The Design and implementation of RFC3542 Support on DragonFly BSD

Huang Dashu
Mentored by Hasso Tepper

1 Introduction

The standard application program interface (API) for TCP/IP applications is the "sockets" interface. Although this API was developed for Unix in the early 1980s, it has also been implemented on DragonFly BSD with support for IPv6 applications.

While the Advanced Socket API defines interfaces used for accessing special IPv6 packet information such as the IPv6 header and the extension headers. The Advanced Socket API is also used to extend the capability of IPv6 raw sockets. Not all applications need this API, but a wide range of applications such as unicast and multicast routing daemons or network management tools such as traceroute or ping for IPv6 require the services provided by the Advanced Socket API.

Today, to fit new demands, the Advanced Socket API standard that support IPv6 applications has experience some changes from RFC2292 to RFC3542. However, the DragonFly BSD operating system now only support RFC2292, and it don't support RFC3542 advanced sockets API, to make it catch up the change, we need to make it support RFC3542.

This document will introduce how to implement RFC3542 support on DragonFly BSD. In this document, the Part 2 will introduce the significant changes from RFC3542 to RFC2292 and how to implement them in Dragonfly BSD; Part 3 is about the new features that introduced in RFC3542, and how to make Dragonfly BSD catch up these new features; Part4 is about the implementation of some tests and the test result; Part 5 is about the materials that this document and the codes has referred to.

To get more detail about the advanced socket API that defines in RFC3542, please reference to RFC3542: http://www.rfc-editor.org/rfc/rfc3542.txt

2. Significant changes that affect the compatibility to RFC 2292

2.1 Removed the IPV6_PKTOPTIONS socket option, add separate IPV6_RECVxxx socket option.

2.1.1 Description

There are six types of optional information described in RFC 2292, they are

1. The send/receive interface and source/destination address,
2. The hop limit,
3. Next hop address,
4. Hop-by-Hop options,
5. Destination options
6. Routing header.

In RFC2292, IPV6_PKTOPTIONS socket option is used for the TCP application that wants to send or receive the optional information described above without using recvmsg() and sendmsg().

In addition to the six types of information described in RFC 2292, RFC3542 adds another type of optional information “The traffic class”.

RFC3542 has removed the IPV6_PKTOPTIONS socket option and added separate IPV6_RECVxxx options to enable the receipt of the corresponding ancillary data items, and added IPV6_RTHDRDSTOPTS to specify a Destination Options header before the Routing header.

1. IPV6_RECVPKTINFO
2. IPV6_RECVHOPLIMIT
3. IPV6_RECVRTHDR
4. PV6_RECVHOPOPTS
5. IPV6_RECVDSTOPTS
6. IPV6_RECVTCLASS

2.1.2 Implementation

(1) Add below new socket options to “\netinet6\In6.h”

```
#define IPV6_RTHDRDSTOPTS  35 /* ip6_dest; send dst option before rthdr */
#define IPV6_PKTINFO       46 /* in6_pktinfo; send if, src addr */
#define IPV6_HOPLIMIT      47 /* int; send hop limit */
#define IPV6_NEXTHOP       48 /* sockaddr; next hop addr */
#define IPV6_HOPOPTS       49 /* ip6_hbh; send hop-by-hop option */
#define IPV6_DSTOPTS       50 /* ip6_dest; send dst option befor rthdr */
#define IPV6_RTHDR         51 /* ip6_rthdr; send routing header */
#define IPV6_RTHDRDSTOPTS  35 /* ip6_dest; send dst option before rthdr */
#define IPV6_RECVPKTINFO   36 /* bool; recv if, dst addr */
```

```
#define IPV6_RECVHOPLIMIT   37 /* bool; recv hop limit */
#define IPV6_RECVRTHDR      38 /* bool; recv routing header */
#define IPV6_RECVHOPOPTS    39 /* bool; recv hop-by-hop option */
#define IPV6_RECVDSTOPTS    40 /* bool; recv dst option after rthdr */
#define IPV6_RECVRTHDRDSTOPTS       41 /* bool; recv dst option before rthdr */
/* below socket option is used to support application that use RFC2292 socket API */
#define IPV6_2292PKTINFO    19 /* bool; send/recv if, src/dst addr */
#define IPV6_2292HOPLIMIT   20 /* bool; hop limit */
#define IPV6_2292NEXTHOP    21 /* bool; next hop addr */
#define IPV6_2292HOPOPTS    22 /* bool; hop-by-hop option */
#define IPV6_2292DSTOPTS    23 /* bool; destinaion option */
#define IPV6_2292RTHDR      24 /* bool; routing header */
#define IPV6_2292PKTOPTIONS25 /* buf/cmsghdr; set/get IPv6 options */
#define USE_RFC2292BIS      75   /*This means useing RFC3542 advanced Socket API*/
```

