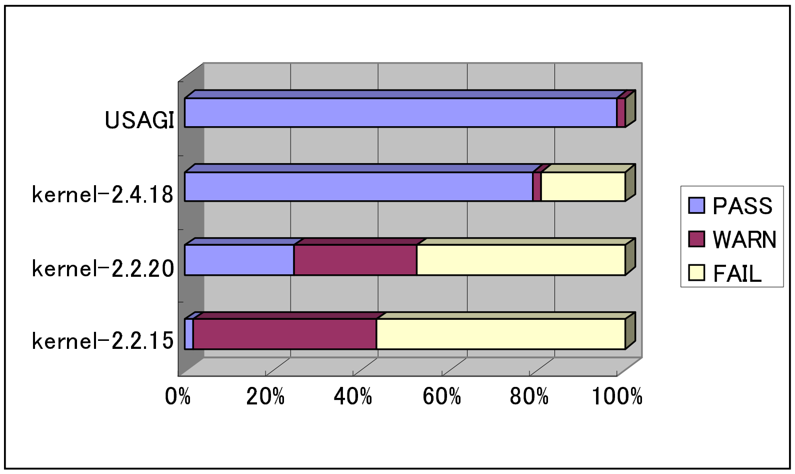

Figure 8: Result of IPv6 Address Autoconfiguration Test on USAGI

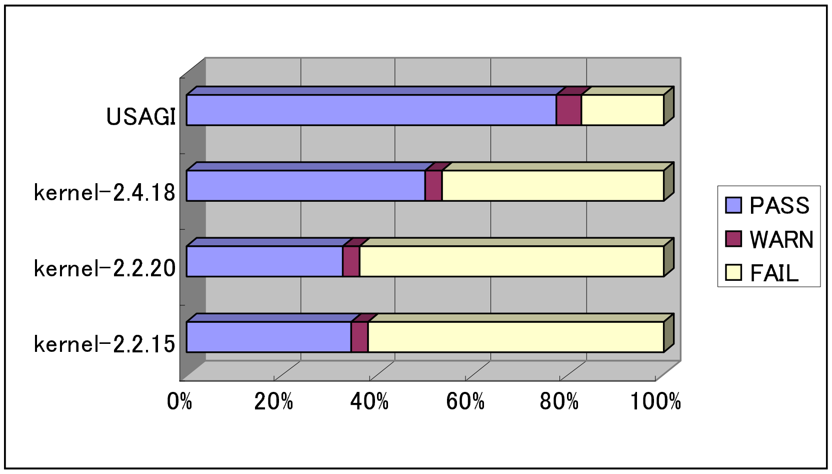

Figure 9: Result of IPv6 NDP Test on USAGI

Table 2: IPv6 Conformance Test For Stateless Address Configuration on Linux kernel 2.2.15

Test No.	Title	Result
1	DAD is performed on NUT by Stateless Link-local address autoconfiguration	WARN
2	DAD Success when NUT received no packet on Stateless Link-local address autoconfiguration	FAIL
3	DAD Fail when NUT received Valid NS in random delaying phase on Stateless Link-local address autoconfiguration	PASS
4	DAD Fail when NUT received Valid NS (dst MAC addr != MAC addr of NUT) on Stateless Link-local address autoconfiguration	WARN
5	DAD Fail when NUT received Valid NS (dst MAC addr == MAC addr of NUT) on Stateless Link-local address autoconfiguration	WARN
6	DAD Fail when NUT received Surprise NS (Prefix Option) on Stateless Link-local address autoconfiguration (Surprise test)	WARN
7	DAD Fail when NUT received Valid NA (dst MAC addr != MAC addr of NUT) on Stateless Link-local address autoconfiguration	WARN
8	DAD Fail when NUT received Valid NA (dst MAC addr == MAC addr of NUT) on Stateless Link-local address autoconfiguration	WARN
9	DAD Fail when NUT received NA (No TLL option) on Stateless Link-local address autoconfiguration	WARN
10	DAD Fail when NUT received NA (dst addr == solicited node multicast) on Stateless Link-local address autoconfiguration	WARN
11	DAD Fail when NUT received Surprise NA (Many Options) on Stateless Link-local address autoconfiguration (Surprise test)	WARN
12	DAD Success when NUT received Invalid NS (Dst addr is Allnodes) on Stateless Link-local address autoconfiguration	FAIL
13	DAD Success when NUT received Invalid NS (Dst addr is Tentative) on Stateless Link-local address autoconfiguration	FAIL
14	DAD Success when NUT received Invalid NS (Hoplimit != 255) on Stateless Link-local address autoconfiguration	FAIL
15	DAD Success when NUT received Invalid NS (Include SLL opt) on Stateless Link-local address autoconfiguration	FAIL
16	DAD Success when NUT received NS (Src addr is Unicast) on Stateless Link-local address autoconfiguration	FAIL
17	DAD Success when NUT received Invalid NA (Hoplimit != 255) on Stateless Link-local address autoconfiguration	FAIL
18	DAD Success when NUT received Invalid NA (S flag == 1) on Stateless Link-local address autoconfiguration	FAIL
19	DAD Success when NUT received NA (Dst addr is unicast) on Stateless Link-local address autoconfiguration	FAIL
20	DAD is performed on NUT by Manual Link-local address configuration	FAIL
21	DAD Success when NUT received no packet on Manual Link-local address configuration	FAIL
22	DAD is performed on NUT by Manual Global address configuration	WARN
23	DAD Success when NUT received no packet on Manual Global address configuration	FAIL
24	DAD Fail when NUT received Valid NS (dst MAC addr == MAC addr of NUT) on Manual Global address configuration	WARN
25	DAD Fail when NUT received Valid NA (dst MAC addr == MAC addr of NUT) on	

