Tunnels over IP in Linux-2.2

Alexey N. Kuznetsov
Institute for Nuclear Research, Moscow
kuznet@ms2.inr.ac.ru
March 17, 1999

Contents

1 Instead of introduction: micro-FAQ.

- Q: In linux-2.0.36 I used:

  ```
  ifconfig tunl1 10.0.0.1 pointopoint 193.233.7.65
  ```

  to create tunnel. It does not work in 2.2.0!

  A: You are right, it does not work. The command written above is split to two commands.

  ```
  ip tunnel add MY-TUNNEL mode ipip remote 193.233.7.65
  ```

  will create tunnel device with name `MY-TUNNEL`. Now you may configure it with:

```
ifconfig MY-TUNNEL 10.0.0.1
```

Certainly, if you prefer name `tunl1` to `MY-TUNNEL`, you still may use it.

- Q: In linux-2.0.36 I used:

  ```
  ifconfig tunl0 10.0.0.1
  route add -net 10.0.0.0 gw 193.233.7.65 dev tunl0
  ```

  to tunnel net 10.0.0.0 via router 193.233.7.65. It does not work in 2.2.0! Moreover, `route` prints a funny error sort of "network unreachable" and after this I found a strange direct route to 10.0.0.0 via `tunl0` in routing table.

  A: Yes, in 2.2 the rule that *normal* gateway must reside on directly connected network has not any exceptions. You may tell kernel, that this particular route is *abnormal*:

  ```
  ifconfig tunl0 10.0.0.1 netmask 255.255.255.255
  ip route add 10.0.0.0/8 via 193.233.7.65 dev tunl0 onlink
  ```

  Note keyword `onlink`, it is the magic key that orders kernel not to check for consistency of gateway address. Probably, after this explanation you have already guessed another method to cheat kernel:

  ```
  ifconfig tunl0 10.0.0.1 netmask 255.255.255.255
  route add -host 193.233.7.65 dev tunl0
  route add -net 10.0.0.0 netmask 255.0.0.0 gw 193.233.7.65
  route del -host 193.233.7.65 dev tunl0
  ```

  Well, if you like such tricks, nobody may prohibit you to use them. Only do not forget that between `route add` and `route del` host 193.233.7.65 is unreachable.

- Q: In 2.0.36 I used to load `tunnel` device module and `ipip` module. I cannot find any `tunnel` in 2.2!

  A: Linux-2.2 has single module `ipip` for both directions of tunneling and for all IPIP tunnel devices.

- Q: `traceroute` does not work over tunnel! Well, stop... It works, only skips some number of hops.

  A: Yes. By default tunnel driver copies `ttl` value from inner packet to outer one. It means that path traversed by tunneled packets to another endpoint is not hidden. If you dislike this, or if you are going to use some routing protocol expecting that packets with ttl 1 will reach peering host (f.e. RIP, OSPF or EBGP) and you are not afraid of tunnel loops, you may append option `ttl 64`, when creating tunnel with `ip tunnel add`.

- Q: ... Well, list of things, which 2.0 was able to do finishes.

Summary of differences between 2.2 and 2.0.

- **In 2.0** you could compile tunnel device into kernel and got set of 4 devices `tunl0` ... `tunl3` or, alternatively, compile it as module and load new module for each new tunnel. Also, module `ipip` was necessary to receive tunneled packets.

  2.2 has *one* module `ipip`. Loading it you get base tunnel device `tunl0` and another tunnels may be created with command `ip tunnel add`. These new devices may have arbitrary names.

- **In 2.0** you set remote tunnel endpoint address with the command `ifconfig` ... `pointopoint A`.

  In 2.2 this command has the same semantics on all the interfaces, namely it sets not tunnel endpoint, but address of peering host, which is directly reachable via this tunnel, rather than via Internet. Actual tunnel endpoint address `A` should be set with `ip tunnel add ... remote A`.

- **In 2.0** you create tunnel routes with the command:

  ```
  route add -net 10.0.0.0 gw A dev tunl0
  ```

  2.2 interprets this command equally for all device kinds and gateway is required to be directly reachable via this tunnel, rather than via Internet. You still may use `ip route add ... onlink` to override this behaviour.

2 Tunnel setup: basics

Standard Linux-2.2 kernel supports three flavor of tunnels, listed in the following table:

Mode	Description	Base device
ipip	IP over IP	tunl0
sit	IPv6 over IP	sit0
gre	ANY over GRE over IP	gre0

All the kinds of tunnels are created with one command:

```
ip tunnel add <NAME> mode <MODE> [ local <S> ] [ remote <D> ]
```

This command creates new tunnel device with name `<NAME>`. The `<NAME>` is an arbitrary string. Particularly, it may be even `eth0`. The rest of parameters set different tunnel characteristics.

- `mode <MODE>` sets tunnel mode. Three modes are available now `ipip`, `sit` and `gre`.