(2) Delete IPV6_PKTOPTIONS socket option from \netinet6\In6.h

(3) Update the IPv6 socket option processing function (ip6_ctloutput which is in "/netinet6/Ip6_output.c") to catch up the change.

(4) Update the codes in function "ip6_savecontrol" of the file "sys/netinet6/ip6_input.c".

2.2 Removed the ability to be able to specify Hop-by-Hop and Destination options using multiple ancillary data items. Instead using the inet6_opt_xxx() routines responsible for formatting the whole extension header.

2.2.1 Description

In RFC2292, the ancillary data object of the type of Hop-by-Hop and Destination options can appear multiple times as control information; this is done through below functions:

1. inet6_option_space
2. inet6_option_init
3. inet6_option_append
4. inet6_option_alloc
5. inet6_option_next
6. inet6_option_find

Rfc3542 has remove this kind of ability, all the Hop-by-Hop options or Destination options must be specified by as a single ancillary data object. The options is normally constructed using the function of inet6_opt_init(), inet6_opt_append(), inet6_opt_finish(), inet6_opt_set_val(), inet6_opt_next(), inet6_opt_find() and inet6_opt_get_val().

2.2.2 Implementation

Functions inet6_opt_init(), inet6_opt_append(), inet6_opt_finish(), inet6_opt_set_val(), inet6_opt_next(), inet6_opt_find() and inet6_opt_get_val() have already been defined by

dragonFly BSD in \src\lib\libc\net\Ip6opt.c, so we can just use them. To let the DragonFly BSD support inet6_opt_XXX() related man page, we can get the file “”\src\lib\libc\net\inet6_opt_init.3” from FreeBSD, put it into directory “\src\lib\libc\net”, then update the file “\src\lib\libc\net\Makefile.inc”.

To provide back support rfc2292, we don’t delete the function that used by RFC2292.

2.3 Removed the support for the loose/strict Routing header since that has been removed from the IPv6 specification.

2.3.1 Description

In RFC2292, the type 0 routing header supports up to 23 intermediate nodes. With this maximum number of intermediate nodes, a source, and a destination, and 24 hops, each of which is defined as a strict or loose hop, RFC2292 defines IPV6_RTHD_LOOSE and IPV6_RTHD_STRICT to represent these two kinds of hops.

2.3.2 Implementation

To compatible with application that use RFC2292 socket API, we don’t remove this support.

2.4 Loosened the constraints for jumbo payload option that this option was always hidden from applications

2.4.1 Description

In Rfc2292, the Jumbo payload option should not be passed back to an application and an application should receive an error if it attempts to set it. The jumbo payload option is processed entirely by the kernel, while Rfc3542 has make jumbo payload option available to the application

2.4.2 Implementation

Move the definition of Jumbo payload option data structure “ip6_opt_jumbo” form “/sys/net/pf/pf_norm.c” to the file “/sys/netinet/ip6.h”.

2.5 Disabled the use of the IPV6_HOPLIMIT sticky option.

2.5.1 Description

In RFC2292, IPV6_HOPLIMIT option can be used both as an ancillary data and as a sticky

option, while RFC3542 defines IPV6_HOPLIMIT as an ancillary-only option, because RFC3493 has define more fine-grained socket options: IPV6_UNICAST_HOPS and IPV6_MULTICAST_HOPS. What's more in RFC3542 using IPV6_RECVHOPLIMIT substitute IPV6_HOPLIMIT to let application enable this socket option.