Table continues on next page. . .

Table 2 continued...

Test No.	Title	Result
	Manual Global address configuration	WARN
26	DAD Success when NUT received Invalid NS (Dst addr is Allnodes) on Manual Global address configuration	FAIL
27	DAD Success when NUT received Invalid NS (Dst addr is Tentative) on Manual Global address configuration	FAIL
28	DAD is performed on NUT by Stateless Global address autoconfiguration	WARN
29	DAD is performed on NUT by Stateless Global address autoconfiguration after DAD Failed for Link-local address autoconfiguration	WARN
30	ADDRCONF Success when NUT received Valid RA (Global address)	FAIL
31	ADDRCONF Success when NUT received Valid RA (Site-local address)	FAIL
32	ADDRCONF Success when NUT received Surprise RA (Many link-layer options) (Surprise test)	FAIL
33	NUT ignores prefixopt if PreferredLifeTime > ValidLifeTime	WARN
34	NUT ignores prefixopt if Prefixlen > 64 (interface ID len is 64)	WARN
35	NUT ignores prefixopt if Prefixlen < 64 (interface ID len is 64)	WARN
36	NUT ignores prefixopt if A flag is 0	WARN
37	NUT ignores prefixopt if prefix is Link-local	WARN
38	NUT ignores prefixopt if Prefixlen > 128	WARN
39	NUT ignores prefixopt if ValidLifeTime is 0 (unknown prefix)	WARN
40	NUT ignores prefixopt if ValidLifeTime is 0 (known prefix but without IPSEC authentication)	FAIL
41	NUT ignores prefixopt if prefix is Global Multicast (Surprise test)	WARN
42	Probe PrefixOptions processing order of same prefixes in one RA (Surprise test)	FAIL
43	Check if ValidLifetime is reset on NUT by RA with same prefix (before expiry, greater VLT)	FAIL
44	Check if ValidLifetime is NOT reset on NUT by RA with same prefix (before expiry, same VLT)	FAIL
45	Check if ValidLifetime is reset on NUT by RA with same prefix (after expiry, same VLT)	FAIL
46	Check if ValidLifetime is NOT reset on NUT by RA with same prefix (before expiry, less VLT)	FAIL
47	Check if ValidLifetime is reset on NUT by RA with same prefix (after expiry, less VLT)	FAIL
48	Packet receiving and Global address lifetime expiry (valid preferred, valid deprecated, invalid)	FAIL
49	Packet receiving and Site-local address lifetime expiry (valid preferred, valid deprecated, invalid)	FAIL
50	Source address selection and address lifetime expiry (valid deprecated VS valid preferred)	FAIL
51	Source address selection and address lifetime expiry (valid deprecated VS valid deprecated)	FAIL
52	Source address selection and address lifetime expiry (invalid VS valid deprecated)	FAIL
53	Source address selection and address lifetime expiry (invalid VS invalid)	FAIL

This Report was generated by TAHI IPv6 Conformance Test Suite

Table 3: IPv6 Conformance Test For Stateless Address Configuration on Linux kernel 2.2.20