- `remote <D>` sets remote endpoint of the tunnel to IP address `<D>`.
- `local <S>` sets fixed local address for tunneled packets. It must be an address on another interface of this host.

Both `remote` and `local` may be omitted. In this case we say that they are zero or wildcard. Two tunnels of one mode cannot have the same `remote` and `local`. Particularly it means that base device or fallback tunnel cannot be replicated.[1]

Tunnels are divided to two classes: **pointopoint** tunnels, which have some not wildcard `remote` address and deliver all the packets to this destination, and **NBMA** (i.e. Non-Broadcast Multi-Access) tunnels, which have no `remote`. Particularly, base devices (f.e. `tunl0`) are NBMA, because they have neither `remote` nor `local` addresses.

After tunnel device is created you should configure it as you did it with another devices. Certainly, the configuration of tunnels has some features related to the fact that they work over existing Internet routing infrastructure and simultaneously create new virtual links, which changes this infrastructure. The danger that not enough careful tunnel setup will result in formation of tunnel loops, collapse of routing or flooding network with exponentially growing number of tunneled fragments is very real.

Protocol setup on pointopoint tunnels does not differ of configuration of another devices. You should set a protocol address with `ifconfig` and add routes with `route` utility.

NBMA tunnels are different. To route something via NBMA tunnel you have to explain to driver, where it should deliver packets to. The only way to make it is to create special routes with gateway address pointing to desired endpoint. F.e.

```
ip route add 10.0.0.0/24 via <A> dev tunl0 onlink
```

It is important to use option `onlink`, otherwise kernel will refuse request to create route via gateway not directly reachable over device `tunl0`. With IPv6 the situation is much simpler: when you start device `sit0`, it automatically configures itself with all IPv4 addresses mapped to IPv6 space, so that all IPv4 Internet is *really reachable* via `sit0`! Excellent, the command

```
ip route add 3FFE::/16 via ::193.233.7.65 dev sit0
```

will route `3FFE::/16` via `sit0`, sending all the packets destined to this prefix to 193.233.7.65.

[1] This restriction is relaxed for keyed GRE tunnels.

3 Tunnel setup: options

Command `ip tunnel add` has several additional options.

- `ttl N` — set fixed TTL `N` on tunneled packets. `N` is number in the range 1–255. 0 is special value, meaning that packets inherit TTL value. Default value is: `inherit`.
- `tos T` — set fixed tos `T` on tunneled packets. Default value is: `inherit`.
- `dev DEV` — bind tunnel to device `DEV`, so that tunneled packets will be routed only via this device and will not be able to escape to another device, when route to endpoint changes.
- `nopmtudisc` — disable Path MTU Discovery on this tunnel. It is enabled by default. Note that fixed ttl is incompatible with this option: tunnels with fixed ttl always make pmtu discovery.

`ipip` and `sit` tunnels have no more options. `gre` tunnels are more complicated:

- `key K` — use keyed GRE with key `K`. `K` is either number or IP address-like dotted quad.
- `csum` — checksum tunneled packets.
- `seq` — serialize packets.

  NB. I think this option does not work. At least, I did not test it, did not debug it and even do not understand, how it is supposed to work and for what purpose Cisco planned to use it.

Actually, these GRE options can be set separately for input and output directions by prefixing corresponding keywords with letter `i` or `o`. F.e. `icsum` orders to accept only packets with correct checksum and `ocsum` means, that our host will calculate and send checksum.

Command `ip tunnel add` is not the only operation, which can be made with tunnels. Certainly, you may get short help page with:

```
ip tunnel help
```

Besides that, you may view list of installed tunnels with the help of command:

```
ip tunnel ls
```

Also you may look at statistics:

```
ip -s tunnel ls Cisco
```

where `Cisco` is name of tunnel device. Command

```
ip tunnel del Cisco
```

destroys tunnel `Cisco`. And, finally,

```
ip tunnel change Cisco mode sit local ME remote HE ttl 32
```

changes its parameters.

4 Differences 2.2 and 2.0 tunnels revisited.

Now we can discuss more subtle differences between tunneling in 2.0 and 2.2.

- In 2.0 all tunneled packets were received promiscuously as soon as you loaded module `ipip`. 2.2 tries to select the best tunnel device and packet looks as received on this. F.e. if host received `ipip` packet from host `D` destined to our local address `S`, kernel searches for matching tunnels in order:

  1. `remote` is `D` and `local` is `S`
  2. `remote` is `D` and `local` is wildcard
  3. `remote` is wildcard and `local` is `S`
  4. `tunl0`

  If tunnel exists, but it is not in `UP` state, the tunnel is ignored. Note, that if `tunl0` is `UP` it receives all the IPIP packets, not acknowledged by more specific tunnels. Be careful, it means that without carefully installed firewall rules anyone on the Internet may inject to your network any packets with source addresses indistinguishable from local ones. It is not so bad idea to design tunnels in the way enforcing maximal route symmetry and to enable reversed path filter (`rp_filter` sysctl option) on tunnel devices.

