Advanced Sockets API for IPv6

yhmiu

RFC 2292
RFC 1883

Outline

- IPv6 Raw sockets
- Ancillary Data
- Options described using ancillary data
  - Packet information
  - Hop-By-Hop Options
  - Destination Options
  - Routing Header Option

IPv6 Raw sockets

- Raw sockets bypass the transport layer（TCP or UDP）
- All fields in the IPv6 header that an application might want to change can be modified using ancillary data and/or socket options by the application for output
- All fields in a received IPv6 header and all extension headers are also made available to the application as ancillary data on input

Example of ICMPv6 Raw socket

- Calculate and insert the ICMPv6 checksum for ICMPv6 raw socket

```
fd=socket(PF_INET6,SOCK_RAW,IPPROTO_ICMPV6);
int offset=2;
setsockopt(fd,IPPROTO_IPV6,IPV6_CHECKSUM,&offset,sizeof(offset));
```

Ancillary Data (1)

- Be used to exchange the following optional information between the AP and kernel
  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

Ancillary Data (2)

- 4.2BSD allowed file descriptors to be transferred between separate processes across a UNIX domain socket using sendmsg() and recvmsg() functions
- Two member of msghdr structure will be used
  - msg_control
  - msg_controllen

Ancillary Data (3)

- Ancillary data object
  - Defined by the cmsghdr structure
- Functions that operate on the ancillary data objects
  - GMSG_FIRSTHDR()
  - GMSG_NXTHDR()
  - GMSG_DATA()
  - GMSG_SPACE()
  - GMSG_LEN()

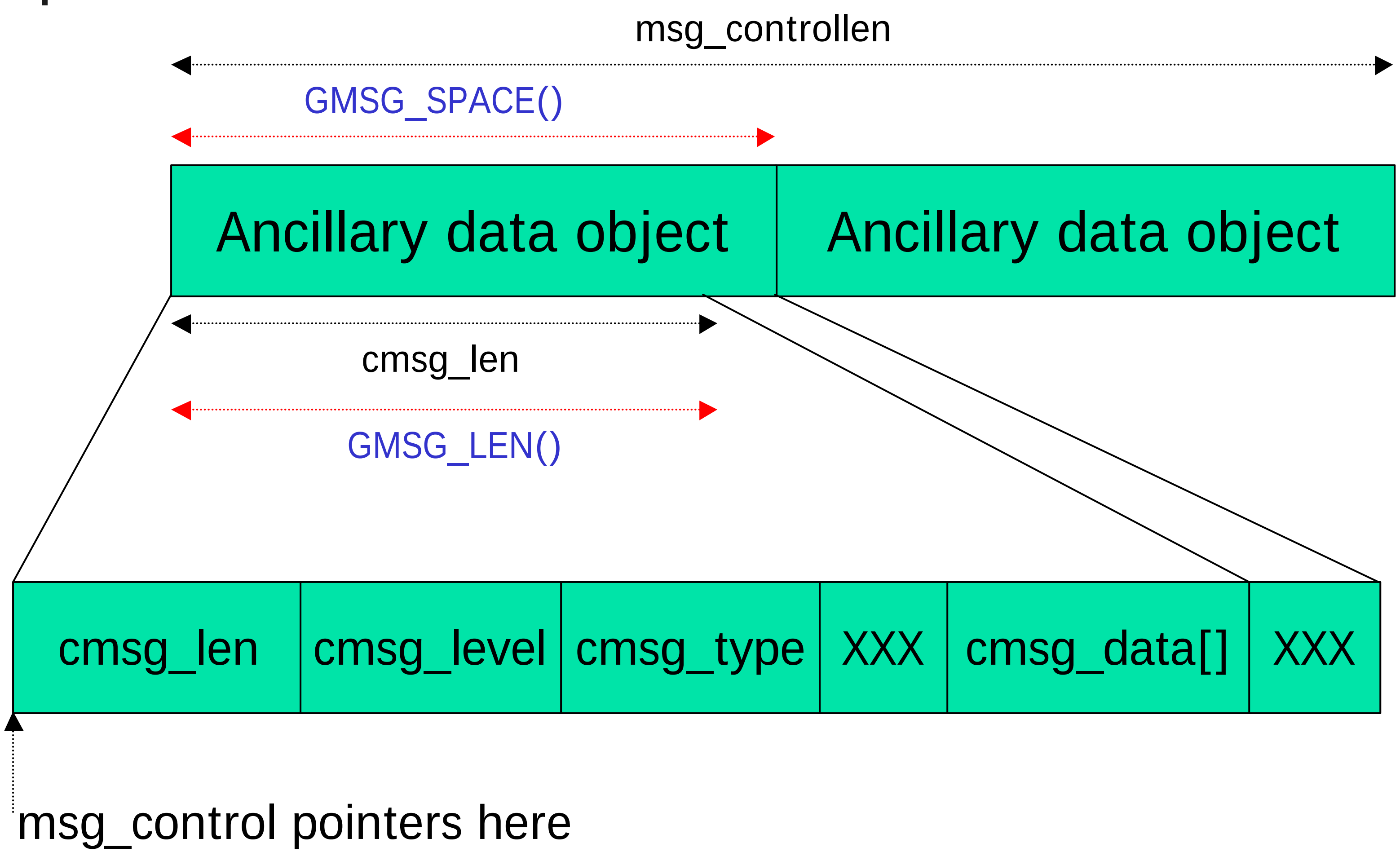
The msghsr structure

```
struct msghdr {
   void          *msg_name;        /* ptr to socket address structure */
   socklen_t     msg_namelen;    /* size of socket address structure */
   struct iovec  *msg_iov;       /* scatter/gather array */
   size_t        msg_iovlen;      /* # elements in msg_iov */
   void          *msg_control;    /* ancillary data */
   socklen_t     msg_controllen;  /* ancillary data buffer length */
   int           msg_flags;       /* flags on received message */
 };
```