2.5.2 Implementation

(1) In "/sys/netinet6/In6.h"
#define IPV6_RECVHOPLIMIT 37 /* bool; recv hop limit */
(2) In function "ip6_ctloutput()" (this function is used to process the IPv6 socket option) of the file "/sys/netinet6/ip6_output.c", add some lines to handle the Hop-by-hop socket option.
(3) Update the codes in function "ip6_savecontrol" of the file "sys/netinet6/ip6_input.c".

2.6 Removed the ip6_rthdr0 structure.

2.6.1 Description

Because the functionality provided by IPv6's Type 0 Routing Header can be exploited in order to achieve traffic amplification over a remote path for the purposes of generating denial-of-service traffic, RFC5095 has deprecation the implementation of RH0. See RFC5095 for more detail: http://rfc.net/rfc5095.html

2.6.2 Implementation

(1) In "/sys/netinet/ip6.h", remove the data structure of "ip6_rthdr0".
(2) In current DragonFly BSD system, the function "ip6_output()" in file "sys/netinet6/ip6_output.c" still using "ip6_rthdr0" data structure, so remove this part of codes.
(3) In function "icmp6_notify_error()" of the file "src/sys/netinet6/icmp6.c", remove data structure "ip6_rthdr0" related codes.
(4) Because RH0 is not support any more, it's relate function inet6_rth_xxx() and inet6_rthdr_xxx() in file "src/lib/libc/net/rthdr.c" will stay as stubs and wait for codes to hand type 2 headers.

2.7 Intentionally unspecified how to get received packet's information on TCP sockets.

2.7.1 Description

As described in section 1.1, RFC2292 use IPV6_PKTOPTIONS socket option to send or receive the optional information for TCP application. However, because it is unclear how a TCP application can use received information (such as extension headers) due to the lack of mapping

between received TCP segments and received operations, RFC3542 does not define how to get the received information on TCP sockets.

2.7.2 Implementation

To compatible with application that use RFC2292 socket API, we don't change the OS.

3 New features in RFC3542

3.1 Added IPV6_RTHDRDSTOPTS to specify a Destination Options header before the Routing header.

See section 1.1.

3.2 Added separate IPV6_RECVxxx options to enable the receipt of the corresponding ancillary data items.

See Section 1.1.

3.3 Added inet6_rth_xxx() and inet6_opt_xxx() functions to deal with routing or IPv6 options headers.

3.3.1 Description

Source routing in IPv6 is accomplished by specifying a routing header as an extension header, and IPv6 currently defines only the Type 0 Routing header, however RFC5095 has obsolete this type of routing header, we will just stay its related functions inet6_rth_xxx() as stubs and wait for code to handle type 2 headers.

Inet6_opt_xxx() functions has already described in section 2.2, so it will not described in this section.

3.3.2 Implementation

(1) Current Dragonfly BSD has already define the six inet6_rth_xxx() functions in "\sys\netinet6\In6.h", so we don't need to define it again.
(2) To let the DragonFly BSD support inet6_rth_XXX() related man page, we can get the file ""\src\lib\libc\net\inet6_rth_space.3" from FreeBSD, put it into directory "\src\lib\libc\net", then update the file "\src\lib\libc\net\Makefile.inc".

.

3.4 Added extensions of libraries for the "r" commands.

3.4.1 Description

Library functions that support the “r” commands hide the creation of a socket and the name resolution procedure from an application. The library functions rresvport, rcmd and rexec only support AF_INET socket and do not support AF_INET6 socket, in order to support AF_INET6 sockets for the “r” commands while keeping backward compatibility, RFC2292 has defined rresvport_af() function that behaves the same as rresvport() function, but provide support to both AF_INET socket and AF_INET6 socket.

Besides rresvport_af() function, RFC3542 has defined two more library functions: rcmd_af() and rexec_af(). These two functions behave same as function rcmd() and rexec() separately, but provide support to both AF_INET socket and AF_INET6 socket.

3.4.2 Implementation

Current Dragonfly BSD have already implement functions rresvport_af() and rcmd_af() in “/src/lib/libc/net/rcmd.c”, so we just need to implement function rexec_af() in “/src/lib/libcompat/4.3/rexec.c”.