Test No.	Title	Result
1	DAD is performed on NUT by Stateless Link-local address autoconfiguration	PASS
2	DAD Success when NUT received no packet on Stateless Link-local address autoconfiguration	PASS
3	DAD Fail when NUT received Valid NS in random delaying phase on Stateless Link-local address autoconfiguration	PASS
4	DAD Fail when NUT received Valid NS (dst MAC addr != MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
5	DAD Fail when NUT received Valid NS (dst MAC addr == MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
6	DAD Fail when NUT received Surprise NS (Prefix Option) on Stateless Link-local address autoconfiguration (Surprise test)	PASS
7	DAD Fail when NUT received Valid NA (dst MAC addr != MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
8	DAD Fail when NUT received Valid NA (dst MAC addr == MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
9	DAD Fail when NUT received NA (No TLL option) on Stateless Link-local address autoconfiguration	PASS
10	DAD Fail when NUT received NA (dst addr == solicited node multicast) on Stateless Link-local address autoconfiguration	PASS
11	DAD Fail when NUT received Surprise NA (Many Options) on Stateless Link-local address autoconfiguration (Surprise test)	PASS
12	DAD Success when NUT received Invalid NS (Dst addr is Allnodes) on Stateless Link-local address autoconfiguration	FAIL
13	DAD Success when NUT received Invalid NS (Dst addr is Tentative) on Stateless Link-local address autoconfiguration	FAIL
14	DAD Success when NUT received Invalid NS (Hoplimit != 255) on Stateless Link-local address autoconfiguration	FAIL
15	DAD Success when NUT received Invalid NS (Include SLL opt) on Stateless Link-local address autoconfiguration	FAIL
16	DAD Success when NUT received NS (Src addr is Unicast) on Stateless Link-local address autoconfiguration	FAIL
17	DAD Success when NUT received Invalid NA (Hoplimit != 255) on Stateless Link-local address autoconfiguration	FAIL
18	DAD Success when NUT received Invalid NA (S flag == 1) on Stateless Link-local address autoconfiguration	FAIL
19	DAD Success when NUT received NA (Dst addr is unicast) on Stateless Link-local address autoconfiguration	FAIL
20	DAD is performed on NUT by Manual Link-local address configuration	WARN
21	DAD Success when NUT received no packet on Manual Link-local address configuration	FAIL
22	DAD is performed on NUT by Manual Global address configuration	WARN
23	DAD Success when NUT received no packet on Manual Global address configuration	FAIL
24	DAD Fail when NUT received Valid NS (dst MAC addr == MAC addr of NUT) on Manual Global address configuration	WARN
25	DAD Fail when NUT received Valid NA (dst MAC addr == MAC addr of NUT) on	

Table continues on next page...

Table 3 continued…

Test No.	Title	Result
	Manual Global address configuration	WARN
26	DAD Success when NUT received Invalid NS (Dst addr is Allnodes) on Manual Global address configuration	FAIL
27	DAD Success when NUT received Invalid NS (Dst addr is Tentative) on Manual Global address configuration	FAIL
28	DAD is performed on NUT by Stateless Global address autoconfiguration	WARN
29	DAD is performed on NUT by Stateless Global address autoconfiguration after DAD Failed for Link-local address autoconfiguration	PASS
30	ADDRCONF Success when NUT received Valid RA (Global address)	FAIL
31	ADDRCONF Success when NUT received Valid RA (Site-local address)	FAIL
32	ADDRCONF Success when NUT received Surprise RA (Many link-layer options) (Surprise test)	FAIL
33	NUT ignores prefixopt if PreferredLifeTime > ValidLifeTime	WARN
34	NUT ignores prefixopt if Prefixlen > 64 (interface ID len is 64)	WARN
35	NUT ignores prefixopt if Prefixlen < 64 (interface ID len is 64)	WARN
36	NUT ignores prefixopt if A flag is 0	WARN
37	NUT ignores prefixopt if prefix is Link-local	PASS
38	NUT ignores prefixopt if Prefixlen > 128	WARN
39	NUT ignores prefixopt if ValidLifeTime is 0 (unknown prefix)	WARN
40	NUT ignores prefixopt if ValidLifeTime is 0 (known prefix but without IPSEC authentication)	FAIL
41	NUT ignores prefixopt if prefix is Global Multicast (Surprise test)	WARN
42	Probe PrefixOptions processing order of same prefixes in one RA (Surprise test)	FAIL
43	Check if ValidLifetime is reset on NUT by RA with same prefix (before expiry, greater VLT)	FAIL
44	Check if ValidLifetime is NOT reset on NUT by RA with same prefix (before expiry, same VLT)	FAIL
45	Check if ValidLifetime is reset on NUT by RA with same prefix (after expiry, same VLT)	FAIL
46	Check if ValidLifetime is NOT reset on NUT by RA with same prefix (before expiry, less VLT)	FAIL
47	Check if ValidLifetime is reset on NUT by RA with same prefix (after expiry, less VLT)	FAIL
48	Packet receiving and Global address lifetime expiry (valid preferred, valid deprecated, invalid)	FAIL
49	Packet receiving and Site-local address lifetime expiry (valid preferred, valid deprecated, invalid)	FAIL
50	Source address selection and address lifetime expiry (valid deprecated VS valid preferred)	WARN
51	Source address selection and address lifetime expiry (valid deprecated VS valid deprecated)	WARN
52	Source address selection and address lifetime expiry (invalid VS valid deprecated)	WARN
53	Source address selection and address lifetime expiry (invalid VS invalid)	FAIL

This Report was generated by TAHI IPv6 Conformance Test Suite

Table 4: IPv6 Conformance Test For Stateless Address Configuration on Linux kernel 2.4.18

