IPv6 Flow Labels in Linux-2.2.

Alexey N. Kuznetsov
Institute for Nuclear Research, Moscow
kuznet@ms2.inr.ac.ru
April 11, 1999

Contents

1 Introduction.

Every IPv6 packet carries 28 bits of flow information. RFC2460 splits these bits to two fields: 8 bits of traffic class (or DS field, if you prefer this term) and 20 bits of flow label. Currently there exist no well-defined API to manage IPv6 flow information. In this document I describe an attempt to design the API for Linux-2.2 IPv6 stack.

The API must solve the following tasks:

1. To allow user to set traffic class bits.

2. To allow user to read traffic class bits of received packets. This feature is not so useful as the first one, however it will be necessary f.e. to implement ECN [RFC2481] for datagram oriented services or to implement receiver side of SRP or another end-to-end protocol using traffic class bits.

3. To assign flow labels to packets sent by user.

4. To get flow labels of received packets. I do not know any applications of this feature, but it is possible that receiver will want to use flow labels to distinguish sub-flows.

5. To allocate flow labels in the way, compliant to RFC2460. Namely:

   - Flow labels must be uniformly distributed (pseudo-)random numbers, so that any subset of 20 bits can be used as hash key.
   - Flows with coinciding source address and flow label must have identical destination address and not-fragmentable extensions headers (i.e. hop by hop options and all the headers up to and including routing header, if it is present.)

     NB. There is a hole in specs: some hop-by-hop options can be defined only on per-packet base (f.e. jumbo payload option). Essentially, it means that such options cannot present in packets with flow labels.

     NB. NB notes here and below reflect only my personal opinion, they should be read with smile or should not be read at all :-).
   - Flow labels have finite lifetime and source is not allowed to reuse flow label for another flow within the maximal lifetime has expired, so that intermediate nodes will be able to invalidate flow state before the label is taken over by another flow. Flow state, including lifetime, is propagated along datagram path by some application specific methods (f.e. in RSVP PATH messages or in some hop-by-hop option).

2 Sending/receiving flow information.

Discussion. It was proposed (Where? I do not remember any explicit statement) to solve the first four tasks using `sin6_flowinfo` field added to `struct sockaddr_in6` (see RFC2553).

> NB. This method is difficult to consider as reasonable, because it puts additional overhead to all the services, despite of only very small subset of them (none, to be more exact) really use it. It contradicts both to IETF spirit and the letter. Before RFC2553 one justification existed, IPv6 address alignment left 4 byte hole in `sockaddr_in6` in any case. Now it has no justification.

We have two problems with this method. The first one is common for all OSes: if `recvmsg()` initializes `sin6_flowinfo` to flow info of received packet, we loose one very important property of BSD socket API, namely, we are not allowed to use received address for reply directly and have to mangle it, even if we are not interested in flowinfo subtleties.

NB. RFC2553 adds new requirement: to clear sin6_flowinfo. Certainly, it is not solution but rather attempt to force applications to make unnecessary work. Well, as usually, one mistake in design is followed by attempts to patch the hole and more mistakes...

Another problem is Linux specific. Historically Linux IPv6 did not initialize sin6_flowinfo at all, so that, if kernel does not support flow labels, this field is not zero, but a random number. Some applications also did not take care about it.

NB. Following RFC2553 such applications can be considered as broken, but I still think that they are right: clearing all the address before filling known fields is robust but stupid solution. Useless wasting CPU cycles and memory bandwidth is not a good idea. Such patches are acceptable as temporary hacks, but not as standard of the future.

Implementation. By default Linux IPv6 does not read sin6_flowinfo field assuming that common applications are not obliged to initialize it and are permitted to consider it as pure alignment padding. In order to tell kernel that application is aware of this field, it is necessary to set socket option IPV6_FLOWINFO_SEND.

```
int on = 1;
setsockopt(sock, SOL_IPV6, IPV6_FLOWINFO_SEND,
           (void*)&on, sizeof(on));
```

Linux kernel never fills sin6_flowinfo field, when passing message to user space, though the kernels which support flow labels initialize it to zero. If user wants to get received flowinfo, he will set option IPV6_FLOWINFO and after this he will receive flowinfo as ancillary data object of type IPV6_FLOWINFO (cf. RFC2292).

```
int on = 1;
setsockopt(sock, SOL_IPV6, IPV6_FLOWINFO, (void*)&on, sizeof(on));
```

Flowinfo received and latched by a connected TCP socket also may be fetched with getsockopt() IPV6_PKTOPTIONS together with another optional information.