3.5 Introduced additional IPv6 option definitions.

Besides the socket options that have described in previous content, RFC3542 have defined below additional IPv6 options relative to RFC2292. (Note: Current Dragonfly BSD has already included some this kind of additional IPv6 option, and we will not describe them here)

(1) Options that will put into the file “\netinet6\In6.h”.

```
#define IPV6_DONTFRAG          62 /* bool; disable IPv6 fragmentation */
#define IPV6_PATHMTU           44 /* mtuinfo; get the current path MTU (sopt),
                                     4 bytes int; MTU notification (cmsg) */
#define IPV6_USE_MIN_MTU    42 /* bool; send packets at the minimum MTU */
#define IPV6_RECVPATHMTU    43 /* bool; notify an according MTU */
#define IPV6_TCLASS         61 /* int; send traffic class value */
```

(2) Options that will put into the file “\netinet\Icmp6.h”.

```
#define MLD_LISTENER_REDUCTION MLD_LISTENER_DONE /* RFC3542 definition */
#define ICMP6_DST_UNREACH_BEYONDSCOPE 2      /* beyond scope of source address */
```

(3) Options that will put into the file “\netinet\Ip6.h”.

```
#define IP6OPT_ROUTER_ALERT      0x05        /* 00 0 00101 (RFC3542, recommended) */
/* Router alert values (in network byte order) */
#if BYTE_ORDER == BIG_ENDIAN
#define IP6_ALERT_MLD    0x0000
#define IP6_ALERT_RSVP   0x0001
```

```
#define IP6_ALERT_AN       0x0002
#else
#if BYTE_ORDER == LITTLE_ENDIAN
#define IP6_ALERT_MLD    0x0000
#define IP6_ALERT_RSVP   0x0100
#define IP6_ALERT_AN       0x0200
#endif /* LITTLE_ENDIAN */
#endif
```

(4) Move the definition of structures “ip6_opt_jumbo”, “ip6_opt”, “ip6_opt_nsap”, “ip6_opt_tunnel”, “ip6_opt_router” from file “/sys/net/pf/pf_norm.c” to file “/sys/netinet/ip6.h”.

3.6 Added MLD and router renumbering definitions.

Current Dragonfly BSD has covered all the MLD and router renumbering definitions except the definition of “MLD_LISTENER_REDUCTION”, which is described in section 3.5.

3.7 Added options and ancillary data items to manipulate the traffic class field.

3.7.1 Description

In RFC3542, if an application wants to receive the traffic class value, it can enable the IPV6_RECVTCLASS socket option. And if an application wants to override either the kernel’s default or a previously specified traffic class value, it can use setsocketopt() or by specifying the control information as ancillary data for sendmsg().

3.7.2 Implementation

Carefully read related codes in FreeBSD and DragonFly BSD and do below updates to DragonFly BSD source codes.

(1) Add below definition into file “/sys/netinet6/in6.h”

```
#define IPV6_RECVTCLASS              57 /* bool; recv traffic class values */
#define IPV6_TCLASS         61 /* int; send traffic class value */
```

(2) Add below definition into file “/sys/netinet/in_pcb.h”

```
#define    IN6P_TCLASS              0x400000 /* receive traffic class value */
```

(3) In function “ip6_ctloutput()” of file “/sys/netinet6/ip6_output.c” add the socket set and get process for “IPV6_RECVTCLASS” and “IPV6_TCLASS”. Do more changes for the functions of “ip6_getpcbopt()”, “ip6_clearpktopts()”, “ip6_setpktoption()” in file “/sys/netinet6/ip6_output.c”.

3.8 Added MTU-related socket options and ancillary data items.

3.8.1 Sending with the minimum MTU

[1] Description

Rfc3542 defines a mechanism to avoid path MTU discovery by sending at the minimum IPv6 MTU, application can do this through a new socket option “IPV6_USE_MIN_MTU” or by sending this option as ancillary data. For example a unicast application can disable pathe MTU discovery through below lines:

```
int on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_USE_MIN_MTU, &on, sizeof(on));
```

This option takes three types of integer arguments:

(1) -1 : Means perform path MTU discovery for unicast destinations but do not form multicast destinations.
(2) 0 : Always perform path MTU discovery.
(3) 1 : Always disable path MTU discovery.