The cmsghdr structure

```
struct cmsghdr {
    socklen_t  cmsg_len;   /* #bytes, including this header */
    int        cmsg_level; /* originating protocol */
    int        cmsg_type;  /* protocol-specific type */
              /* followed by unsigned char cmsg_data[]; */
  };
```

Ancillary Data (4)



Ancillary data object macros(1)

- GMSG_FIRSTHDR()
  - struct cmsghdr *CMSG_FIRSTHDR(const struct msghdr *mhdr);
  - Return a pointer to the first cmsghdr structure in the msghdr structure pointed to by mhdr
- GMSG_NXTHDR()
  - struct cmsghdr *CMSG_NXTHDR(const struct msghdr *mhdr, const struct cmsghdr *cmsg);
  - Return a pointer to the cmsghdr structure describing the next ancillary data object
  - mhdr is a pointer to a msghdr structure and cmsg is a pointer to a cmsghdr structure
  - If there is not another ancillary data object, the return value is NULL

Ancillary data object macros(2)

- GMSG_DATA()
  - unsigned char *CMSG_DATA(const struct cmsghdr *cmsg);
  - Return a pointer to the data (cmsg_data[]) following a cmsghdr structure
- GMSG_SPACE()
  - unsigned int CMSG_SPACE(unsigned int length);
  - returns the space required by the object and its cmsghdr structure, including any padding needed to satisfy alignment requirements

Ancillary data object macros(3)

- CMSG_LEN()
  - unsigned int CMSG_LEN(unsigned int length);
  - returns the value to store in the cmsg_len member of the cmsghdr structure, taking into account any padding needed to satisfy alignment requirements

Options described using acillary data

1. The application must call setsockopt() to turn on the corresponding flag, then to receive any of this optional information
2. When any of these options are enabled, the corresponding data is returned as control information by recvmsg(), as one or more ancillary data objects
3. **在傳送端,** the application just calls sendmsg() and specifics one or more ancillary data objects as control information.**並不需要像接收端一樣要**call setsockopt()

cmsghdr fields of the ancillary data objects

cmsg_level	cmsg_type	cmsg_data[]
IPPROTO_IP6	IPV6_PKTINFO	in6_pktinfo structure
IPPROTO_IP6	IPV6_HOPLIMIT	int
IPPROTO_IP6	IPV6_NEXTHOP	Socket address structure
IPPROTO_IP6	IPV6_HOPOPTS	implementation dependent
IPPROTO_IP6	IPV6_DSTOPTS	implementation dependent
IPPROTO_IP6	IPV6_RTHDR	implementation dependent