Test No.	Title	Result
1	DAD is performed on NUT by Stateless Link-local address autoconfiguration	PASS
2	DAD Success when NUT received no packet on Stateless Link-local address autoconfiguration	PASS
3	DAD Fail when NUT received Valid NS in random delaying phase on Stateless Link-local address autoconfiguration	PASS
4	DAD Fail when NUT received Valid NS (dst MAC addr != MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
5	DAD Fail when NUT received Valid NS (dst MAC addr == MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
6	DAD Fail when NUT received Surprise NS (Prefix Option) on Stateless Link-local address autoconfiguration (Surprise test)	PASS
7	DAD Fail when NUT received Valid NA (dst MAC addr != MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
8	DAD Fail when NUT received Valid NA (dst MAC addr == MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
9	DAD Fail when NUT received NA (No TLL option) on Stateless Link-local address autoconfiguration	PASS
10	DAD Fail when NUT received NA (dst addr == solicited node multicast) on Stateless Link-local address autoconfiguration	PASS
11	DAD Fail when NUT received Surprise NA (Many Options) on Stateless Link-local address autoconfiguration (Surprise test)	PASS
12	DAD Success when NUT received Invalid NS (Dst addr is Allnodes) on Stateless Link-local address autoconfiguration	FAIL
13	DAD Success when NUT received Invalid NS (Dst addr is Tentative) on Stateless Link-local address autoconfiguration	PASS
14	DAD Success when NUT received Invalid NS (Hoplimit != 255) on Stateless Link-local address autoconfiguration	PASS
15	DAD Success when NUT received Invalid NS (Include SLL opt) on Stateless Link-local address autoconfiguration	FAIL
16	DAD Success when NUT received NS (Src addr is Unicast) on Stateless Link-local address autoconfiguration	PASS
17	DAD Success when NUT received Invalid NA (Hoplimit != 255) on Stateless Link-local address autoconfiguration	PASS
18	DAD Success when NUT received Invalid NA (S flag == 1) on Stateless Link-local address autoconfiguration	PASS
19	DAD Success when NUT received NA (Dst addr is unicast) on Stateless Link-local address autoconfiguration	PASS
20	DAD is performed on NUT by Manual Link-local address configuration	PASS
21	DAD Success when NUT received no packet on Manual Link-local address configuration	PASS
22	DAD is performed on NUT by Manual Global address configuration	PASS
23	DAD Success when NUT received no packet on Manual Global address configuration	PASS
24	DAD Fail when NUT received Valid NS (dst MAC addr == MAC addr of NUT) on Manual Global address configuration	PASS
25	DAD Fail when NUT received Valid NA (dst MAC addr == MAC addr of NUT) on	

Table continues on next page…

Table 4 continued. . .

Test No.	Title	Result
	Manual Global address configuration	PASS
26	DAD Success when NUT received Invalid NS (Dst addr is Allnodes) on Manual Global address configuration	FAIL
27	DAD Success when NUT received Invalid NS (Dst addr is Tentative) on Manual Global address configuration	PASS
28	DAD is performed on NUT by Stateless Global address autoconfiguration	PASS
29	DAD is performed on NUT by Stateless Global address autoconfiguration after DAD Failed for Link-local address autoconfiguration	PASS
30	ADDRCONF Success when NUT received Valid RA (Global address)	PASS
31	ADDRCONF Success when NUT received Valid RA (Site-local address)	PASS
32	ADDRCONF Success when NUT received Surprise RA (Many link-layer options) (Surprise test)	PASS
33	NUT ignores prefixopt if PreferredLifeTime > ValidLifeTime	PASS
34	NUT ignores prefixopt if Prefixlen > 64 (interface ID len is 64)	PASS
35	NUT ignores prefixopt if Prefixlen < 64 (interface ID len is 64)	PASS
36	NUT ignores prefixopt if A flag is 0	PASS
37	NUT ignores prefixopt if prefix is Link-local	PASS
38	NUT ignores prefixopt if Prefixlen > 128	PASS
39	NUT ignores prefixopt if ValidLifeTime is 0 (unknown prefix)	PASS
40	NUT ignores prefixopt if ValidLifeTime is 0 (known prefix but without IPSEC authentication)	PASS
41	NUT ignores prefixopt if prefix is Global Multicast (Surprise test)	PASS
42	Probe PrefixOptions processing order of same prefixes in one RA (Surprise test)	WARN
43	Check if ValidLifetime is reset on NUT by RA with same prefix (before expiry, greater VLT)	FAIL
44	Check if ValidLifetime is NOT reset on NUT by RA with same prefix (before expiry, same VLT)	FAIL
45	Check if ValidLifetime is reset on NUT by RA with same prefix (after expiry, same VLT)	FAIL
46	Check if ValidLifetime is NOT reset on NUT by RA with same prefix (before expiry, less VLT)	PASS
47	Check if ValidLifetime is reset on NUT by RA with same prefix (after expiry, less VLT)	FAIL
48	Packet receiving and Global address lifetime expiry (valid preferred, valid deprecated, invalid)	FAIL
49	Packet receiving and Site-local address lifetime expiry (valid preferred, valid deprecated, invalid)	FAIL
50	Source address selection and address lifetime expiry (valid deprecated VS valid preferred)	PASS
51	Source address selection and address lifetime expiry (valid deprecated VS valid deprecated)	PASS
52	Source address selection and address lifetime expiry (invalid VS valid deprecated)	PASS
53	Source address selection and address lifetime expiry (invalid VS invalid)	FAIL