[2] Implementation

(1) In “/sys/netinet6/in6.h” add the definition of IPV6_USE_MIN_MTU

```
#define IPV6_USE_MIN_MTU    42 /* bool; send packets at the minimum MTU */
```

(2) In function “ip6_output()” of the file “/sys/netinet6/ip6_output.c”, change the MTU determine process for the output packet according to rule that described above.

(3) In function “ip6_ctloutput()” of file “/sys/netinet6/ip6_output.c” add the socket set and get process for “IPV6_USE_MIN_MTU”, Do more changes for the functions of “ip6_getpcbopt()”, “ip6_setpktoption()” in file “/sys/netinet6/ip6_output.c”.

3.8.2 Sending without fragmentation

[1] Description

By default, if packet is too big for the path MTU, a fragment header will automatically insert into the packet, but some applications such as traceroute6 might not want this behavior. RFC3542 provides IPV_DONTFRAG socket option to disable this behavior. This socket option can be used as below.

```
int on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_DONTFRAG, &on, sizeof(on));
```

[2] Implementation

(1) In “/sys/netinet6/in6.h” add the definition of IPV6_DONTFRAG.

```
#define IPV6_DONTFRAG          62 /* bool; disable IPv6 fragmentation */
```

(2) In function “ip6_output()” of the file “/sys/netinet6/ip6_output.c”, Add IPV6_DONTFRAG related codes for the outgoing packet.

In function “ip6_ctloutput()” of file “/sys/netinet6/ip6_output.c” add the socket set and get process for “IPV6_DONTFRAG”, Do more changes for the functions of “ip6_getpcbopt()”,

“ip6_setpktoption()” in file “/sys/netinet6/ip6_output.c”

3.8.3 Path MTU Discovery and UDP

[1] Description

Some UDP and raw socket applications may want to get the path MTU (maximum send transport-message size) to a given destination, this is accomplished using a new ancillary data item (IPV6_PATHMTU) which is delivered to recvmsg() without any actual data. RFC3542 provides a new socket option IPV6_RECVPATHMTU that applications can use to enable the receipt of IPV6_PATHMTU ancillary data items.

For example, to enable the receipt of IPV6_PATHMTU, application can do it as below.

```
int on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_RECVPATHMTU, &on, sizeof(on));
```

RFC3542 has also defined data structure “ip6_mtuinfo” that used to carry the path MTU information.

[2] Implementation

(1) In “/sys/netinet6/in6.h” add the definition of IPV6_RECVPATHMTU.

```
#define IPV6_RECVPATHMTU    43 /* bool; notify an according MTU */
```

(2) Add structure ip6_mtuinfo to the file “sys/netinet6/in6.h”

```
struct ip6_mtuinfo {
    struct sockaddr_in6 ip6m_addr;     /* or sockaddr_storage? */
    uint32_t ip6m_mtu;
};
```

(3) Add function “ip6_notify_pmtu” to file “/sys/netinet6/ip6_input.c”. This function is used to notify the new path MTU for a destination to the application. And will be called in the function of “in6_pcbnotify()” in file “/sys/netinet6/in6_pcb.c”

(4) In function “ip6_ctloutput()” of file “/sys/netinet6/ip6_output.c” add the socket set and get process for “IPV6_RECVPATHMTU”.

3.8.4 Determining the Current Path MTU

[1] Description

As describe in section 3.8.3, RFC3542 defines a get-only socket option “IPV6_PATHMTU” to retrieve the current path MTU value for the destination of a given connected socket, this option takes a pointer to the “ip6_mtuinfo” structure as the fourth argument, and the size of the structure should be passed as a value-result parameter in the fifth argument. This socket option can be used as below:

```
struct ip6_mtuinfo mtuinfo;
socklen_t infolen = sizeof(mtuinfo);
getsockopt(fd, IPPROTO_IPV6, IPV6_PATHMTU, &mtuinfo, &infolen);
```

[2] Implementation

(1) In “/sys/netinet6/in6.h” add the definition of “IPV6_PATHMTU”.

