Untangling DSCP, TOS and ECN bits in the kernel

Guillaume Nault

Red Hat

gnault@redhat.com

Linux Plumbers Conference

September 24, 2021

Why this talk?

- TOS handling is inconsistent in the kernel.
- Regressions introduced regularly.
- Several corner cases still to be fixed.
- New features proposed upstream with bad or dangerous implementation.

Evolution of the TOS field for IPv4

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Type of Service field is 1 byte long. Its definition has varied over time:

RFC 791 (1981): `pppTTTrr`
RFC 1122 (1989): `pppTTTTT`
RFC 1349 (1992): `pppTTTTr`
RFC 2474 (1998): `TTTTTTrr` (introduction of DSCP)
RFC 3168 (2001): `TTTTTTee` (introduction of ECN)

`p`: precedence bits
`T`: bits usable for encoding the Type of Service
`r`: reserved bits
`e`: ECN bits

Evolution of the TOS field for IPv6

For IPv6 too the definition has changed:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Prio. |                   Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RFC 1883 (1995): `TTTT`

RFC 2460 (1998): `TTTTTTTT`

RFC 2474 (1998): `TTTTTTrr` (introduction of DSCP)

RFC 3168 (2001): `TTTTTTee` (introduction of ECN)

RFC 8200 (2017): `TTTTTTee` (follows RFC 2474 and RFC 3168)

`T`: bits usable for encoding the Type of Service

`r`: reserved bits

`e`: ECN bits

Linux kernel implementation

The situation is a bit messy...

- IPv4 ignores ECN bits when matching TOS (apart from some corner cases that need to be fixed).
- IPv6 takes ECN bits into account when matching TOS (so ECT(0) and ECT(1) packets might be treated differently).
- Most IPv4 FIB lookups don't use the high order bits of the TOS (core routing, ip rules) but not all (nft_fib_ipv4).
- IPv6 takes all high order bits into account when matching TOS.
- The configuration paths accepts unusable TOS values (so one can configure a TOS that actually can't ever match).

TOS macros used by IPv4

TOS is generally stored as __u8 and includes the ECN bits. IPv4 often uses the following macros when handling TOS:

- `RT_TOS()` : masks the old precedence bits and the MBZ one: `000xxxx0` (RFC 1349 style).
- `IPTOS_RT_MASK` : like `RT_TOS` but also masks both ECN bits: `000xxx00` (RFC 791 style).

TOS macros used by IPv6

None... but `RT_TOS()` starts spreading into IPv6 code, where it doesn't make sense :(.

Practical consequences

Past problems:

- `ip route get` returning a different route than what real packets would follow.
- Regression (behaviour changes) in VXLAN due to unclear TOS semantic.
- Wrong source address selection.

Current problems:

- Inconsistent handling of the old preference bits.
- Different behaviour between IPv4 and IPv6 (but people should be used to that :-().
- Risky patches posted upstream to make the high order bits usable (blindly modifying the IPv4 TOS macros).

IPv4: edge cases with ip route

- TOS covering ECN bits are accepted, but no packet will ever match:

```
# ip route add 192.0.2.0/24 tos 1 dev eth0
# ping -Q 1 192.0.2.1
ping:  connect:  Network is unreachable
```

- Good old RFC 791 TOS work, but also match packets with high order DSCP bits set:

```
# ip route add 192.0.2.0/24 tos 4 dev eth0
# ping -Q 0xe4 192.0.2.1
[...]
29 packets transmitted, 29 received, 0% packet loss
```

- TOS covering high order DSCP bits are accepted, but no packet will ever match:

```
# ip route add 192.0.2.0/24 tos 0xe4 dev eth0
# ping -Q 0xe4 192.0.2.1
ping:  connect:  Network is unreachable
```

IPv4: edge cases with `ip rule`

[Examples assume `ip route add 192.0.2.0/24 table 100 dev eth0`]

- TOS covering ECN bit 0 are rejected:

  ```
  # ip rule add tos 1 table 100
  Error:  Invalid tos.
  ```

- TOS covering ECN bit 1 are accepted, but no packet will ever match:

  ```
  # ip rule add tos 2 table 100
  # ping -Q 2 192.0.2.1
  ping:  connect:  Network is unreachable
  ```

- Good old RFC 791 TOS work, but also match packet with high order DSCP bits set:

  ```
  # ip rule add tos 4 table 100
  # ping -Q 0xe4 192.0.2.1
  [...]
  26 packets transmitted, 26 received, 0% packet loss
  ```

- TOS covering high order DSCP bits are rejected:

  ```
  # ip rule add tos 0xe4 table 100
  Error:  Invalid tos.
  ```

What about IPv6?

