ANEP Socket
User Guide

Amarrage Project
LORIA-INRIA
Stéphane D'Alu

Thu Apr 27 11:50:24 CEST 2000

Contents

1 Structures **1**
1.1 sockaddr_anep . . . 1
1.2 anep_optionhdr . . . 1
1.3 anep_optype . . . 1
1.4 anep_optbuffer . . . 1

2 Functions **2**
2.1 `sendmsg`, `recvmsg functions` . . . 2
2.1.1 `ANEP_OPTION` . . . 2
2.1.2 `ANEP_DISCARD` . . . 3
2.2 `setsockopt` and `getsockopt` functions . . . 3
2.2.1 `ANEP_OPTYPE` Socket option . . . 3
2.2.2 `ANEP_DISCARD` Socket option . . . 4
2.2.3 `ANEP_PKTOPTIONS` Socket option . . . 4
2.2.4 `ANEP_OPTBUFFER` Socket option . . . 4
2.3 `ioctl` function . . . 5
2.4 Other functions . . . 5

1 Structures

1.1 sockaddr_anep

This structure will be used to specify the ANEP address in the same way that `sockaddr_in` or `sockaddr_in6` were used.

```
struct sockaddr_anep {
    u16             sanep_family;   /* AF_{INET,INET6}     */
    u16             sanep_type;     /* Type ID             */
    u32             unused          /* (64bits alignment)  */
    union {
        struct {
            u32              flowinfo;  /* IPv6 flow information */
            u32              scope_id;  /* scope id (RFC2553)    */
            struct in6_addr  addr;      /* IPv6 address          */
        } ipv6;
        struct {
            struct in_addr   addr;      /* IPv4 address          */
        } ipv4;
    } addr_u;
};

#define sanep_ipv6_addr         addr_u.ipv6.addr
#define sanep_ipv6_flowinfo     addr_u.ipv6.flowinfo
#define sanep_ipv6_scope_id     addr_u.ipv6.scope_id
#define sanep_ipv4_addr         addr_u.ipv4.addr
```

1.2 anep_optionhdr

This structure reflect the header of an ANEP option and is used when building or reading ANEP options.

```
struct anep_optionhdr {
    u16  type;     /* option type including flags                         */
    u16  length;   /* option length in 32bits word                        */
    char data[0];  /* reference to the option data (not available in ANSI) */
};
```

1.3 anep_optype

This structure will be used with `setsockopt` and `getsockopt` to set or get filtering rules on anep packet or option according to the option type.

```
struct anep_optype {
    u16  action;    /* type of action */
    u16  beg_type;  /* beginning type */
    u16  end_type;  /* ending type    */
};
```

1.4 anep_optbuffer

This structure will be used with `setsockopt` and `getsockopt` to set or get the list of buffers to use for sending or receiving option.

```
struct anep_optbuffer {
    void *buff;    /* buffer start pointer */
    u16   length;  /* length in bytes      */
    u16   type;    /* option type          */
};
```

2 Functions

2.1 sendmsg, recvmsg functions

```
#include <sys/socket.h>

ssize_t sendmsg(int sockfd, struct msghdr *msg, int flags);
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

Returns 0 if OK, -1 on error

Protocol	*cmsg_level*	*cmsg_type*	Description
ANEP	SOL_ANEP	ANEP_OPTION	sending of anep options
		ANEP_DISCARD	forward packet if unknown type

Table 1: Summary of ancillary data

ANEP options are sent/received by using the first of the following method if available: ancillary data, sticky options, option buffers.

2.1.1 ANEP_OPTION

This allow to specify anep options, but 1 option per control message only, if you have to set several options you will need to use several `ANEP_OPTION` messages. The message contains the `anep_optionhdr` structure followed by the option's data.

In the following example we assume that `anep_option_data_size` is the size of useful data to be put in the option and is a *4 bytes multiple* (because the length of an option is expressed in 32bits words), and that `anep_option_type` is the type id of the option. Line 2, the total size of the option is determined; lines 4 to 6, the control message header is initialized; lines 8 to 10, the anep option header is initialized and we are also setting the `discard` bit with `ANEP_OPT_DISCARD`, and line 12 we are now ready to write the option data.