Besides that, in the spirit of RFC2292 the option IPV6_FLOWINFO may be used as alternative way to send flowinfo with sendmsg() or to latch it with IPV6_PKTOPTIONS.

Note about IPv6 options and destination address. If sin6_flowinfo does contain not zero flow label, destination address in sin6_addr and non-fragmentable extension headers are ignored. Instead, kernel uses the values cached at flow setup (see below). However, for connected sockets kernel prefers the values set at connection time.

Example. After setting socket option IPV6_FLOWINFO flowlabel and DS field are received as ancillary data object of type IPV6_FLOWINFO and level SOL_IPV6. In the cases when it is convenient to use recvfrom(2), it is possible to replace library variant with your own one, sort of:

```
#include <sys/socket.h>
#include <netinet/in6.h>

size_t recvfrom(int fd, char *buf, size_t len, int flags,
                struct sockaddr *addr, int *addrlen)
{
  size_t cc;
  char cbuf[128];
  struct cmsghdr *c;
  struct iovec iov = { buf, len };
  struct msghdr msg = { addr, *addrlen,
                        &iov,  1,
                        cbuf, sizeof(cbuf),
                        0 };

  cc = recvmsg(fd, &msg, flags);
  if (cc < 0)
    return cc;
  ((struct sockaddr_in6*)addr)->sin6_flowinfo = 0;
  *addrlen = msg.msg_namelen;
  for (c=CMSG_FIRSTHDR(&msg); c; c = CMSG_NEXTHDR(&msg, c)) {
    if (c->cmsg_level != SOL_IPV6 ||
      c->cmsg_type != IPV6_FLOWINFO)
        continue;
    ((struct sockaddr_in6*)addr)->sin6_flowinfo = *(__u32*)CMSG_DATA(c);
  }
  return cc;
}
```

3 Flow label management.

Discussion. Requirements of RFC2460 are pretty tough. Particularly, lifetimes longer than boot time require to store allocated labels at stable storage, so that the full implementation necessarily includes user space flow label manager. There are at least three different approaches:

1. **"Cooperative".** We could leave flow label allocation wholly to user space. When user needs label he requests manager directly. The approach is valid, but as any "cooperative" approach it suffers of security problems.

   NB. One idea is to disallow not privileged user to allocate flow labels, but instead to pass the socket to manager via `SCM_RIGHTS` control message, so that it will allocate label and assign it to socket itself. Hmm... the idea is interesting.

2. **"Indirect".** Kernel redirects requests to user level daemon and does not install label until the daemon acknowledged the request. The approach is the most promising, it is especially pleasant to recognize parallel with IPsec API [RFC2367,Craig]. Actually, it may share API with IPsec.

3. **"Stupid".** To allocate labels in kernel space. It is the simplest method, but it suffers of two serious flaws: the first, we cannot lease labels with lifetimes longer than boot time, the second, it is sensitive to DoS attacks. Kernel have to remember all the obsolete labels until their expiration and malicious user may fastly eat all the flow label space.

Certainly, I choose the most "stupid" method. It is the cheapest one for implementor (i.e. me), and taking into account that flow labels still have no serious applications it is not useful to work on more advanced API, especially, taking into account that eventually we will get it for no fee together with IPsec.

Implementation. Socket option `IPV6_FLOWLABEL_MGR` allows to request flow label manager to allocate new flow label, to reuse already allocated one or to delete old flow label. Its argument is `struct in6_flowlabel_req`:

```
struct in6_flowlabel_req
{
        struct in6_addr flr_dst;
        __u32           flr_label;
        __u8            flr_action;
        __u8            flr_share;
        __u16           flr_flags;
        __u16           flr_expires;
        __u16           flr_linger;
        __u32         __flr_reserved;
        /* Options in format of IPV6_PKTOPTIONS */
};
```

- `dst` is IPv6 destination address associated with the label.

- `label` is flow label value in network byte order. If it is zero, kernel will allocate new pseudo-random number. Otherwise, kernel will try to lease flow label ordered by user. In this case, it is user task to provide necessary flow label randomness.