- In 2.2 you can monitor and debug tunnels with `tcpdump`. F.e. `tcpdump -i Cisco -nvv` will dump packets, which kernel output, via tunnel `Cisco` and the packets received on it from kernel viewpoint.

5 Linux and Cisco IOS tunnels.

Among another tunnels Cisco IOS supports IPIP and GRE. Essentially, Cisco setup is subset of options, available for Linux. Let us consider the simplest example:

```
interface Tunnel0
 tunnel mode gre ip
 tunnel source 10.10.14.1
 tunnel destination 10.10.13.2
```

This command set translates to:

```
ip tunnel add Tunnel0 \
    mode gre \
    local 10.10.14.1 \
    remote 10.10.13.2
```

Any questions? No questions.

6 Interaction IPIP tunnels and DVMRP.

DVMRP exploits IPIP tunnels to route multicasts via Internet. `mrouted` creates IPIP tunnels listed in its configuration file automatically. From kernel and user viewpoints there are no differences between tunnels, created in this way, and tunnels created by `ip tunnel`. I.e. if `mrouted` created some tunnel, it may be used to route unicast packets, provided appropriate routes are added. And vice versa, if administrator has already created a tunnel, it will be reused by `mrouted`, if it requests DVMRP tunnel with the same local and remote addresses.

Do not wonder, if your manually configured tunnel is destroyed, when mrouted exits.

7 Broadcast GRE "tunnels".

It is possible to set `remote` for GRE tunnel to a multicast address. Such tunnel becomes **broadcast** tunnel (though word tunnel is not quite appropriate in this case, it is rather virtual network).

```
ip tunnel add Universe local 193.233.7.65 \
                       remote 224.66.66.66 ttl 16
ip addr add 10.0.0.1/16 dev Universe
ip link set Universe up
```

This tunnel is true broadcast network and broadcast packets are sent to multicast group 224.66.66.66. By default such tunnel starts to resolve both IP and IPv6 addresses via ARP/NDISC, so that if multicast routing is supported in surrounding network, all GRE nodes will find one another automatically and will form virtual Ethernet-like broadcast network. If multicast routing does not work, it is unpleasant but not fatal flaw. The tunnel becomes NBMA rather than broadcast network. You may disable dynamic ARPing by:

```
echo 0 > /proc/sys/net/ipv4/neigh/Universe/mcast_solicit
```

and to add required information to ARP tables manually:

```
ip neigh add 10.0.0.2 lladdr 128.6.190.2 dev Universe nud permanent
```

In this case packets sent to 10.0.0.2 will be encapsulated in GRE and sent to 128.6.190.2. It is possible to facilitate address resolution using methods typical for another NBMA networks f.e. to start user level `arpd` daemon, which will maintain database of hosts attached to GRE virtual network or ask for information dedicated ARP or NHRP server.

Actually, such setup is the most natural for tunneling, it is really flexible, scalable and easily managable, so that it is strongly recommended to be used with GRE tunnels instead of ugly hack with NBMA mode and `onlink` modifier. Unfortunately, by historical reasons broadcast mode is not supported by IPIP tunnels, but this probably will change in future.

8 Traffic control issues.

Tunnels are devices, hence all the power of Linux traffic control applies to them. The simplest (and the most useful in practice) example is limiting tunnel bandwidth. The following command:

```
tc qdisc add dev tunl0 root tbf \
    rate 128Kbit burst 4K limit 10K
```

will limit tunneled traffic to 128Kbit with maximal burst size of 4K and queuing not more than 10K.

However, you should remember, that tunnels are *virtual* devices implemented in software and true queue management is impossible for them just because they have no queues. Instead, it is better to create classes on real physical interfaces and to map tunneled packets to them. In general case of dynamic routing you should create such classes on all outgoing interfaces, or, alternatively, to use option `dev DEV` to bind tunnel to a fixed physical device. In the last case packets will be routed only via specified device and you need to setup corresponding classes only on it. Though you have to pay for this convenience, if routing will change, your tunnel will fail.

Suppose that CBQ class `1:ABC` has been created on device `eth0` specially for tunnel `Cisco` with endpoints `S` and `D`. Now you can select IPIP packets with addresses `S` and `D` with some classifier and map them to class `1:ABC`. F.e. it is easy to make with `rsvp` classifier:

```
tc filter add dev eth0 pref 100 proto ip rsvp \
    session D ipproto ipip filter S \
    classid 1:ABC
```

If you want to make more detailed classification of sub-flows transmitted via tunnel, you can build CBQ subtree, rooted at `1:ABC` and attach to subroot set of rules parsing IPIP packets more deeply.