```
1  anep_option_size = (anep_option_data_size + sizeof(struct anep_optionhdr);
2
3  cmsg->cmsg_len   = CMSG_LEN(anep_option_size);
4  cmsg->cmsg_level = SOL_ANEP;
5  cmsg->cmsg_type  = ANEP_OPTION;
6
7  aoh              = (struct anep_optionhdr *)CMSG_DATA(cmsg);
8  aoh->type        = htons(ANEP_OPT_DISCARD | anep_option_type);
9  aoh->length      = htons(anep_option_size >> 2);
10
11 anep_option_data = (char *)(aoh+1);
```

The following anep option type have been registered by the ANANA[1], the option type marked as *kernel* are computed by the kernel and if such a type is specified it will be silently ignored and the kernel computed one put instead.

[1] Active Networks Assigned Numbers Authority

Option type	Kernel	Description
`ANEP_OPTION_SRCID`		Source identification
`ANEP_OPTION_DESTID`		Destination identification
`ANEP_OPTION_CHECKSUM`	•	Checksum
`ANEP_OPTION_NNAUTH`		Authentication

Table 2: Summary of registered anep option types

When using `recvmsg` the ancillary data corresponding to the anep option will be available only according to the rules specified by the `ANEP_OPTYPE` `setsockopt`, the same is true for `recvfrom` but it also depend of the option buffers set by `ANEP_OPTBUFFER` `setsockopt`.

2.1.2 `ANEP_DISCARD`

The ancillary data is an integer which will be interpreted as a boolean to set or not the `discard` bit on the anep packet (this is not the same as the one on the anep option), if set and the receiver doesn't know this packet anep type, the packet will be discard instead of trying to be forwarded.

2.2 `setsockopt` and `getsockopt` functions

```
#include <sys/socket.h>

int getsockopt(int sockfd, int level, int optname, void *optval,
               socklen_t *optlen)
int setsockopt(int sockfd, int level, int optname, const void *optval,
               socklen_t *optlen)
```

Returns 0 if OK, -1 on error

level	*optname*	get	set	Description	Flag	Datatype
`SOL_ANEP`	`ANEP_OPTYPE`	•	•	filter/select anep options		`anep_optype{}[]`
	`ANEP_DISCARD`	•	•	forward packet if unknown type	•	`int`
	`ANEP_PKTOPTIONS`	•	•	permit sending of sticky options		`cmsghdr{}[]`
	`ANEP_OPTBUFFER`	•	•	buffers for anep options		`anep_optbuffer{}[]`

Table 3: Summary of socket options for `getsockopt` and `setsockopt`

2.2.1 `ANEP_OPTYPE` Socket option

This option allow to specify a list of rules to know what action should be performed on an option or the packet depending on the option types present in it.

All rules are processed and it is the last matching rule which give the action to be done, and if no rules are given (ie: *optlen* or *optval* are null) all the options are rejected. The list of possible actions is described in the table 4.

Each rule is described by the following structure:

```
struct anep_optype {
    u16 action;    /* type of action */
    u16 beg_type;  /* beginning type */
    u16 end_type;  /* ending type    */
};
```

The integers `beg_type` and `end_type` define a contiguous zone of option types which will have the same `action` associated to them. So you need to have the following relation because the type of an option is coded using 14 bits: $0 <= beg\_type <= end\_type <= ANEP\_OPT\_TYPE\_MAX = 16383$

The maximal size which can be transfered to the kernel for this option is `ANEP_RESTRICTED_OPTYPE_SIZE`.

<table>
<tr><th>ANEP_OPTYPE_KNOW</th><th>ANEP_OPTYPE_MEMORIZE</th><th>Description</th></tr>
<tr><td>•</td><td></td><td>the application accept this option, but doesn't want to retrieve its value.</td></tr>
<tr><td>•</td><td>•</td><td>the application accept this option and will be able to retrieve its value by using <code>recvmsg()</code>.</td></tr>
<tr><td></td><td>•</td><td rowspan="2">the application doesn't know this option, either this option will be ignored or the packet destroyed according to the value of the option's flag discard</td></tr>
<tr><td></td><td></td></tr>
</table>

Table 4: Actions specified in rules for anep option filtering

2.2.2 `ANEP_DISCARD` Socket option

This option specifies that we don't want the receiving host to forward this packet if it is not able to understand this particular anep type.

2.2.3 `ANEP_PKTOPTIONS` Socket option

This option allow to have sticky ANEP options, so instead of sending them every call to `sendmsg`, they are set using the `setsockopt` call. The format of this buffer is the same as when using ancillary data (ie: `optval = msg.msg_control` and `optlen = msg.msg_controllen`), but here every message should have the type `ANEP_OPTION`. If there is any kind of ancillary data present when calling `sendmsg`, these sticky packet options won't be used.

The maximal size which can be transfered to the kernel for this option is `ANEP_RESTRICTED_PKTOPTIONS_SIZE`.

> *The kernel try to catch invalid* `ANEP_PKTOPTIONS` *data, but it is still possible to have every call to* `sendto` *or* `sendmsg` *failing due to* bad *options. One case would be to have to much options so that the network layer cannot send the packet because it is too big.*

2.2.4 `ANEP_OPTBUFFER` Socket option

This is used to allow `sendto` and `recvfrom` to send and receive ANEP options (this works also with `sendmsg` and `recvmsg` when there is no ancillary data and no sticky options).