- `action` is requested operation. Currently, only three operations are defined:

  ```
  #define IPV6_FL_A_GET   0   /* Get flow label */
  #define IPV6_FL_A_PUT   1   /* Release flow label */
  #define IPV6_FL_A_RENEW 2   /* Update expire time */
  ```

- `flags` are optional modifiers. Currently only `IPV6_FL_A_GET` has modifiers:

```
#define IPV6_FL_F_CREATE 1   /* Allowed to create new label */
#define IPV6_FL_F_EXCL   2   /* Do not create new label */
```

- `share` defines who is allowed to reuse the same flow label.

```
#define IPV6_FL_S_NONE    0   /* Not defined */
#define IPV6_FL_S_EXCL    1   /* Label is private */
#define IPV6_FL_S_PROCESS 2   /* May be reused by this process */
#define IPV6_FL_S_USER    3   /* May be reused by this user */
#define IPV6_FL_S_ANY     255 /* Anyone may reuse it */
```

- `linger` is time in seconds. After the last user releases flow label, it will not be reused with different destination and options at least during this time. If `share` is not `IPV6_FL_S_EXCL` the label still can be shared by another sockets. Current implementation does not allow unprivileged user to set linger longer than 60 sec.
- `expires` is time in seconds. Flow label will be kept at least for this time, but it will not be destroyed before user released it explicitly or closed all the sockets using it. Current implementation does not allow unprivileged user to set timeout longer than 60 sec. Proviledged applications MAY set longer lifetimes, but in this case they MUST save allocated labels at stable storage and restore them back after reboot before the first application allocates new flow.

This structure is followed by optional extension headers associated with this flow label in format of `IPV6_PKTOPTIONS`. Only `IPV6_HOPOPTS`, `IPV6_RTHDR` and, if `IPV6_RTHDR` presents, `IPV6_DSTOPTS` are allowed.

Example. The function `get_flow_label` allocates private flow label.

```
int get_flow_label(int fd, struct sockaddr_in6 *dst, __u32 fl)
{
        int on = 1;
        struct in6_flowlabel_req freq;

        memset(&freq, 0, sizeof(freq));
        freq.flr_label = htonl(fl);
        freq.flr_action = IPV6_FL_A_GET;
        freq.flr_flags = IPV6_FL_F_CREATE | IPV6_FL_F_EXCL;
        freq.flr_share = IPV6_FL_S_EXCL;
```

```
        memcpy(&freq.flr_dst, &dst->sin6_addr, 16);
        if (setsockopt(fd, SOL_IPV6, IPV6_FLOWLABEL_MGR,
                       &freq, sizeof(freq)) == -1) {
                perror ("can't lease flowlabel");
                return -1;
        }
        dst->sin6_flowinfo |= freq.flr_label;

        if (setsockopt(fd, SOL_IPV6, IPV6_FLOWINFO_SEND,
                       &on, sizeof(on)) == -1) {
                perror ("can't send flowinfo");

                freq.flr_action = IPV6_FL_A_PUT;
                setsockopt(fd, SOL_IPV6, IPV6_FLOWLABEL_MGR,
                           &freq, sizeof(freq));
                return -1;
        }
        return 0;
}
```

A bit more complicated example using routing header can be found in `ping6` utility (`iputils` package). Linux rsvpd backend contains an example of using operation `IPV6_FL_A_RENEW`.

Listing flow labels. List of currently allocated flow labels may be read from `/proc/net/ip6_flowlabel`.

```
Label S Owner Users Linger Expires Dst                              Opt
A1BE5 1 0     0     6      3       3ffe2400000000010a0020fffe71fb30 0
```

- `Label` is hexadecimal flow label value.
- `S` is sharing style.
- `Owner` is ID of creator, it is zero, pid or uid, depending on sharing style.
- `Users` is number of applications using the label now.
- `Linger` is `linger` of this label in seconds.
- `Expires` is time until expiration of the label in seconds. It may be negative, if the label is in use.
- `Dst` is IPv6 destination address.
- `Opt` is length of options, associated with the label. Option data are not accessible.

Flow labels and RSVP. RSVP daemon supports IPv6 flow labels without any modifications to standard ISI RAPI. Sender must allocate flow label, fill corresponding sender template and submit it to local rsvp daemon. rsvpd will check the label and start to announce it in PATH messages. Rsvpd on sender node will renew the flow label, so that it will not be reused before path state expires and all the intermediate routers and receiver purge flow state.

`rtap` utility is modified to parse flow labels. F.e. if user allocated flow label `0xA1234`, he may write:

```
RTAP> sender 3ffe:2400::1/FL0xA1234 <Tspec>
```

Receiver makes reservation with command:

```
RTAP> reserve ff 3ffe:2400::1/FL0xA1234 <Flowspec>
```