This Report was generated by TAHI IPv6 Conformance Test Suite

Table 5: IPv6 Conformance Test For Neighbor Discovery on Linux kernel 2.2.15

Test No.	Title	Result
1	Verify that the NUT send NSs (link-local ==> link-local)	FAIL
2	Verify that the NUT send NSs (global ==> global)	FAIL
3	Verify that the NUT send NSs (link-local ==> global)	FAIL
4	Verify that the NUT send NSs (global ==> link-local)	FAIL
5	Multicast NS w/ Default Config.	PASS
6	Multicast NS w/ RetransTimer=3sec.	PASS
7	Unicast NS w/ Default Config.	PASS
8	Unicast NS w/ RestransTier=3sec.	PASS
9	Address Resolution Queue (one entry for an address ?)	PASS
10	Address Resolution Queue (more then one entry for an address ?)	PASS
11	Address Resolution Queue (one entry per an address ?)	FAIL
12	Receiving valid NSs	WARN
13	Receiving invalid NSs	FAIL
14	NS vs. IsRouter flag	PASS
15	NS vs. NONCE	FAIL
16	NS vs. INCOMPLETE	FAIL
17	NS vs. REACHABLE	FAIL
18	NS vs. STALE	PASS
19	NS vs. PROBE	FAIL
20	R flag vs. IsRouter flag	FAIL
21	NA vs. NONCE	PASS
22	NA vs. INCOMPLETE	FAIL
23	NA vs. REACHABLE	FAIL
24	NA vs. STALE	FAIL
25	NA vs. PROBE	FAIL
26	Sending RS	FAIL
27	Sending RS after receiving unsolicited RA	FAIL
28	Not sending RS after receiving solicited RA	FAIL
29	Ignoring RS	PASS
30	RA set IsRouter flag	FAIL
31	Receiving multiple RAs #1	FAIL
32	Receiving multiple RAs #2	FAIL
33	Ingnoring invalid RAs	FAIL
34	ReachableTIme vs BaseReachableTime	FAIL
35	RouterLifetime=0	PASS
36	RouterLifetime=5	FAIL
37	Next-hop Determination	WARN
38	The Default Router List vs Unreachability Detection	FAIL
39	RA vs NONCE	PASS
40	RA vs INCOMPLETE	PASS
41	RA vs REACHABLE	FAIL
42	RA vs STALE	PASS
43	RA vs PROBE	FAIL
44	Redirect vs NONCE	PASS
45	Redirect vs INCOMPLETE	PASS

Table continues on next page...

Table 5 continued. . .

Test No.	Title	Result
46	Redirect vs REACHABLE	PASS
47	Redirect vs STALE	PASS
48	Redirect vs PROBE	FAIL
49	Invalid Redirect vs Neighbor Cache State	FAIL
50	Redirect vs Destination Cache; Redirect to a host	FAIL
51	Redirect vs Destination Cache; Redirect to a better router	FAIL
52	Redirect vs Neighbor Unreachability Detection; Redirect to a host	FAIL
53	Redirect vs Neighbor Unreachability Detection; Redirect to a better router	FAIL
54	Redirect vs NA w/ RFlag=0 #1	PASS
55	Redirect vs NA w/ RFlag=0 #2	FAIL
56	Redirect vs RA w/ RouterLifetime=0 #1	PASS
57	Redirect vs RA w/ RouterLifetime=0 #2	FAIL
58	Redirect vs NONCE	FAIL

This Report was generated by TAHI IPv6 Conformance Test Suite

Table 6: IPv6 Conformance Test For Neighbor Discovery on Linux kernel 2.2.20

Test No.	Title	Result
1	Verify that the NUT send NSs (link-local ==> link-local)	FAIL
2	Verify that the NUT send NSs (global ==> global)	FAIL
3	Verify that the NUT send NSs (link-local ==> global)	FAIL
4	Verify that the NUT send NSs (global ==> link-local)	FAIL
5	Multicast NS w/ Default Config.	PASS
6	Multicast NS w/ RetransTimer=3sec.	PASS
7	Unicast NS w/ Default Config.	PASS
8	Unicast NS w/ RestransTier=3sec.	PASS
9	Address Resolution Queue (one entry for an address ?)	PASS
10	Address Resolution Queue (more then one entry for an address ?)	PASS
11	Address Resolution Queue (one entry per an address ?)	FAIL
12	Receiving valid NSs	WARN
13	Receiving invalid NSs	FAIL
14	NS vs. IsRouter flag	PASS
15	NS vs. NONCE	FAIL
16	NS vs. INCOMPLETE	FAIL
17	NS vs. REACHABLE	FAIL
18	NS vs. STALE	PASS
19	NS vs. PROBE	FAIL
20	R flag vs. IsRouter flag	FAIL
21	NA vs. NONCE	PASS
22	NA vs. INCOMPLETE	FAIL
23	NA vs. REACHABLE	FAIL
24	NA vs. STALE	FAIL
25	NA vs. PROBE	FAIL
26	Sending RS	FAIL
27	Sending RS after receiving unsolicited RA	FAIL
28	Not sending RS after receiving solicited RA	FAIL
29	Ignoring RS	PASS

Table continues on next page. . .

