Internet Engineering Task Force
INTERNET-DRAFT
Expires: January 13, 2002

Jun-ichiro itojun Hagino
IIJ Research Laboratory
July 13, 2001

Socket API for IPv6 traffic class field
draft-itojun-ipv6-tclass-api-03.txt

Status of this Memo

Abstract

The draft outlines a socket API proposal for controlling the traffic class field in the IPv6 header. The API uses ancillary data stream to manipulate the traffic class field, following practice in the IPv6 advanced API.

The draft is, at this moment, written separately from the IPv6 basic/advanced API RFCs [Gilligan, 2000; Stevens, 1999] , as there can be many discussion items. The ultimate goal of the draft is to be a part of the IPv6 basic/advanced API.

1. Background

The IPv6 traffic class field is a 8bit field in the IPv6 header. The field serves just like the IPv4 type of service (TOS) field. There are two types of proposed use of the field: (1) topmost 6 bits for the differentiated services (diffserv) field [Nichols, 1998] , and (2) lowermost 2 bits for explicit congestion notification (ECN)

[Ramakrishnan, 1999] . Those two proposals plan to rewrite the field at intermediate routers.

There is a certain set of applications which need to manipulate and inspect the traffic class field. Here are some examples.

o ECN implementations outside of the kernel (like UDP ECN).

o A diffserv-aware application, which tries to mark low-priority traffic (such as non-important packets in a video traffic) on its own. In this case, the application does not need to inspect the field on outbound traffic.

o Debugging tools for differentiated services.

2. Inbound traffic

When an application is interested in inspecting the traffic class field on packets, the application should set the IPV6_RECVTCLASS socket option to 1:

```
/* enable */
const int on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_RECVTCLASS, &on, sizeof(on));
```

Subsequent incoming traffic will be accompanied with an ancillary data item that carries an unsigned octet value. The ancillary data item will be tagged with the level IPPROTO_IPV6 and type IPV6_TCLASS. An application can obatain the value of the traffic class field by the following operation, after the recvmsg(2) system call:

```
struct cmsghdr *cm;
u_int8_t tclass;
if (cm->cmsg_len == CMSG_LEN(sizeof(u_int8_t)) &&
    cm->cmsg_level == IPPROTO_IPV6 &&
    cm->cmsg_type == IPV6_TCLASS)
     tclass = *(u_int8_t *)CMSG_DATA(cm);
else
     tclass = 0x00; /* could not obtain traffic class value */
```

By setting the socket option to 0, the behavior is disabled:

```
/* disable */
const int off = 0;
setsockopt(fd, IPPROTO_IPV6, IPV6_RECVTCLASS, &off, sizeof(off));
```

For TCP sockets, an ancillary data item will be present only when the traffic class value is changed. See section 4.1 (TCP Implications) of [Stevens, 1999] for details.

3. Outbound traffic

To control the value of the traffic class field for a single packet transmission, you can use an ancillary data item, just like presented above, with a sendmsg(2) system call. The level of the ancillary data item must be IPPROTO_IPV6, and the type must be IPV6_TCLASS.

```
int s;  /* socket */
u_int8_t tclass;
struct sockaddr_in6 *dst;
struct msghdr m;
struct cmsghdr *cm;
struct iovec iov[2];
u_char cmsgbuf[256]; /* must be > CMSG_SPACE(sizeof(tclass)) */

/* set the data buffer to send */
memset(m, 0, sizeof(m));
memset(iov, 0, sizeof(iov));
m.msg_name = (caddr_t)dst;
m.msg_namelen = sizeof(dst);
iov[0].iov_base = buf;
iov[0].iov_len = len;
m.msg_iov = iov;
m.msg_iovlen = 1;

/* set ancillary data for the traffic class field */
memset(cmsgbuf, 0, sizeof(cmsgbuf));
cm = (struct cmsghdr *)cmsgbuf;
m.msg_control = cm;
m.msg_controllen = CMSG_SPACE(sizeof(tclass));
cm->cmsg_len = CMSG_LEN(sizeof(tclass));
cm->cmsg_level = IPPROTO_IPV6;
cm->cmsg_type = IPV6_TCLASS;
memcpy(CMSG_DATA(cm), &tclass, sizeof(tclass));

sendmsg(s, &m, 0);
```