```
struct anep_optbuffer {
    void *buff;    /* buffer      */
    u16   type;    /* option type */
    u16   length;  /* length      */
};
```

Three constants are defined to specify if one want to set or get the option buffers for sending, receiving or both (both is only possible for setting):

- `ANEP_OPTBUFFER_SEND`: for sending
- `ANEP_OPTBUFFER_RECV`: for receiving

This is used by having the *first* entry in the array of option buffers (structure `anep_optbuffer`) setup as follow:

- the pointer (field `buff`) is null
- the type (field `type`) is a combinaison of `ANEP_OPTBUFFER_SEND` and `ANEP_OPTBUFFER_RECV`

If the *first* entry is not used as described above, it is assumed that one want to set the option buffers for both: sending and receiving.

The macro `ANEP_OPTBUFFER_SET_SIDE` can be used to easy initialize the first entry, as shown in the following example:

```
struct anep_optbuffer optbuff[5];

ANEP_OPTBUFFER_SET_SIDE(optbuff, ANEP_OPTBUFFER_SEND);
```

As a feature to avoid changing the option buffer array in the kernel (ie: doing the `setsockopt`) if one want to temporary remove some options, it is possible to set the length of the option header field to 0, such a length is normally not possible because the minimal size of an option is 1 (ie: the size of the header).

For example in the following code, line 2, we reserve the memory for the option type 15; line 3, the option buffer structure will be used to send data; line 4, the option buffer is for the type 15; line 8, the socket is created; line 9 the structure is passed to the kernel; lines 10 and 11 the option header is filled.

```
 1  char                    option15[1024];
 2  struct anep_optbuffer   optbuff[] = { { NULL, 0, ANEP_OPTBUFFER_SEND },
 3                                        { 15, option15, sizeof(option15) } };
 4  struct anep_optionhdr  *opthdr15   = option15;
 5  int                     fd;
 6
 7  fd = socket(AF_INET6, SOCK_DGRAM, SOL_ANEP);
 8  setsockopt(fd, SOL_ANEP, ANEP_OPTBUFFER, optbuff, sizeof(optbuff));
 9  opthdr15->type    = htons(ANEP_OPT_PRIVATE | 15);
10  opthdr15->length  = htons(length_in_32bits_units);
```

The maximal size which can be transfered to the kernel for this option is `ANEP_RESTRICTED_OPTBUFFER_SIZE`.

2.3 `ioctl` function

```
#include <unistd.h>

int ioctl(int fd, int request, ... /* void *args */);
```

Returns 0 if OK, -1 on error

Two `ioctl` can be performed on an `ANEP` socket: `SIOCOUTQ` and `SIOCINQ`.

Category	*request*	Description	Datatype
socket	`SIOCOUTQ`	get how many bytes of the packet remain to be send.	`int`
	`SIOCINQ`	get how many bytes of data of the packet are available for reading.	`int`

Table 5: Ioctl requests for ANEP socket

2.4 Other functions

All the other functions `bind`, `connect`, `listen`, `recvfrom`, `sendto`, ... have the same kind of behaviors as for UDP or TCP, except that you don't have the notion of sender port number (and so you don't need to automatically or not bind to an ephemeral port before sending data).