Table 6 continued. . .

Test No.	Title	Result
30	RA set IsRouter flag	FAIL
31	Receiving multiple RAs #1	FAIL
32	Receiving multiple RAs #2	FAIL
33	Ingnoring invalid RAs	FAIL
34	ReachableTIme vs BaseReachableTime	FAIL
35	RouterLifetime=0	PASS
36	RouterLifetime=5	FAIL
37	Next-hop Determination	WARN
38	The Default Router List vs Unreachability Detection	FAIL
39	RA vs NONCE	FAIL
40	RA vs INCOMPLETE	PASS
41	RA vs REACHABLE	FAIL
42	RA vs STALE	PASS
43	RA vs PROBE	FAIL
44	Redirect vs NONCE	PASS
45	Redirect vs INCOMPLETE	PASS
46	Redirect vs REACHABLE	PASS
47	Redirect vs STALE	PASS
48	Redirect vs PROBE	FAIL
49	Invalid Redirect vs Neighbor Cache State	FAIL
50	Redirect vs Destination Cache; Redirect to a host	FAIL
51	Redirect vs Destination Cache; Redirect to a better router	FAIL
52	Redirect vs Neighbor Unreachability Detection; Redirect to a host	FAIL
53	Redirect vs Neighbor Unreachability Detection; Redirect to a better router	FAIL
54	Redirect vs NA w/ RFlag=0 #1	PASS
55	Redirect vs NA w/ RFlag=0 #2	FAIL
56	Redirect vs RA w/ RouterLifetime=0 #1	PASS
57	Redirect vs RA w/ RouterLifetime=0 #2	FAIL
58	Redirect vs NONCE	FAIL

This Report was generated by TAHI IPv6 Conformance Test Suite

Table 7: IPv6 Conformance Test For Neighbor Discovery on Linux kernel 2.4.18

Test No.	Title	Result
1	Verify that the NUT send NSs (link-local ==> link-local)	FAIL
2	Verify that the NUT send NSs (global ==> global)	FAIL
3	Verify that the NUT send NSs (link-local ==> global)	FAIL
4	Verify that the NUT send NSs (global ==> link-local)	FAIL
5	Multicast NS w/ Default Config.	PASS
6	Multicast NS w/ RetransTimer=3sec.	PASS
7	Unicast NS w/ Default Config.	PASS
8	Unicast NS w/ RestransTier=3sec.	PASS
9	Address Resolution Queue (one entry for an address ?)	PASS
10	Address Resolution Queue (more then one entry for an address ?)	PASS
11	Address Resolution Queue (one entry per an address ?)	FAIL
12	Receiving valid NSs	WARN
13	Receiving invalid NSs	FAIL
14	NS vs. IsRouter flag	PASS

Table continues on next page. . .

Table 7 continued...

Test No.	Title	Result
15	NS vs. NONCE	FAIL
16	NS vs. INCOMPLETE	FAIL
17	NS vs. REACHABLE	PASS
18	NS vs. STALE	PASS
19	NS vs. PROBE	FAIL
20	R flag vs. IsRouter flag	FAIL
21	NA vs. NONCE	PASS
22	NA vs. INCOMPLETE	FAIL
23	NA vs. REACHABLE	FAIL
24	NA vs. STALE	FAIL
25	NA vs. PROBE	FAIL
26	Sending RS	PASS
27	Sending RS after receiving unsolicited RA	PASS
28	Not sending RS after receiving solicited RA	PASS
29	Ignoring RS	PASS
30	RA set IsRouter flag	FAIL
31	Receiving multiple RAs #1	PASS
32	Receiving multiple RAs #2	PASS
33	Ingnoring invalid RAs	PASS
34	ReachableTIme vs BaseReachableTime	PASS
35	RouterLifetime=0	PASS
36	RouterLifetime=5	FAIL
37	Next-hop Determination	WARN
38	The Default Router List vs Unreachability Detection	FAIL
39	RA vs NONCE	PASS
40	RA vs INCOMPLETE	PASS
41	RA vs REACHABLE	PASS
42	RA vs STALE	PASS
43	RA vs PROBE	FAIL
44	Redirect vs NONCE	PASS
45	Redirect vs INCOMPLETE	PASS
46	Redirect vs REACHABLE	PASS
47	Redirect vs STALE	PASS
48	Redirect vs PROBE	FAIL
49	Invalid Redirect vs Neighbor Cache State	FAIL
50	Redirect vs Destination Cache; Redirect to a host	FAIL
51	Redirect vs Destination Cache; Redirect to a better router	FAIL
52	Redirect vs Neighbor Unreachability Detection; Redirect to a host	FAIL
53	Redirect vs Neighbor Unreachability Detection; Redirect to a better router	FAIL
54	Redirect vs NA w/ RFlag=0 #1	PASS
55	Redirect vs NA w/ RFlag=0 #2	FAIL
56	Redirect vs RA w/ RouterLifetime=0 #1	PASS
57	Redirect vs RA w/ RouterLifetime=0 #2	FAIL
58	Redirect vs NONCE	FAIL

