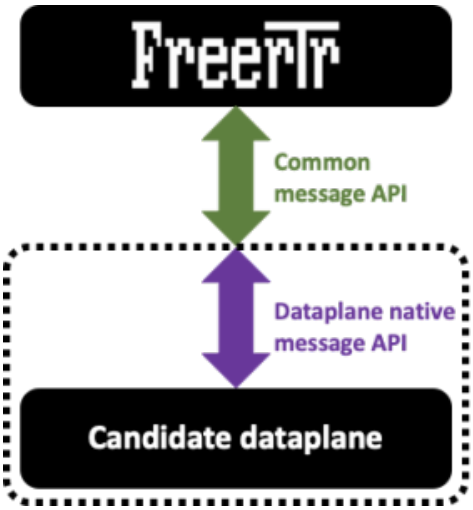
RARE software architecture: [Modular design #002] - "A totally disaggregated model"

This is additional post related to blog series called "RARE software architecture". As its name implies, it deals with topics related to RARE/freeRouter software design choice.

Requirement	
• Basic Linux/Unix knowledge • Service provider networking knowledge	

Overview

In [Modular design #001] article we described RARE/freeRtr main components:



- a control plane
- a data plane which can have different "flavor"
- and a message API interface that is tying the 2 components above

Article objective

In most of our past articles, all the components listed above were running on the same host as it is mostly the case on traditional monolithic hardware. In this article we demonstrate that RARE/freeRtr inherently exposes a totally disaggregated model. Practically, what does this means ? We will show in subsequent section that each of the components above can run on different hosts.

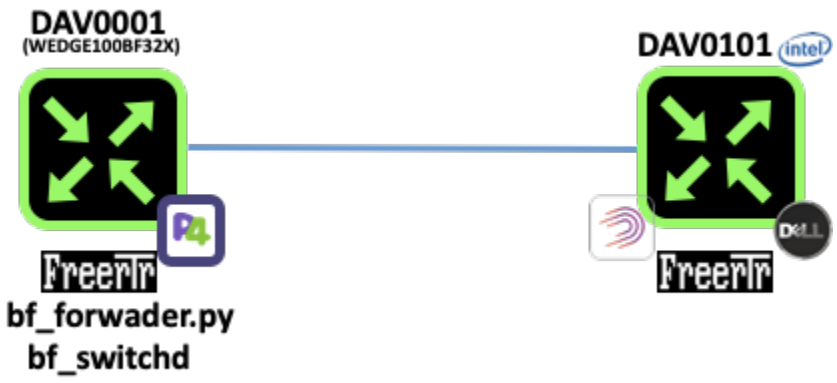
> **Note**
>
> While the control plane and the message API interface can be run from any host such as bare metal server, VM or even container, the dataplane of your choice will have to run on specific platform. For example, if you plan to implement a core backbone MPLS router able to switch 6.4 Tbps of packet throughput, the dataplane must be a specific hardware. (e.g. powered by INTEL TOFINO switch ASIC)

Diagram

In this example we will consider 2 routers named DAV0001 and DAV0101 respectively. Both if these routers are establishing an OSPF adjacency between them.

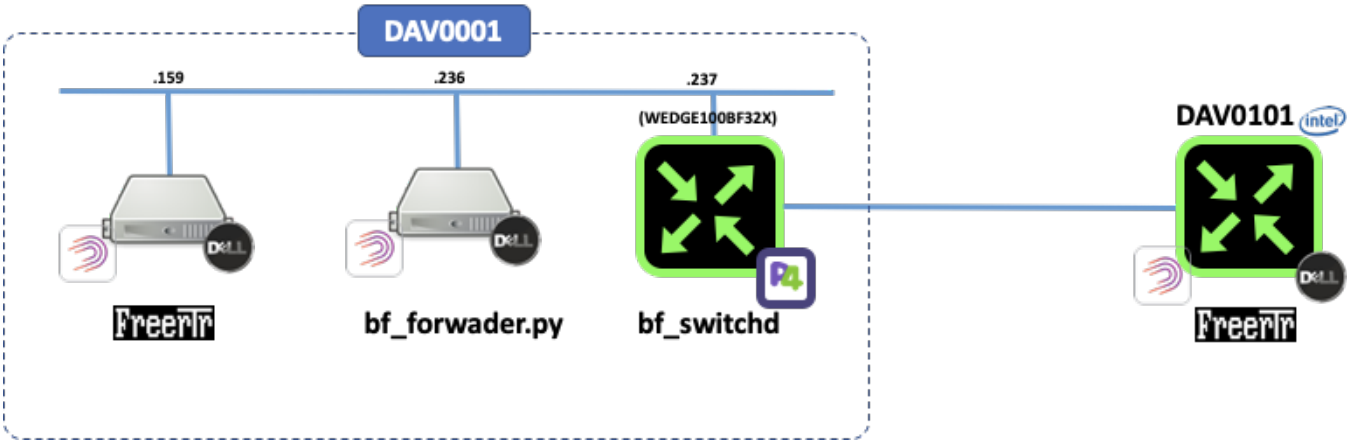
- DAV0001 is a P4 switch powered INTEL TOFINO ASIC
- DAV0101 is a SOHO RARE/freeRtr DELL VEP1445

