Moving from ROSv6 to v7 with examples

Routing Tables

By default, all routes are added to the "main" routing table as it was before. From a configuration point of view, the biggest differences are routing table limit increase, routing table monitoring differences, and how routes are added to specific routing tables (see next example)
v7 introduces a new menu /routing route, which shows all address family routes as well as all filtered routes with all possible route attributes. `/ip route` a nd `/ipv6 route` menus are used to add static routes and for simplicity show only basic route attributes.

For more in-depth information on routing see this article (IP Routing).

Another new change is that most common route print requests are processed by the routing process which significantly improves the speed compared to v6.

Use of Routing Tables and Policy Routing

The main difference from v6 is that the routing table must be added to the `/routing table` menu before actually referencing it anywhere in the configuration. And **fib** parameter should be specified if the routing table is intended to push routes to the FIB.
The routing rule configuration is the same except for the menu location (instead of `/ip route rule`, now it is `/routing rule`).

Let's consider a basic example where we want to resolve 8.8.8.8 only in the routing table named myTable to the gateway 172.16.1.1:

```
/routing table add name=myTable fib
/routing rule add dst-address=8.8.8.8 action=lookup-only-in-table table=myTable
/ip route add dst-address=8.8.8.8 gateway=172.16.1.1@main routing-table=myTable
```

Instead of routing rules, you could use mangle to mark packets with routing-mark, the same way as it was in ROSv6.

OSPF Configuration

OSPFv3 and OSPFv2 are now merged into one single menu `/routing ospf`. At the time of writing this article, there are no default instances and areas. To start both OSPFv2 and OSPF v3 instances, first, you need to create an instance for each and then add an area to the instance.

```
/routing ospf instance
add name=v2inst version=2 router-id=1.2.3.4
add name=v3inst version=3 router-id=1.2.3.4
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0 instance=v2inst
add name=backbone_v3 area-id=0.0.0.0 instance=v3inst
```

At this point, you are ready to start OSPF on the network interface. In the case of IPv6, you add either interface on which you want to run OSPF (the same as ROSv6) or the IPv6 network. In the second case, OSPF will automatically detect the interface. Here are some interface configuration examples:

```
/routing ospf interface-template
add network=192.168.0.0/24 area=backbone_v2
add network=2001:db8::/64 area=backbone_v3
add network=ether1 area=backbone_v3
```

ROSv7 uses templates to match the interface against the template and apply configuration from the matched template. OSPF menus `interface` and `neighbor` contains read-only entries purely for status monitoring.

~~All route distribution control is now done purely with routing filter select, no more redistribution knobs in the instance~~ (Since the v7.1beta7 redistribution knob is back, you still need to use routing filters to set route costs and type if necessary). This gives greater flexibility on what routes from which protocols you want to redistribute.
For example, let's say you want to redistribute only static IPv4 routes from the 192.168.0.0/16 network range.

```
/routing ospf instance
set backbone_v2 out-filter-chain=ospf_out redistribute=static
```

```
/routing filter rule add chain=ospf_out rule="if (dst in 192.168.0.0/16) {accept}"
```

⚠ The default action of the routing filter chain is "drop"

BGP Configuration

There is a complete redesign of the BGP configuration compared to ROSv6. The first biggest difference is that there is no more **`instance`** and **`peer`** configuration menus. Instead, we have **`connection`**, **`template`** and **`session`** menus.
The reason for such a structure is to strictly split parameters that are responsible for connection and parameters that are BGP protocol specific.

Let's start with the Template. It contains all BGP protocol-related configuration options. It can be used as a template for dynamic peers and apply a similar config to a group of peers. Note that this is not the same as peer groups on Cisco devices, where the group is more than just a common configuration.

By default, there is a default template that requires you to set your own AS.

```
/routing/bgp/template set default as=65533
```

⚠ Starting from v7.1beta4 template parameters are exposed in the "connection" configuration. This means that the template is not mandatory anymore, allowing for an easier basic BGP connection setup, similar to what it was in ROSv6.

Most of the parameters are similar to ROSv6 except that some are grouped in the output and input section making the config more readable and easier to understand whether the option is applied on input or output. If you are familiar with CapsMan then the syntax is the same, for example, to specify the output selection chain you set `output.filter-chain=myBgpChain`.

You can even inherit template parameters from another template, for example:

```
/routing/bgp/template
add name=myAsTemplate as=65500 output.filter-chain=myAsFilter
set default template=myAsTemplate
```

Another important aspect of the new routing configuration is the global Router ID, which sets router-id and group peers in one instance. RouterOS adds a default ID which picks instance-id from any interface's highest IP. The default BGP template by default is set to use the "default" ID.
If for any reason you need to tweak or add new instances it can be done in `/routing id` menu.

Very interesting parameters are **`input.affinity`** and **`output.affinity`**, they allow control in which process input and output of active session will be processed:

- **alone** - input and output of each session are processed in its own process, most likely the best option when there are a lot of cores and a lot of peers
- **afi, instance, vrf, remote-as** - try to run input/output of new session in process with similar parameters
- **main** - run input/output in the main process (could potentially increase performance on single-core even possibly on multicore devices with small amount of cores)
- **input** - run output in the same process as input (can be set only for output affinity)

Now that we have parameters set for the template we can add BGP connections. A minimal set of parameters are `remote.address`, `template, connect`, `listen` and `local.role`

Connect and listen to parameters specify whether peers will try to connect and listen to a remote address or just connect or just listen. It is possible that in setups where peer uses the multi-hop connection `local.address` must be configured too (similar as it was with `update-source` in ROSv6).

> ⚠ It is not mandatory to specify a remote AS number. ROS v7 can determine remote ASN from an open message. You should specify the remote AS only when you want to accept a connection from that specific AS.

Peer role is now a mandatory parameter, for basic setups, you can just use ibgp, ebgp (more information on available roles can be found in the corresponding RFC draft https://datatracker.ietf.org/doc/draft-ietf-idr-bgp-open-policy/?include_text=1), keep in mind that at the moment capabilities, communities, and filtering described in the draft is not implemented.

Very basic iBGP set up to listen on the whole local network for connections:

```
/routing/bgp/connection
add remote.address=10.155.101.0/24 listen=yes template=default local.role=ibgp
```

Now you can monitor the status of all connected and disconnected peers from `/routing bgp session` menu.

Other great debugging information on all routing processes can be monitored from `/routing stats` menu

```
[admin@v7_ccr_bgp] /routing/stats/process> print interval=1
Columns: TASKS, PRIVATE-MEM-BLOCKS, SHARED-MEM-BLOCKS, PSS, RSS, VMS, RETIRED, ID, PID, RPID, PROCESS-TIME,
KERNEL-TIME, CUR-B>
# TASKS PRIVATE-M SHARED-ME PSS RSS VMS RET ID PID R PROCESS-TI KERN>
0 routing tables 12.2MiB 20.0MiB 18.7MiB 42.2MiB 83.4MiB 8 main 319 0 19s750ms 8s50>
rib >
connected networks >
1 fib 512.0KiB 0 7.4MiB 30.9MiB 83.4MiB fib 384 1 5s160ms 22s5>
2 ospf 1024.0KiB 1024.0KiB 5.9MiB 25.9MiB 83.4MiB 382 ospf 388 1 1m42s170ms 1m31>
connected networks >
3 fantasy 512.0KiB 0 2061.0KiB 5.9MiB 83.4MiB fantasy 389 1 1s410ms 870m>
4 configuration and reporting 40.0MiB 512.0KiB 45.0MiB 64.8MiB 83.4MiB static 390 1 12s550ms 1s17>
5 rip 768.0KiB 0 5.3MiB 24.7MiB 83.4MiB rip 387 1 1s380ms 1s20>
connected networks >
6 routing policy configuration 512.0KiB 256.0KiB 2189.0KiB 6.0MiB 83.4MiB policy 385 1 1s540ms 1s20>
7 BGP service 768.0KiB 0 2445.0KiB 6.2MiB 83.4MiB bgp 386 1 6s170ms 9s38>
8 BGP Input 10.155.101.217 8.8MiB 6.0MiB 15.6MiB 38.5MiB 83.4MiB 20 21338 1 25s170ms 3s23>
BGP Output 10.155.101.217 >
9 Global memory 256.0KiB global 0 0 >
-- [Q quit|D dump|C-z pause|right]
```

Route filtering differs a bit from ROSv6. In the BGP template, you can now specify output.filter-chain, output.filter-select, input.filter as well as several input.accept-* options.

Now input.accept-* allows filtering incoming messages directly before they are even parsed and stored in memory, that way significantly reducing memory usage. Regular input filter chain can only reject prefixes which means that it will still eat memory and will be visible in /routing route table as "not active, filtered",

A very basic example of a BGP input filter to accept prefixes from 192.168.0.0/16 subnet without modifying any attributes. For other prefixes subtract 1 from the received local pref value and set IGP metric to value from OSPF ext. Additionally, we will accept only specific prefixes from the address list to reduce memory usage

```
/ip/firewall/address-list
add list=bgp_list dst-address=192.168.1.0/24
add list=bgp_list dst-address=192.168.0.0/24
add list=bgp_list dst-address=172.16.0.0/24
```

```
/routing/bgp/template
set default input.filter=bgp_in .accept-nlri=bgp_list
```

```
/routing/filter/rule
add chain=bgp_in rule="if (dst in 192.168.0.0/16) {accept}"
add chain=bgp_in rule="set bgp-local-pref -1; set bgp-igp-metric ospf-ext-metric; accept"
```

If the routing filter chain is not specified BGP will try to advertise every active route it can find in the routing table

The default action of the routing filter chain is "drop"

Monitoring Advertisements

RouterOS v7 by default disables monitoring of the BGP output. This allows to significantly reduce resource usage on setups with large routing tables.

To be able to see output advertisements several steps should be taken:

- enable "output.keep-sent-attributes" in BGP connection configuration
- run "dump-saved-advertisements" from BGP session menu
- view saved output from "/routing/stats/pcap" menu

```
[admin@arm-bgp] /routing/bgp/connection>  set 0 output.keep-sent-attributes=yes
[admin@arm-bgp] /routing/bgp/session> print
Flags: E - established
 0 E remote.address=10.155.101.183 .as=444 .id=192.168.44.2 .refused-cap-opt=no .capabilities=mp,rr,gr,as4
     .afi=ip,ipv6 .messages=4 .bytes=219 .eor=""
     local.address=10.155.101.186 .as=456 .id=10.155.255.186 .capabilities=mp,rr,gr,as4 .afi=ip,ipv6
     .messages=1 .bytes=19 .eor=""
     output.procid=66 .filter-chain=bgp_out .network=bgp-nets .keep-sent-attributes=yes
     input.procid=66 ebgp
     hold-time=3m keepalive-time=1m uptime=4s30ms

[admin@arm-bgp] /routing/bgp/session> dump-saved-advertisements 0 save-to=test_out.pcap
```

Networks

Lastly, you might notice that the `network` menu is missing and probably wondering how to advertise your own networks. Now networks are added to the firewall address-list and referenced in the BGP configuration.
Following ROSv6 network configuration:

```
/routing bgp network add network=192.168.0.0/24 synchronize=yes
/ip route add dst-address=192.168.0.0/24 type=blackhole
```

would translate to v7 as:

```
/ip/firewall/address-list/
add list=bgp-networks address=192.168.0.0/24

/ip/route
add dst-address=192.168.0.0/24 blackhole

/routing/bgp/connection
set peer_name output.network=bgp-networks
```

There is more configuration to be done when adding just one network but offers simplicity when you have to deal with a large number of networks. v7 even allows specifying for each BGP connection its own set of networks.

> ⚠ In v7 it is not possible to turn off synchronization with IGP routes (the network will be advertised only if the corresponding IGP route is present in the routing table).

Routing Filters

Starting from ROSv7.1beta4, the routing filter configuration is changed to a script-like configuration. The rule now can have "if .. then" syntax to set parameters or apply actions based on conditions from the "if" statement.

Multiple rules without action are stacked in a single rule and executed in order like a firewall, the reason is that the "set" parameter order is important and writing one "set"s per line, allows for an easier understanding from top to bottom on what actions were applied.

For example, match static default route and apply action accept can be written in one config rule:

```
/routing/filter/rule
add chain=ospf_in rule="if (dst==0.0.0.0/0 && protocol static) { accept }"
```

For example, ROSv6 rule "/routing filter add chain=ospf_in prefix=172.16.0.0/16 prefix-length=24 protocol=static action=accept" converted to ROSv7 would be:

```
/routing/filter/rule
add chain=ospf_in rule="if (dst in 172.16.0.0/16 && dst-len==24 && protocol static) { accept }"
```

Another example, to match prefixes from the 172.16.0.0/16 range with prefix length equal to 24 and set BGP med and prepend values

```
/routing/filter/rule
add chain=BGP_OUT rule="if (dst-len==24 && dst in 172.16.0.0/16) { \n
    set bgp-med 20; set bgp-path-prepend 2; accept }"
```

It is also possible to match prefix length range like this

```
/routing/filter/rule
add chain=BGP_OUT rule="if (dst-len>13 && dst-len<31 && dst in 172.16.0.0/16) { accept }"
```

Filter rules now can be used to match or set communities, large communities, and extended communities from the community list:

```
/routing/filter/rule
add chain=bgp_in rule="set bgp-large-communities 200001:200001:10 "
```

If there are a lot of community sets, that need to be applied in multiple rules, then it is possible to define community sets and use them to match or set:

```
/routing/filter/large-community-set
add set=myLargeComSet communities=200001:200001:10


/routing/filter/rule
add chain=bgp_in rule="append bgp-large-communities myLargeComSet "
```

Since route-target is encoded in extended community attribute to change or match RT you need to operate on extended community attribute, for example:

```
/routing/filter/rule
add chain=bgp_in rule="set bgp-ext-communities rt:327824:20 "
```

RPKI

RouterOS implements an RTR client. You connect to the server which will send route validity information. This information then can be used to validate routes in route filters against a group with "rpki-validate" and further in filters "match-rpki" can be used to match the exact state.

For more info refer to the RPKI documentation.

RIP Configuration

To start RIP, the instance should be configured. There you should select which routes will be redistributed by RIP and if it will redistribute the default route.

```
/routing/rip/instance
add name=instance1 originate-default=never redistribute=connected,static
```

Then interface-template should be configured. There is no need to define networks in ROS version 7 as it was in version 6.

```
/routing/rip/interface-template
add interfaces=ether1 instance=instance1
```

Now the basic configuration is completed on one router. RIP neighbor router should be configured in a similar way.

In ROS v7 the neighbors will appear only when there are routes to be sent or/and to be received.

Prefix lists from ROSv6 are deprecated, now all the filtering must be done by the routing filters.