This Report was generated by TAHI IPv6 Conformance Test Suite

Table 8: IPv6 Conformance Test For Stateless Address Configuration on USAGI kernel

Test No.	Title	Result
1	DAD is performed on NUT by Stateless Link-local address autoconfiguration	PASS
2	DAD Success when NUT received no packet on Stateless Link-local address autoconfiguration	PASS
3	DAD Fail when NUT received Valid NS in random delaying phase on Stateless Link-local address autoconfiguration	PASS
4	DAD Fail when NUT received Valid NS (dst MAC addr != MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
5	DAD Fail when NUT received Valid NS (dst MAC addr == MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
6	DAD Fail when NUT received Surprise NS (Prefix Option) on Stateless Link-local address autoconfiguration (Surprise test)	PASS
7	DAD Fail when NUT received Valid NA (dst MAC addr != MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
8	DAD Fail when NUT received Valid NA (dst MAC addr == MAC addr of NUT) on Stateless Link-local address autoconfiguration	PASS
9	DAD Fail when NUT received NA (No TLL option) on Stateless Link-local address autoconfiguration	PASS
10	DAD Fail when NUT received NA (dst addr == solicited node multicast) on Stateless Link-local address autoconfiguration	PASS
11	DAD Fail when NUT received Surprise NA (Many Options) on Stateless Link-local address autoconfiguration (Surprise test)	PASS
12	DAD Success when NUT received Invalid NS (Dst addr is Allnodes) on Stateless Link-local address autoconfiguration	PASS
13	DAD Success when NUT received Invalid NS (Dst addr is Tentative) on Stateless Link-local address autoconfiguration	PASS
14	DAD Success when NUT received Invalid NS (Hoplimit != 255) on Stateless Link-local address autoconfiguration	PASS
15	DAD Success when NUT received Invalid NS (Include SLL opt) on Stateless Link-local address autoconfiguration	PASS
16	DAD Success when NUT received NS (Src addr is Unicast) on Stateless Link-local address autoconfiguration	PASS
17	DAD Success when NUT received Invalid NA (Hoplimit != 255) on Stateless Link-local address autoconfiguration	PASS
18	DAD Success when NUT received Invalid NA (S flag == 1) on Stateless Link-local address autoconfiguration	PASS
19	DAD Success when NUT received NA (Dst addr is unicast) on Stateless Link-local address autoconfiguration	PASS
20	DAD is performed on NUT by Manual Link-local address configuration	PASS
21	DAD Success when NUT received no packet on Manual Link-local address configuration	PASS
22	DAD is performed on NUT by Manual Global address configuration	PASS
23	DAD Success when NUT received no packet on Manual Global address configuration	PASS
24	DAD Fail when NUT received Valid NS (dst MAC addr == MAC addr of NUT) on Manual Global address configuration	PASS
25	DAD Fail when NUT received Valid NA (dst MAC addr == MAC addr of NUT) on	

Table continues on next page...

Table 8 continued. . .