`ip route` : `tos` parameter ignored for IPv6.

`ip rule` : any TOS accepted (between 0 to 0xff), no mask applied when matching packets: what you type is really what you get.
Fine, but do we really want to let the admin mess with ECN?

What can we do?

Obvious steps:

- Fix remaining bugs:
  - IPv6: remove code that masks high order DSCP bits (`RT_TOS`).
  - IPv4: mask ECN bits where this is missing.
- Remove `IPTOS_TOS_MASK` and derived macros (`RT_TOS()`, `IPTOS_TOS()`): they generally don't make sense.

Long term:

- Define the expected behaviours:
  → Should we consider the result of any of the previous `ip` commands as bug?
- Rework internal code to avoid introducing more bugs or inconsistent behaviours.

Possible long-term improvements

- Option 1: define a `dscp_t` type:
  - Ensure ECN bits are cleared.
  - Sparse could warn about incorrect uses.

or

- Option 2: add a bit-mask for TOS configuration:
  - TOS values (as read from packets) would remain 8-bits integers and contain the ECN bits.
  - TOS configuration would always have a value *and* a mask.
  - TOS mask might allow covering the ECN bits (for compatibility with current IPv6 behaviour).

Option 1: define a dscp_t type

Something like:

```
typedef u8 __bitwise dscp_t;

#define INET_DSCP_MASK 0xfc

static inline dscp_t dscp_from_u8(__u8 tos)
{
    return (__force dscp_t)(tos & INET_DSCP_MASK);
}

static inline __u8 dscp_to_u8(dscp_t dscp)
{
    return (__force __u8)dscp;
}
```

Option 1: drawbacks of the `dscp_t` type approach

- Code churn (lots of code and structures to modify).
- Sparse warnings can go unnoticed (maybe patchwork can help).
- For IPv4, should the mask cover all DSCP bits or just the original 3 TOS bits?
- What about IPv6? Clear the ECN bits or not? If not, how to handle code that works on both IPv4 and IPv6?

Option 2: Add a bit-mask for TOS configuration

- New type for storing TOS configuration (TOS value + mask):

```
typedef u16 __bitwise tos_cfg_t;
```

- Allow optional TOS mask attribute every time we configure a TOS:

```
ip rule add tos 0xf4/0xfc table 100
```

- Allows using the whole DSCP range.
- Possible different default TOS mask depending on context and expected behaviour.

Option 2: drawbacks of the bit-mask approach

- Not as mechanical as option 1.
- Edge cases:
  - Packets may match different configured TOS:

    ```
        ip route add 192.168.0.2/24 tos 0x10/0x30 ...
        ip route add 192.168.0.2/24 tos 0x40/0xc0 ...
    ```

    Which route should be selected for a packet with TOS 0x50? First match wins? Use arbitrary rule (like compare TOS masks as integer and select the biggest one)?
  - Null TOS with non-null mask, like `0x00/0x04` (or `0x00/$default_mask`)? Wild card or not?
- Is it worth the pain (is that really going to be useful to anyone)?

Conclusion

What we would get in an ideal world:

- Full DSCP support for IPV4.
- TOS shouldn't break ECN.
- Same behaviour for IPv4 and IPv6.

What we can realistically do:

- Fix existing bugs (IPv4 not masking ECN bits, IPv6 masking DSCP bits).
- Remove uses of `IPTOS_TOS_MASK` and derived macros like `RT_TOS()` so that people stop copy/pasting them.
- Clearly define the expected effect of TOS.
- Rework existing code so that we won't re-introduce TOS bugs:

  Option 1 : with Sparse (`dscp_t`).

  Option 2 : with a TOS mask (`tos_cfg_t`).

Discussions

Questions?

Comments?