If you want to put specific value to the traffic class field on multiple packets, you can use a "sticky" option:

```
u_int8_t tclass;
setsockopt(fd, IPPROTO_IPV6, IPV6_TCLASS, &tclass, sizeof(tclass));
```

4. Conflict resolution

There are two entities which may modify the traffic class field, in the kernel of the originating node: a kernel IPv6 code with diffserv marking enabled, and an ECN-capable TCP stack. Those entities may modify the traffic class field, even if an application tries to manipulate the

value. It may present a difficult constraint to the API. For outbound traffic, even if an application specifies the value to be put into the

traffic class field, in-kernel mechanism(s) may need to modify the field. The specified value may not be reflected into the packet on the wire (example: outbound processing in an ECN-capable TCP stack). For inbound traffic, even if the kernel presents the value on the field to the application, the value may not be the same as the value on the packet on the wire, due to manipulation in the kernel (example: traffic received by a diffserv egress node itself).

The following text proposes a suggested behavior. One of the goals of the suggestion is to allow applications to implement UDP ECN by themselves. The behavior may need more discussions:

Outbound traffic

If there is no conflict (for example, the TCP stack is not ECN-capable), the kernel should honor the value an application specified, and put the specified value into the traffic class field as is. If there is a conflict, the kernel should override the value specified by the application, for the part of the field (bits) the kernel is using. For example, if the kernel has an ECN-capable TCP stack but does not support diffserv, the kernel should override ECN bits only.

Inbound traffic

Kernel should present the traffic class value appeared on the wire as is to applications. Note that, in some cases, the kernel may want to alter specific bits in the field, before presenting the value to the userland. For example, if the kernel implements TCP ECN and would like to make it transparent to the user programs, the kernel may want to hide ECN bits

>From diffserv and ECN protocol specifications, the traffic class field may be rewritten by intermediate routers. So even if the sender specifies a value, the value may be altered before the packet reaches the final destination.

5. Issues

o Revise conflict resolution rule?

6. Security consideration

The API could be used for attempted theft of service. An attacker may try to inject packets, with some specific value in traffic class field, into a diffserv cloud. Refer to RFC2474 [Nichols, 1998] section 7.1 for detail. Note that the theft of diffserv service is possible even without the API.

References

Gilligan, 2000.
R. Gilligan, S. Thomson, J. Bound, and W. Stevens, "Basic Socket Interface Extensions for IPv6" in draft-ietf-ipngwg-rfc2553bis-00.txt (May 2000). work in progress material.

Stevens, 1999.
W. Richard Stevens, Matt Thomas, and Eric Nordmark, "Advanced Sockets API for IPv6" in draft-ietf-ipngwg-rfc2292bis-01.txt (October 1999). work in progress material.

Nichols, 1998.
K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers" in RFC2474 (December 1998). ftp://ftp.isi.edu/in-notes/rfc2474.txt.

Ramakrishnan, 1999.
K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP" in RFC2481 (January 1999). ftp://ftp.isi.edu/in-notes/rfc2481.txt.

Acknowledgements

The document was made possible by numerous invaluable comments from members of WIDE research group and KAME team. Here are people gave comments on the draft: Brian Carpenter (in no particular order).

Change history

00 -> 01
Improve the section on security consideration, based on comments from Brian Carpenter.

01 -> 02
Wording nits.

02 -> 03
Checked with WIDE diffserv/MPLS guys.

Author's address

Jun-ichiro itojun HAGINO
Research Laboratory, Internet Initiative Japan Inc.
Takebashi Yasuda Bldg.,
3-13 Kanda Nishiki-cho,
Chiyoda-ku, Tokyo 101-0054, JAPAN
Tel: +81-3-5259-6350
Fax: +81-3-5259-6351
Email: itojun@iijlab.net