## Classes

"Class" is a complete routing table in common sense.
I.e. it is tree of nodes (destination prefix, tos, metric)
with attached information: gateway, device etc.
This tree is looked up as specified in RFC1812 5.2.4.3

1. Basic match
2. Longest match
3. Weak TOS.
4. Metric. (should not be in kernel space, but they are)
5. Additional pruning rules. (not in kernel space).

We have two special type of nodes:
REJECT - abort route lookup and return an error value.
THROW  - abort route lookup in this class.

Currently the number of classes is limited to 255
(0 is reserved for "not specified class")

Three classes are builtin:

RT_CLASS_LOCAL=255 - local interface addresses,
broadcasts, nat addresses.

RT_CLASS_MAIN=254  - all normal routes are put there
by default.

RT_CLASS_DEFAULT=253 - if ip_fib_model==1, then
normal default routes are put there, if ip_fib_model==2
all gateway routes are put there.

## Rules

Rule is a record of (src prefix, src interface, tos, dst prefix)
with attached information.

Rule types:
RTP_ROUTE - lookup in attached class
RTP_NAT   - lookup in attached class and if a match is found,
            translate packet source address.
RTP_MASQUERADE - lookup in attached class and if a match is found,
            masquerade packet as sourced by us.
RTP_DROP   - silently drop the packet.
RTP_REJECT - drop the packet and send ICMP NET UNREACHABLE.
RTP_PROHIBIT - drop the packet and send ICMP COMM. ADM. PROHIBITED.

Rule flags:
RTRF_LOG - log route creations.
RTRF_VALVE - One way route (used with masquerading)

Default setup:

```
root@amber:/pub/ip-routing # iproute -r
```

```
Kernel routing policy rules
Pref Source            Destination       TOS Iface   Cl
   0 default           default           00  *       255
 254 default           default           00  *       254
 255 default           default           00  *       253
```

## Lookup algorithm

We scan rules list, and if a rule is matched, apply it.
If a route is found, return it.
If it is not found or a THROW node was matched, continue
to scan rules.

## Applications

1. Just ignore classes. All the routes are put into MAIN class (and/or into DEFAULT class).

   HOWTO:
   ```
   iproute add PREFIX [ tos TOS ] [ gw GW ] [ dev DEV ]
   [ metric METRIC ] [ reject ] ... (look at iproute utility)
   ```

   or use route utility from current net-tools.

2. Opposite case. Just forget all that you know about routing tables. Every rule is supplied with its own gateway, device info. record. This approach is not appropriate for automated route maintenance, but it is ideal for manual configuration.

   HOWTO:
   ```
   iproute addrule [ from PREFIX ] [ to PREFIX ] [ tos TOS ]
   [ dev INPUTDEV] [ pref PREFERENCE ] route [ gw GATEWAY ]
   [ dev OUTDEV ] .....
   ```

   Warning: As of now the size of the routing table in this approach is limited to 256. If someone likes this model, I'll relax this limitation.

3. OSPF classes (see RFC1583, RFC1812 E.3.3)
   Very clean, stable and robust algorithm for OSPF routing domains. Unfortunately, it is not widely used in the Internet.

   Proposed setup:
   255 local addresses
   254 interface routes
   253 ASE routes with external metric
   252 ASE routes with internal metric
   251 inter-area routes
   250 intra-area routes for 1st area
   249 intra-area routes for 2nd area
   etc.

   Rules:
   ```
   iproute addrule class 253
   iproute addrule class 252
   ```

```
        iproute addrule class 251
        iproute addrule to a-prefix-for-1st-area class 250
        iproute addrule to another-prefix-for-1st-area class 250
        ...
        iproute addrule to a-prefix-for-2nd-area class 249
        ...

        Area classes must be terminated with reject record.
        iproute add default reject class 250
        iproute add default reject class 249
        ...
```

4.      The Variant Router Requirements Algorithm (RFC1812 E.3.2)
        Create 16 classes for different TOS values.
        It is a funny, but pretty useless algorithm.
        I listed it just to show the power of new routing code.

5.      All the variety of combinations......

## GATED

Gated does not understand classes, but it will work happily in MAIN+DEFAULT. All policy routes can be set and maintained manually.

## IMPORTANT NOTE

route.c has a compilation time switch CONFIG_IP_LOCAL_RT_POLICY. If it is set, locally originated packets are routed using all the policy list. This is not very convenient and pretty ambiguous when used with NAT and masquerading. I set it to FALSE by default.

Alexey Kuznetov
kuznet@ms2.inr.ac.ru