Routing header (1)

- IPv6 current define only Type 0 Routing header
  - This type supports up to 23 intermediate nodes
  - Each hop is defined as a strict or loose hop

Routing header (2)

- 4 functions build a Routing header
  - inet6_rthdr_space()
    - Return #bytes required for ancillary data
  - inet6_rthdr_init()
    - Initialize ancillary data for Routing header
  - inet6_rthdr_add()
    - Add IPv6 address & flags to Routing header
  - inet6_rthdr_lasthop()
    - Specify the flags for the final hop

Routing header (3)

- 4 functions deal with a returned Routing header
  - inet6_rthdr_reverse()
    - Reverse a Routing header
  - inet6_rthdr_segments()
    - Return #segments in a Routing header
  - inet6_rthdr_getaddr()
    - Fetch one address from a Routing header
  - inet6_rthdr_getflags()
    - Fetch one flag from a Routing header
- **以上8個**function**皆被定義於**<netinet6/in6.h>

Routing header (4)

- In order to receive a Routing header, the application must enable the IPV6_RTHDR option

```
int  on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_RTHDR, &on, sizeof(on));
```

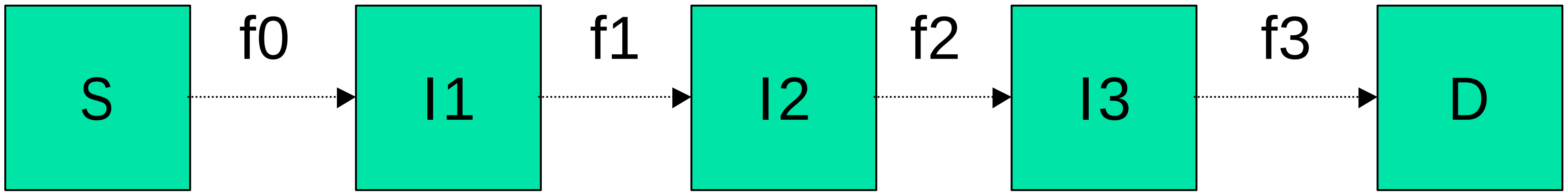
- To send a Routing header the application just specifies it as ancillary data in a call to sendmsg()

補充

- The following constants are defined in the <netinet6/in6.h> header

```
#define IPV6_RTHDR_LOOSE    0 /* this hop need not be a neighbor */
#define IPV6_RTHDR_STRICT   1 /* this hop must be a neighbor */
#define IPV6_RTHDR_TYPE_0   0 /* IPv6 Routing header type 0 */
```

Routing header example



src:	*	S	S	S	S	S
dst:	D	I1	I2	I3	D	D
A[1]:	I1	I2	I1	I1	I1	I1
A[2]:	I2	I3	I3	I2	I2	I2
A[3]:	I3	D	D	D	I3	I3
#seg:	3	3	2	1	0	3

check: f0 f1 f2 f3

S端AP在call sendmsg()前該做的事

```
void  *ptr;
    struct msghdr  msg;
    struct cmsghdr  *cmsgptr;
    struct sockaddr_in6  I1, I2, I3, D;
    unsigned int  f0, f1, f2, f3;

    ptr = malloc(inet6_rthdr_space(IPV6_RTHDR_TYPE_0, 3));
    cmsgptr = inet6_rthdr_init(ptr, IPV6_RTHDR_TYPE_0);

    inet6_rthdr_add(cmsgptr, &I1.sin6_addr, f0);
    inet6_rthdr_add(cmsgptr, &I2.sin6_addr, f1);
    inet6_rthdr_add(cmsgptr, &I3.sin6_addr, f2);
    inet6_rthdr_lasthop(cmsgptr, f3);

    msg.msg_control = ptr;
    msg.msg_controllen = cmsgptr->cmsg_len;

    /* finish filling in msg{}, msg_name = D */
    /* call sendmsg() */
```

call inet6_rthdr_init() initializes the ancillary data object to contain a Type 0 Routing header