Test No.	Title	Result
	Manual Global address configuration	PASS
26	DAD Success when NUT received Invalid NS (Dst addr is Allnodes) on Manual Global address configuration	PASS
27	DAD Success when NUT received Invalid NS (Dst addr is Tentative) on Manual Global address configuration	PASS
28	DAD is performed on NUT by Stateless Global address autoconfiguration	PASS
29	DAD is performed on NUT by Stateless Global address autoconfiguration after DAD Failed for Link-local address autoconfiguration	PASS
30	ADDRCONF Success when NUT received Valid RA (Global address)	PASS
31	ADDRCONF Success when NUT received Valid RA (Site-local address)	PASS
32	ADDRCONF Success when NUT received Surprise RA (Many link-layer options) (Surprise test)	PASS
33	NUT ignores prefixopt if PreferredLifeTime > ValidLifeTime	PASS
34	NUT ignores prefixopt if Prefixlen > 64 (interface ID len is 64)	PASS
35	NUT ignores prefixopt if Prefixlen < 64 (interface ID len is 64)	PASS
36	NUT ignores prefixopt if A flag is 0	PASS
37	NUT ignores prefixopt if prefix is Link-local	PASS
38	NUT ignores prefixopt if Prefixlen > 128	PASS
39	NUT ignores prefixopt if ValidLifeTime is 0 (unknown prefix)	PASS
40	NUT ignores prefixopt if ValidLifeTime is 0 (known prefix but without IPSEC authentication)	PASS
41	NUT ignores prefixopt if prefix is Global Multicast (Surprise test)	PASS
42	Probe PrefixOptions processing order of same prefixes in one RA (Surprise test)	WARN
43	Check if ValidLifetime is reset on NUT by RA with same prefix (before expiry, greater VLT)	PASS
44	Check if ValidLifetime is NOT reset on NUT by RA with same prefix (before expiry, same VLT)	PASS
45	Check if ValidLifetime is reset on NUT by RA with same prefix (after expiry, same VLT)	PASS
46	Check if ValidLifetime is NOT reset on NUT by RA with same prefix (before expiry, less VLT)	PASS
47	Check if ValidLifetime is reset on NUT by RA with same prefix (after expiry, less VLT)	PASS
48	Packet receiving and Global address lifetime expiry (valid preferred, valid deprecated, invalid)	PASS
49	Packet receiving and Site-local address lifetime expiry (valid preferred, valid deprecated, invalid)	PASS
50	Source address selection and address lifetime expiry (valid deprecated VS valid preferred)	PASS
51	Source address selection and address lifetime expiry (valid deprecated VS valid deprecated)	PASS
52	Source address selection and address lifetime expiry (invalid VS valid deprecated)	PASS
53	Source address selection and address lifetime expiry (invalid VS invalid)	PASS

This Report was generated by TAHI IPv6 Conformance Test Suite

Table 9: IPv6 Conformance Test For Neighbor Discovery on USAGI kernel

Test No.	Title	Result
1	Verify that the NUT send NSs (link-local ==> link-local)	PASS
2	Verify that the NUT send NSs (global ==> global)	PASS
3	Verify that the NUT send NSs (link-local ==> global)	PASS
4	Verify that the NUT send NSs (global ==> link-local)	PASS
5	Multicast NS w/ Default Config.	PASS
6	Multicast NS w/ RetransTimer=3sec.	PASS
7	Unicast NS w/ Default Config.	PASS
8	Unicast NS w/ RestransTier=3sec.	PASS
9	Address Resolution Queue (one entry for an address ?)	PASS
10	Address Resolution Queue (more then one entry for an address ?)	PASS
11	Address Resolution Queue (one entry per an address ?)	PASS
12	Receiving valid NSs	WARN
13	Receiving invalid NSs	PASS
14	NS vs. IsRouter flag	PASS
15	NS vs. NONCE	PASS
16	NS vs. INCOMPLETE	PASS
17	NS vs. REACHABLE	PASS
18	NS vs. STALE	PASS
19	NS vs. PROBE	PASS
20	R flag vs. IsRouter flag	WARN
21	NA vs. NONCE	PASS
22	NA vs. INCOMPLETE	PASS
23	NA vs. REACHABLE	PASS
24	NA vs. STALE	PASS
25	NA vs. PROBE	PASS
26	Sending RS	PASS
27	Sending RS after receiving unsolicited RA	PASS
28	Not sending RS after receiving solicited RA	PASS
29	Ignoring RS	PASS
30	RA set IsRouter flag	FAIL
31	Receiving multiple RAs #1	PASS
32	Receiving multiple RAs #2	PASS
33	Ingnoring invalid RAs	PASS
34	ReachableTIme vs BaseReachableTime	PASS
35	RouterLifetime=0	PASS
36	RouterLifetime=5	FAIL
37	Next-hop Determination	WARN
38	The Default Router List vs Unreachability Detection	PASS
39	RA vs NONCE	PASS
40	RA vs INCOMPLETE	PASS
41	RA vs REACHABLE	PASS
42	RA vs STALE	PASS
43	RA vs PROBE	PASS
44	Redirect vs NONCE	PASS
45	Redirect vs INCOMPLETE	PASS

Table continues on next page. . .

Table 9 continued. . .

Test No.	Title	Result
46	Redirect vs REACHABLE	PASS
47	Redirect vs STALE	PASS
48	Redirect vs PROBE	PASS
49	Invalid Redirect vs Neighbor Cache State	FAIL
50	Redirect vs Destination Cache; Redirect to a host	FAIL
51	Redirect vs Destination Cache; Redirect to a better router	FAIL
52	Redirect vs Neighbor Unreachability Detection; Redirect to a host	FAIL
53	Redirect vs Neighbor Unreachability Detection; Redirect to a better router	FAIL
54	Redirect vs NA w/ RFlag=0 #1	PASS
55	Redirect vs NA w/ RFlag=0 #2	FAIL
56	Redirect vs RA w/ RouterLifetime=0 #1	PASS
57	Redirect vs RA w/ RouterLifetime=0 #2	FAIL
58	Redirect vs NONCE	FAIL

This Report was generated by TAHI IPv6 Conformance Test Suite

Proceedings of the
Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada