NetConf - ISP Workhop

Donald Jolley <donald@networkstack.co.za>
&
Jaco Kroon <jaco@iewc.co.za>

Workshop Pre-requisites

3 Mikrotik routers with firmware at least 6.40 (newest recommended).

PC (Windows, Linux or MAC OS) capable of running Winbox (or mactelnet @ https://github.com/haakonnessjoen/MAC-Telnet). You will need to be able to connect to your routers using MAC address, so please make sure you're able to do this.

IP Resources

Some IP resources will be required for each section of this workshop. Each participant will be supplied with a single number # on the day of the conference. Wherever you see # simply replace with that given number.

My # value is: _____

Note that we will be using the RFC1918 range 10.0.0.0/8 as our "public internet" range, and we will treat 192.168.0.0/16 and 172.16.0.0/12 as private as per the RFC. Additionally we will treat the 100.64.0.0/10 range as per defined in RFC6598 for LSN purposes.

If by some fluke we get to deal with IPv6 we will be using the ULA range fd00::/8 for allocating to participants.

IXP IPv4 Address: 10.255.0.#/24
"Public" range: 10.#.0.0/16
LSN Range: 100.64.0.0/10
"Private" Ranges: 172.16.0.0/12

IXP IPv6 Address: fd00:ffff::#/64
IPv6 allocation: fd00:#::/32

Your ASN: 65000 + #
IXP ASN: 65500

You need three routers, they will be labeled A through C, eg: Router-A.

Wherever there is an X in an IP address, replace as follows:

Router-A: 1
Router-B: 2
Router-C: 3

If you see something like (#+n) or (X+n) where n is numeric, simply take the values of # or X as per above, and add n to it to get the desired value.

Winbox General notes

Your interface facing your MikroTik router needs to have an IP address. For connecting to MAC address it doesn't matter what the IP is.

You need a default route on your system. It's not quite clear what the requirements are but essentially you need to have a route to 0.0.0.0 as far as I can determine (ie, "ip ro get 0.0.0.0" should come back with a response).

Winbox works under wine in Linux.

https://github.com/haakonnessjoen/MAC-Telnet provides a CLI mactelnet tool for Linux. It's discovery seems to be more reliable than built-in Winbox.

To monitor for CDP on all interfaces: mactelnet -l

To connect to a router: mactelnet aa:bb:cc:dd:ee:ff

This is my preferred mechanism. Seems to be more reliable than Winbox in most cases. Doesn't require IP (L3) to be functional at least. Same requirements in terms of IP configuration as Winbox (IP address on interface + default route on system).

If you're running a firewall, you need to allow traffic to broadcast (255.255.255.255) with the following:

UDP **source** and **destination** port 5678
UDP **source** port 20547

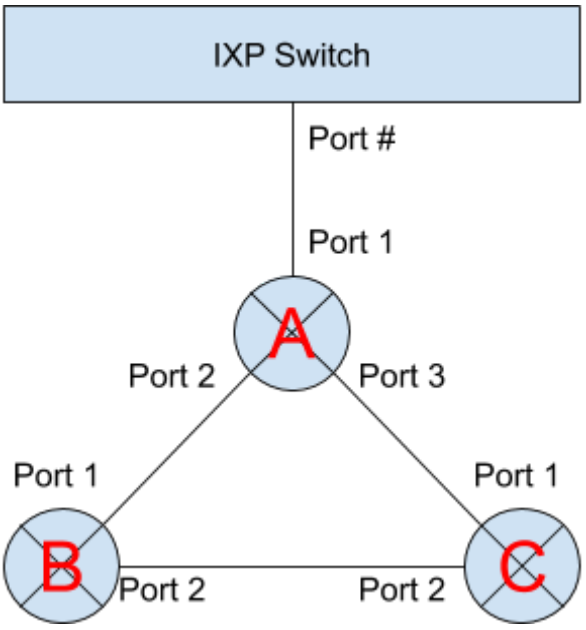
Router Wiring

For the bulk of the workshop we will work with wiring as per the following diagram. Please also change the identities of each router after factory resetting and removing config:

/system reset-configuration
(answer yes).
Once you log back in you will be asked whether you want to keep the default config, or remove it. **Remove it.**

/system identity
set name=????-Router-X#

For example, name=JKroon-Router-A0 (this makes it easier to identify the routers when we see them in CDP.



Bridged Setup

On Router-A, create a bridge interface:

```
/interface bridge
add name=bridge0
/interface bridge port
add bridge=bridge0 interface=ether2
add bridge=bridge0 interface=ether3
add bridge=bridge0 interface=ether4
```

For Router-B and Router-C:

```
/interface bridge
add name=bridge0
/interface bridge port
add bridge=bridge0 interface=ether1
add bridge=bridge0 interface=ether2
add bridge=bridge0 interface=ether4
```

On all three routers:

```
/ip address
add address=10.#.0.X/24 interface=bridge0
```

1. Plug your laptop into port 5 of either Router-B or C.
2. Assign yourself an IP of 10.#.0.4/24.
3. Ping 10.#.0.1.

- What path is the traffic following?
- Is the responses reliable (ping -t for Windows)
- What if you disable STP (/interface bridge [/interface bridge find name=bridge0] set protocol-mode=none)?

General notes:

- No clear border between customer and provider. You really want some form of CE/PE (CPE) separation.
- It's non-trivial to prevent customers from bypassing the CPE and plugging straight into your network. Normally you'd plug a "CPE" in where your laptop is now, which would provide NAT services to the network behind it.
- You require some form of STP to prevent loops on your network - this DISABLES links. There is GOING to be major inefficiencies in you network unless you're really careful about setting bridge parameters (which links are disabled?). Port-status only seems to be visible via Winbox, not CLI.
- VLANs can be used to separate the Customer and Provider networks, so Ports facing customers are set up as Access Ports, and backhaul as Trunk Ports. Use newer RouterOS so that you can handle this in the bridge setup. If you'd like to explore this, mention please and we can look at it. PVIDs are crucial on bridge interface going to CPU, and /interface bridge vlan, for example:

  ```
  /interface bridge
  set [find bridge=bridge0] pvid=1 vlan-filtering=yes

  /interface bridge vlan
  add bridge=bridge0 tagged=ether1,ether2 untagged=ether4
  ```

  RouterOS is finicky about PVID values and cannot run a port without a PVID, set to some unused VLAN if the port should be purely trunked (it insists ports are access or hybrid).

Routed Setup

Please connect to port 4 on Router-A. We'll be working from the far edge back to convert from bridged to routed. We will be using 10.#.1.0/30 for A-B, 10.#.1.4/30 for A-C and 10.#.1.8/30 for B-C. We will also from Router-B and Router-C add a default route via Router-A. For the moment we will also disable the addresses assigned to bridge0, which we will re-introduce later on.

On routers B and C we will also add customer facing IP ranges.

Router-B + Router-C

/interface bridge ports
remove [find interface=ether2]
remove [find interface=ether4]
remove [find interface=ether1]

Note the order. After dropping ether1 you might get disconnected.

Router-B:

/ip address
remove [find]
add address=10.#.1.2/30 interface=ether1
add address=10.#.1.9/30 interface=ether2
add address=10.#.2.1/24 interface=ether4
print

The print is just to verify, in my case with #=0:

```
Flags: X - disabled, I - invalid, D - dynamic
 #    ADDRESS          NETWORK          INTERFACE
 0    10.0.1.2/30      10.0.1.0         ether1
 1    10.0.1.9/30      10.0.1.8         ether2
 2    10.0.2.1/24      10.0.2.0         ether4
```

/ip route
add dst-address=0.0.0.0/0 gateway=10.#.1.1

Router-C:

/ip address
remove [find]
add address=10.#.1.6/30 interface=ether1
add address=10.#.1.10/30 interface=ether2
add address=10.#.3.1/24 interface=ether4
print

For example:

```
Flags: X - disabled, I - invalid, D - dynamic
 #    ADDRESS          NETWORK          INTERFACE
 0    10.0.1.6/30      10.0.1.4         ether1
 1    10.0.1.10/30     10.0.1.8         ether2
 2    10.0.3.1/24      10.0.3.0         ether4
```

At this point these two routers should be able to ping each other directly:

Router-B: /ping 10.#.1.10
Router-C: /ping 10.#.1.9

Since Router-A will be our border, let's point our default gateways for Router-B and Router-C at Router-A:

/ip route
Router-B: add dst-address=0.0.0.0/0 gateway=10.#.1.1 ← **TODO FIX already added.**
Router-C: add dst-address=0.0.0.0/0 gateway=10.#.1.5

At this point, it's probably best to disconnect from Router-B and Router-C since the next step is going to temporarily kill our connectivity anyway.

Now we need to update Router-A:

/interface bridge port
remove [find]

/ip address
remove [find]
add address=10.#.1.1/30 interface=ether2
add address=10.#.1.5/30 interface=ether3
print

Flags: X - disabled, I - invalid, D - dynamic

#	ADDRESS	NETWORK	INTERFACE
0	10.0.1.1/30	10.0.1.0	ether2
1	10.0.1.5/30	10.0.1.4	ether3

And let's just add a management interface here to make things easier later on:

/ip address
add address=10.#.255.1/24 interface=ether4

I recommend adding 10.#.255.2/24 to your ethernet interface on your laptop at this stage, with a default gateway pointing at 10.#.255.1. If you prefer pointing your default gateway at the conference WiFi, then at least point the following routes at 10.#.255.1 (and wherever a default route is mentioned going forward for your laptop, use these routes):

10.0.0.0/8
172.16.0.0/12
192.168.0.0/16
100.64.0.0/10

After adding the above routes, you should be able to access Router-B and Router-C again on 10.#.1.2 and 10.#.1.6 respectively, using either ssh or winbox.

At this point, it should be noted that you cannot access Router-B or Router-C using 10.#.1.9 and 10.#.1.10 respectively from your laptop. Or their customer facing IP addresses. In order to do this, on Router-A:

/ip route
add dst-address=10.#.2.0/24 gateway=10.#.1.2
add dst-address=10.#.3.0/24 gateway=10.#.1.6

The above two will enable access to the customer LANs … but what about 10.#.1.8/30 - should we route that through 10.#.1.2 or 10.#.1.6 or both?

add dst-address=10.#.1.8/30 gateway=10.#.1.2,10.#.1.6

Will work.

So let's test this, from your laptop, ping each of the following:

10.#.1.9
10.#.1.10
10.#.2.1
10.#.3.1

Manual Routing Redundancy

From the above pings, how do things change if you break any of the three links between the routers?

10.#.2.1 becomes unavailable if you break the A-B link.
10.#.3.1 becomes unavailable if you break the A-C link.

10.#.1.9 and .10 follows the same pattern, however, this is because return routing breaks, Router-A actually does fail over, which can be verified by breaking either A-B or A-C and then running "/ip route check 10.#.1.9 once" and "/ip route check 10.#.1.10 once".

When both the A-B and A-C links are available, it seems that the routing is based on some form of hash function, so 10.#.1.9 and 10.#.1.10 could individually break or not depending on unknown factors if the B-C link breaks. This can be verified by replugging port 1 and or port 2 on either Router-B or Router-C into (unconfigured) port 3 on same, and then checking /ip route check again on Router-A.

So basically, we want to make sure things keep working, no matter which link gets broken right?

We also want traffic between customers on Router-B and Router-C to take the shortest possible path.

So let's fix that first, on Router-B

/ip route add dst-address=10.#.3.0/24 gateway=10.#.1.10

And for Router-C

/ip route add dst-address=10.#.2.0/24 gateway=10.#.1.9

If you compare /ip route check from either router before and after the above you will note that previously it would have routed to Router-A, which would have routed it back. So traffic from B to C would go B -> A -> C and vice versa. With the new routes it utilizes the direct link.

However, if you now break A-C then those customers can't route to each other any more at all. Fortunately RouterOS has a gateway check mechanism which (partially) solves this issue, so on all routers:

/ip route
set [find static] check-gateway=arp

At this point, start "/ping 10.#.3.1 src-address=10.#.2.1" on Router-B, and "/ip route check 10.#.2.1" on Router-C. You should now notice that if you unplug the B-C link that the traffic will now re-route via Router-A.

Similar strategies are possible for sorting out routing from Router-A too … but this should be adequate to illustrate the redundancy solution - and hopefully you can clearly see that this will not scale.

OSPF

Fortunately the "Internet Engineers" has solved this problem, and various dynamic routing solutions exist. The one we will use is called OSPF, or "Open Shortest Path First". OSPF has various interesting options, including the concept of areas, and different route types that affects metrics et al. For our purposes we will use type-1 routes (route cost increases by link cost as we move away from the origin, compared to type-2 where the cost remains unchanged irrespective of link cost).

OSPF by default has no authentication. You do NOT want to expose this to your customers. It should ONLY be running on your interfaces on your CORE network. So with this in mind, and keeping the IP addressing from the Routed setup we proceed to enable OSPF with a single "area". Be really aware of what's going on, and always check the active interfaces.

Since all our “core” links are using addresses from 10.#.1.0/24 (divided into /30s) we will use this as our “backbone” area (Which RouterOS creates for us by default, and really is just area 0.0.0.0 in OSPF).

On all routers:

/routing ospf
network add area=backbone network=10.#.1.0/24

“interface print” at this point should show ether2 and ether3 active on Router-A, and ether1 and ether2 on Router-B and Router-C.

On any router you can also issue (from /routing ospf) “route print” which will show our backbone routes on all three routers. In order to get our customer and management ranges redistributed we merely need to inform RouterOS to redistribute “connected” routes as type-1 (so that the shortest effective path will be followed):

/routing ospf instance set 0 redistribute-connected=as-type-1

If you re-issue /routing ospf route print you should now see a few extra routes, specifically you should see 6 routes in total (assuming all ether4 interfaces have link, so by default you will only see 4 routes, the three /30 subnets and the /24 to which you’re connected), for example, I’m attached to my “mangement” network on Router-A, and on Router-A:

```
[admin@JKroon-Router-A0] /routing ospf> route print
 # DST-ADDRESS      STATE         COST                GATEWAY       INTERFACE
 0 10.0.1.0/30      intra-area    10                  0.0.0.0       ether2
 1 10.0.1.4/30      intra-area    10                  0.0.0.0       ether3
 2 10.0.1.8/30      intra-area    20                  10.0.1.2      ether2
                                                      10.0.1.6      ether3
 3 10.0.255.0/24    imported-ext-1 20
```

The following should be noted: On Router-A the B-C link is picked up correctly as a multi-path route.

If I link ether4 on Router-B with ether4 on Router-C (simply so that they have link) the above becomes:

```
[admin@JKroon-Router-A0] /routing ospf> route print
 # DST-ADDRESS      STATE         COST                GATEWAY       INTERFACE
 0 10.0.1.0/30      intra-area    10                  0.0.0.0       ether2
 1 10.0.1.4/30      intra-area    10                  0.0.0.0       ether3
 2 10.0.1.8/30      intra-area    20                  10.0.1.2      ether2
                                                             10.0.1.6      ether3
 3 10.0.2.0/24      ext-1         30                  10.0.1.2      ether2
 4 10.0.3.0/24      ext-1         30                  10.0.1.6      ether3
 5 10.0.255.0/24    imported-ext-1 20
```

Note that the /24 ranges follows the shortest path, and does not add multi-path routes.

If you issue /ip route print you will see the “o” (OSPF) routes added.

At this point you can drop all static routes:

/ip route
remove [find static]

And all routers should still be accessible from whichever /24 (ether4) network you’re attached to.

Note that there is no longer a default route on any of the routers. We can originate one from Router-A by doing:

/routing ospf instance set 0 distribute-default=always-as-type-1

By default OSPF will NEVER originate a default route, and has to be told to do so explicitly. You can redistribute bgp, and even if BGP has a default, it still won’t redistribute unless explicitly told to do so.

Take the time to see how traffic re-routes on link failures.

There are a few caveats we will look at, at various other stages in the below. You’ll probably discover a few others.

Loopback addressing

Given the above config - on which IP addresses should you access your routers? Depending on link state, not all IP addresses will appear in your routing tables.

And thus enters the concept of a “loopback address”.

We’ve previously added (and subsequently removed) IP ranges from 10.#.0.0/24. We will now re-add those addresses, but this time as /32 addresses.

/ip address add interface=bridge0 address=10.#.0.X/32

We assign these addresses to the previously utilised bridge0 interface, which no longer has any ports. This is because RouterOS doesn’t have a “lo” or “loopback” interface that we can assign extra addresses to. A bridge interface is the closest thing to a loopback interface that RouterOS provides us with, and in this particular use-case is as good as.

/ip route print on any of the routers will immediately show the additional /32 routes having being added to the routing tables. Since bridge0 is “always up” these loopback addresses will always be available to OSPF, and thus to the routing table.

Furthermore, I always explicitly set things like “OSPF router-id” to this value. It helps a great deal if you only need to deal with a single set of addresses.

/routing ospf instance set 0 router-id=10.#.0.X

It’s also possible to use routing filters to get OSPF to install routes into the kernel routing table using a preferred source of the loopback, and for good measure, enable ARP monitoring of any given gateways (in case link stays up but the gateway becomes non-responsive):

/routing filter
add chain=ospf-in set-check-gateway=arp set-pref-src=10.#.0.X

In this way you can use public addresses for loopbacks (so that when people action a traceroute through your network they see public addresses, not privates, some people seem to think you have to always use public addressing on all ISP infrastructure), and private RFC1918 or LSN (100.64.0.0/10) addresses for inter-router links, which will never be exposed.

If you’re using PPPoE as a mechanism of connecting your customers onto your network, you can use the loopback for the local address, and either addressing from Radius, local static or IP Pools.

OSPF and IP Pools

On Router-B, create an IP pool:

/ip pool add name=customers ranges=10.#.4.0/24

And bring up a PPPoE service on ether4:

/ppp profile
set [find] local-address=10.#.0.2 remote-address=customers
/ppp secret
add name=pppuser password=ppppass service=pppoe

/interface pppoe-server server
add disabled=no interface=ether4 service-name=service1

Now either bring up that PPPoE from your laptop (possibly simpler), or use a bridge between Router-B/ether4 and Router-C/ether4 and dial a PPPoE from Router-C:

```
/interface pppoe-client add interface=ether4 \
        add-default-route=no name=pppoe-dummy \
        user=pppuser password=ppppass disabled=no
```

If at this point you issue /ip route print on Router-A you will note a /32 for the newly connected PPPoE. This is definitely not ideal.

If you're using PPPoE from Router-C, note that it doesn't redistribute it's own local address, it's not the Connected address, however, it's address is Connected on Router-B, and as such gets redistributed from Router-B and thus why Router-A has the /32 pointing at Router-B.

This is not optimal, since 256 customers can connect to Router-B here, this would result in 256 individual routes. Absolute chaos in your routing tables throughout your entire network, and we can do better. On Router-B:

```
/ip route add type=unreachable dst-address=10.#.4.0/24
/routing ospf instance set 0 redistribute-static=as-type-1
```

This will add a static unreachable route on Router-B, and then request OSPF to redistribute this new static route. Note that if you have other static routes (which up to now would note have been redistributed) they too will now be redistributed.

At this point if you look on either Router-A or Router-C you will note a route for 10.#.4.0/24 pointing at Router-B.

The /32 routes are still remaining, so let's use routing filters to not redistribute those:

```
/routing filter
add action=discard chain=ospf-out prefix=10.#.4.0/24 prefix-length=32 protocol=connect
```

This will prevent the /32 routes from being advertised to neighbors.

LSA updates may still occur as a result of link-state-changes.

The unreachable route is critical, it has to match the pool exactly. It serves two purposes:

1. It provides a static for OSPF to redistribute; and
2. It results in an unreachable routing response in case traffic gets sent here, which would otherwise be routed to the default route (and end up being routed back here).

Peering at the INX (eBGP)

So being able to communicate inside your own network is useful, but your customers are probably not going to appreciate you not having Google, Facebook, Netflix or any of the other content providers available.

The internet standard is to use BGP between autonomous systems. Once you sign up with AfriNIC or one of the other RIRs, you will obtain an assignment of IPv4 (presumably a /22, so you need to stretch this as far as possible), IPv6 (probably a /32) as well as an AS number. You should set up the required whois object as quickly as possible, and ensure they are properly protected, AfriNIC has resources on this.