0 … 31

cmsg_len=20			
cmsg_level=IPPRTO_IPV6			
cmsg_type=IPV6_RTHDR			
Next Header	Hdr Ext Len=0	Routing Type=0	Seg Left=0
Reserved	Strict/Loose Bit Map		

The first call to inet6_rthdr_add() adds I1 to the list

<table>
<tr><td colspan="2">0</td><td colspan="3" align="right">31</td></tr>
<tr><td colspan="5">cmsg_len=36</td></tr>
<tr><td colspan="5">cmsg_level=IPPRTO_IPV6</td></tr>
<tr><td colspan="5">cmsg_type=IPV6_RTHDR</td></tr>
<tr><td>Next Header</td><td colspan="2">Hdr Ext Len=2</td><td>Routing Type=0</td><td>Seg Left=1</td></tr>
<tr><td>Reserved</td><td>X</td><td colspan="3">Strict/Loose Bit Map</td></tr>
<tr><td colspan="5">Address[1]=I1</td></tr>
</table>

The next call to inet6_rthdr_add() adds I2 to the list

<table>
<tr><td>0</td><td></td><td></td><td></td><td>31</td></tr>
<tr><td colspan="5">cmsg_len=52</td></tr>
<tr><td colspan="5">cmsg_level=IPPRTO_IPV6</td></tr>
<tr><td colspan="5">cmsg_type=IPV6_RTHDR</td></tr>
<tr><td>Next Header</td><td colspan="2">Hdr Ext Len=4</td><td>Routing Type=0</td><td>Seg Left=2</td></tr>
<tr><td>Reserved</td><td>X</td><td>X</td><td colspan="2">Strict/Loose Bit Map</td></tr>
<tr><td colspan="5">Address[1]=I1</td></tr>
<tr><td colspan="5">Address[2]=I2</td></tr>
</table>

The last call to inet6_rthdr_add() adds I3 to the list

<table>
<tr><td>0</td><td colspan="5"></td><td>31</td></tr>
<tr><td colspan="7">cmsg_len=68</td></tr>
<tr><td colspan="7">cmsg_level=IPPRTO_IPV6</td></tr>
<tr><td colspan="7">cmsg_type=IPV6_RTHDR</td></tr>
<tr><td>Next Header</td><td colspan="3">Hdr Ext Len=6</td><td>Routing Type=0</td><td colspan="2">Seg Left=3</td></tr>
<tr><td>Reserved</td><td>X</td><td>X</td><td>X</td><td colspan="3">Strict/Loose Bit Map</td></tr>
<tr><td colspan="7">Address[1]=I1</td></tr>
<tr><td colspan="7">Address[2]=I2</td></tr>
<tr><td colspan="7">Address[3]=I3</td></tr>
</table>

the call to inet6_rthdr_lasthop() sets the next bit of the Strict/Loose Bit Map to the value specified by f3

<table>
<tr><td>0</td><td colspan="5"></td><td>31</td></tr>
<tr><td colspan="7">cmsg_len=68</td></tr>
<tr><td colspan="7">cmsg_level=IPPRTO_IPV6</td></tr>
<tr><td colspan="7">cmsg_type=IPV6_RTHDR</td></tr>
<tr><td>Next Header</td><td colspan="4">Hdr Ext Len=6</td><td>Routing Type=0</td><td>Seg Left=3</td></tr>
<tr><td>Reserved</td><td>X</td><td>X</td><td>X</td><td>X</td><td colspan="2">Strict/Loose Bit Map</td></tr>
<tr><td colspan="7">Address[1]=I1</td></tr>
<tr><td colspan="7">Address[2]=I2</td></tr>
<tr><td colspan="7">Address[3]=I3</td></tr>
</table>