#define IPV6_PATHMTU 44

(2) In function “ip6_ctloutput()” of file “/sys/netinet6/ip6_output.c” add the socket get process for “IPV6_PATHMTU”.

4 Test the codes.

Before doing the test, we should first update the development DragonFlyBSD source codes with RFC3542 related patches. Then, we need to rebuild and install the world, rebuild and install the kernel is also needed.

4.1 Test the Ping6 command

Remove below codes in the file “/src/sbin/ping6/Makefile.c”.

CFLAGS+=-DINET6 –DIPSEC

Add below code to the file “/src/sbin/ping6/Makefile.c”.

CFLAGS+=-DINET6 -DIPSEC -DUSE_RFC3542

This will make “ping6” command using the advanced socket API that defined in RFC3542.

4.1.1 Test environment

Using two DragonFly BSD computers, one computer will use the stable kernel and configured with IPv6 address “2001::1”; another computer will use the development kernel (the latest kernel that has been rebuilt after updated with my codes) and configured with IPv6 address 2001::2. Then, in the shell of the second computer, input below commands to do the test.

4.1.2 Test commands

(1) # ping6 2001::1

This command can test below RFC3542 related Advanced socket API

1) IPV6_USE_MIN_MTU socket option
2) ICMP6_FILTER socket option
3) IPV6_PKTINFO socket option
4) IPV6_UNICAST_HOPS socket option
5) IPV6_MULTICAST_HOPS socket option
6) IPV6_RTHDR socket option
7) IPV6_RECVPKTINFO socket option
8) IPV6_RECVHOPLIMIT socket option
9) inet6_opt_next()
10) inet6_opt_get_val()

(2) #ping6 –m 2001::1

This command can test IPV6_RECVPATHMTU socket API.

(3) #ping6 –v 2001::1

This command can test below RFC3542 related Advanced socket API

1) IPV6_RECVHOPOPTS socket option
2) IPV6_RECVDSTOPTS socket option
3) IPV6_RECVRTHDRDSTOPTS socket option
4) IPV6_RECVRTHDR socket option

(4) #ping6 –h 1 2001::1

This command can test IPV6_HOPLIMIT CMSG data.

4.2 Test the Traceroute6 command

4.2.1 Test environment

Remove below codes in the file “/src/usr.sbin/traceroute6/Makefile”.

```
CFLAGS+=-DINET6 -DIPSEC -DHAVE_POLL
```

Add below code to the file “/src/usr.sbin/traceroute6/Makefile”.

```
CFLAGS+=-DINET6 -DIPSEC -DHAVE_POLL -DUSE_RFC3542
```

This will make “traceroute6” command using the advanced socket API that defined in RFC3542.

4.2.2 Test commands

(1) traceroute6 2001::1

This command can test below RFC3542 related Advanced socket API

1) IPV6_RECVPKTINFO socket option
2) IPV6_RECVHOPLIMIT socket option
3) IPV6_RTHDR socket option
4) IPV6_UNICAST_HOPS socket option

5 References

[1] W. Stevens, M. Thomas, E. Nordmark, T. Jinmei. “Advanced Sockets Application Program Interface (API) for IPv6”. RFC 3542. May 2003.

[2] W. Stevens, M. Thomas, AltaVista. “Advanced Socket API for IPv6”. RFC2292. February 1998.

[3] R. Gilligan, S. Thomson, J. Bound, J.McCann, W. Stevens. “Basic Socket Interface Extensions for IPv6”. RFC3493. February 2003.

[4] Qing Li, Tatuya Jinmei, Keiichl Shima. “ipv6_core_protocols_implementation”. Morgan Kaufmann. October 12, 2006.

[5] Qing Li, Tatuya Jinmei, Keiichl Shima. “IPv6 Advanced Protocols Implementation”. Morgan Kaufmann. April 6, 2007.

[6] DragonFly BSD Source Code. http://www.dragonflybsd.org/cvsweb/src/

[7] Free BSD Source Code. http://www.freebsd.org/cgi/cvsweb.cgi/src/

[8] Net BSD Source Code. http://cvsweb.netbsd.org/bsdweb.cgi/src/

[9] DragonFly BSD Guides. http://www.dragonflybsd.org/docs/guides.shtml