Peering with a transit provider is similar to peering at an INX, chances are they will give you options with respect to how much detail you'd like to receive from them on the peering, generally you will want to simply get a default route from them, or their "local" view + default, or (and not everybody can provide this to you) the full view of the internet, without a default route. The last option only really makes sense if you've got multiple transit providers and really care about the finest of details and insist on operating in the DFZ (Default Free Zone).

Everything we will do in this section will happen on Router-A.

First we need to assign the INX address to ether1:

/ip address add address=10.255.0.#/24 interface=ether1

Most INXes have strict policies regarding what you may and may not do. For example, at NAP you may not transmit ANY traffic that isn't INX related, in other words, you may not broadcast anything other than ARP (CDP, I'm looking at you). You may not push DHCP or request it. You must find out and comply with these requirements. In our demo environment, just follow the instructions and all will be fine. From stock RouterOS you merely need to disable neighbor discovery. You may want to keep CDP working on some interface (personally I just like to switch it off completely):

/ip neighbor discovery-settings set discover-interface-list=none

At this point you should be able to ping 10.255.0.254 (which for our purposes will be the INX's route reflector).

The next step is to set up your BGP instance.

/routing bgp
instance set 0 as=(#+65000) router-id=10.#.0.1

And now we can establish with the remote peer:

peer add address-families=ip default-originate=never name=netconf-inx \
nexthop-choice=force-self remote-address=10.255.0.254 remote-as=65500

Issuing “peer print detail” should show the connection as established:

```
[admin@JKroon-Router-A0] /routing bgp> peer print detail
Flags: X - disabled, E - established
 0 E name="netconf-inx" instance=default remote-address=10.255.0.254
remote-as=65500 tcp-md5-key=""
       nexthop-choice=force-self multihop=no route-reflect=no hold-time=3m ttl=255
in-filter=""
       out-filter="" address-families=ip default-originate=never
remove-private-as=no
       as-override=no passive=no use-bfd=no
```

At this point if you do /ip route print you will not see any effective changes, simply because, well, in spite of everybody peering nobody is actually advertising any routes!

So let’s advertise our routes:

instance set 0 redistribute-ospf=yes redistribute-connected=yes

If other parties have completed this, you should see their internal routes in your routing table!!
This is all good and well, but you can immediately see there is a bunch of /32s et all from other parties that, frankly, we really don’t want.

The general policy on the internet is that you may not distribute routes more specific than a /24 anyway (Our INX is completely unfiltered for illustrative purposes, but places like NAP is extremely strict and enforces routing filters based on IRR objects from whois). Even with /24s as the most specific that’s a potential total of 2^{24} = 16’777’216 routes in any given routing table.

So let’s disable redistributing routes again:

instance set 0 redistribute-ospf=no redistribute-connected=no

Instead, we will advertise our network as a whole:

/routing bgp network add network=10.#.0.0/16
/ip route add type=unreachable dst-address=10.#.0.0/16

The latter of these statements is critical. The first line tells BGP that if we have a route matching this network, then advertise it. The latter adds a route (albeit and unreachable one) to that network. This also has a side effect of eliminating routing loops. Consider what would happen if you had a default route going out from Router-A, say pointing at another

peer on the INX (10.255.0.253), and someone sent you traffic to 10.#.100.1 which you don't have an internal route for, but matches the default route - this would result in you passing traffic destined for your network back out, and that party would send it back to you. TTL (or HL in IPv6) times.

At this point in time inter-AS routing should be functional.

You should see a 10.#.0.0/16 route for every other participant on your Router-A.

These routes won't propagate into your network, only a default route is created by OSPF, Router-A will respond with destination-network-unreachable for any destinations it doesn't see.

Prefix Filters

Obviously we don't trust our peering partners. As such we need to filter the advertisements that we receive from partners. INXs tends to do this for us, and our "upstream" transit providers we generally need to trust. However, it's probably a good idea to still apply at least basic filters to these.

We should also filter our egress routes, just to make sure that we're good netizens. And if we advertise routes, we should be prepared to make good on our promise of delivery.

We will keep things simple here, but it should be noted that given communities etc this can get quite complicated.

Firstly, let's leak a "private" route:

```
/routing bgp network add network=192.168.#.0/24
/ip route add type=unreachable dst-address=192.168.#.0/24
```

(Note that normally this would come from some other eBGP or iBGP session, but we're trying to illustrate the concept here.)

At this point everyone should see a leaked /24 from everyone else.

Let's stop the leaking.

Firstly, we create a chain that will match out "public" ranges, if (and only if) the prefix length is in the acceptable range:

```
/routing filter
add action=accept chain=bgp-ip4own prefix=10.#.0.0/16 prefix-length=0-32
add action=discard chain=bgp-ip4own
```

This will match any route that's 10.#.0.0/16 or more specific. Generally if you've got multiple blocks from an RIR, just expand the chain.

From that we can then create a filter which will allow all advertisements for our own routes, that's at most specific to a /24.

```
add action=accept chain=bgp4-ixp-out match-chain=bgp-ip4own prefix-length=0-24
add action=reject chain=bgp4-ixp-out
```

Note that the above can be done in a single chain, however, we've got an opportunity for chain re-use on our own routes (also why I set prefix-length 0-32 on the inner chain and 0-24 on the outer).

And lastly we need to link the chain to the bgp peer:

```
/routing bgp peer
set [find name=netconf-inx] out-filter=bgp4-ixp-out
```

As others add these filters to their setup, you should no longer see them incoming to you. They may need to re-advertise their routes (or just wait long enough):

```
/routing bgp peer
resend [find name=netconf-inx]
```

And here is why you should know what the INX looking glass access mechanisms are - because at this point you need to ask someone to check for you, instead of looking for yourself. Our netconf-inx has no looking glass feature (But the presenter will leave his routing table up on the projector so you can view that).

You should also be filtering ingress.

Our example here should be considered the absolute minimum. Firstly, a chain that will match on bogons:

```
/routing filter
add action=accept chain=bgp4-bogons prefix=100.64.0.0/10 prefix-length=0-32
add action=accept chain=bgp4-bogons prefix=192.168.0.0/16 prefix-length=0-32
add action=accept chain=bgp4-bogons prefix=10.0.0.0/8 prefix-length=0-32 disabled=yes
add action=accept chain=bgp4-bogons prefix=172.16.0.0/12 prefix-length=0-32
add action=reject chain=bgp4-bogons
```

10.0.0.0/8 is excluded here because it's our fake public range.

Note that this will accept in case of bogon, so when we filter the statement reads more logical:

add action=discard chain=bgp4-inx-in match-chain=bgp4-bogons

I read that as “if this is a bogon, discard the route”.

Should a route that’s more specific than a /24 be considered a bogon or is that some other “condition”? Depending on your viewpoint, you could use one of the following options:

add action=accept chain=bgp4-bogons prefix-length=25-32

And move the rule above the reject, or:

add action=discard chain=bgp4-ixp-in prefix-length=25-32

The option you prefer is up to you. Alternatively, as part of your final accept, just restrict to prefix-length=0-24.
Before we do that, reject your own prefixes, nobody should be sending them to you ever (note the re-use here):

add action=discard chain=bgp4-ixp-in match-chain=bgp-ip4own

In the real world we also need to reject our own customer prefixes (or add them to bgp-ip4own more specifically). We should filter as-paths, and we should filter for as/route originations. See below. For now, let’s just assume that everybody plays nice and accept everything other than the most obvious problems:

add action=accept chain=bgp4-ixp-in prefix-length=0-24

And a final rule to reject everything else:

add action=discard chain=bgp4-ixp-in

And finally, link the filter to ingress:

/routing bgp peer
set [find name=netconf-inx] in-filter=bgp4-ixp-in

This time we need to request the remote peer to resend back to us:

/routing bgp peer
refresh [find name=netconf-inx]

If there were (presenter to originate a few bogons) still bad stuff coming from the peers, the above should filter at least the worst of it.

You really should use tools such as bgpq3 (https://github.com/snar/bgpq3/) to build proper filters. MikroTik patches exist. Basically my strategy is as follows:

- Trust an INX to filter what it receives. If you're going to filter this you're chains are going to be insane.
- Trust your upstreams / transit providers (to an extent) to not be stupid.
- With all due respect to them, distrust your downstreams and peers.

What Next?

If there is still time left, and there is demand, there are hordes of other items we can dig into, some possible ideas:

MPLS
IPv6
PPPoE
iBGP