Let's assume the logical figure below:



[#002] - Modular design

In reality, we will run DAV0001 components in a different host respectively. We assume of course that there is connectivity between each of them.



Let's first start the INTEL TOFINO dataplane.

pcapInt between WEDGE100BF32X and freeRtr

```
root@rare:~# /opt/freertr/bin/start_bfswd.sh
```

remote-dav0001-hw.txt

```
bf_switchd: no process found
Using SDE /opt/bf_switchd
Using SDE_INSTALL /opt/bf_switchd/install
Setting up DMA Memory Pool
Using TARGET_CONFIG_FILE /opt/bf_switchd/install/share/p4/targets/tofino/bf_router.conf
Using PATH /opt/bf_switchd/install/bin:/opt/freertr/bin:/opt/bf_switchd/install/bin:/opt/bf_switchd:/opt
/bf_switchd/tools:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
Using LD_LIBRARY_PATH /usr/local/lib:/opt/bf_switchd/install/lib:
bf_sysfs_fname /sys/class/bf/bf0/device/dev_add
kernel mode packet driver present, forcing kernel_pkt option!
Install dir: /opt/bf_switchd/install (0x55ac03623d20)
bf_switchd: system services initialized
bf_switchd: loading conf_file /opt/bf_switchd/install/share/p4/targets/tofino/bf_router.conf...
bf_switchd: processing device configuration...
Configuration for dev_id 0
  Family        : tofino
  pci_sysfs_str : /sys/devices/pci0000:00/0000:00:03.0/0000:05:00.0
  pci_domain    : 0
  pci_bus       : 5
  pci_fn        : 0
  pci_dev       : 0
  pci_int_mode  : 1
  sbus_master_fw: /opt/bf_switchd/install/
  pcie_fw       : /opt/bf_switchd/install/
  serdes_fw     : /opt/bf_switchd/install/
  sds_fw_path   : /opt/bf_switchd/install/share/tofino_sds_fw/avago/firmware
  microp_fw_path:
bf_switchd: processing P4 configuration...
P4 profile for dev_id 0
num P4 programs 1
  p4_name: bf_router
  p4_pipeline_name: pipe
    libpd:
    libpdthrift:
    context: /opt/bf_switchd/install/share/tofinopd/bf_router/pipe/context.json
    config: /opt/bf_switchd/install/share/tofinopd/bf_router/pipe/tofino.bin
  Pipes in scope [0 1 2 3 ]
  diag:
  accton diag:
  Agent[0]: /opt/bf_switchd/install/lib/libpltfm_mgr.so
  non_default_port_ppgs: 0
  SAI default initialize: 1
bf_switchd: library /opt/bf_switchd/install/lib/libpltfm_mgr.so loaded
bf_switchd: agent[0] initialized
Health monitor started
Operational mode set to ASIC
Initialized the device types using platforms infra API
ASIC detected at PCI /sys/class/bf/bf0/device
ASIC pci device id is 16
Skipped pkt-mgr init
Starting PD-API RPC server on port 9090
bf_switchd: drivers initialized
bf_switchd: dev_id 0 initialized
bf_switchd: initialized 1 devices
Adding Thrift service for bf-platforms to server
bf_switchd: thrift initialized for agent : 0
bf_switchd: spawning cli server thread
bf_switchd: spawning driver shell
bf_switchd: server started - listening on port 9999
bfruntime gRPC server started on 0.0.0.0:50052

        *******************************************
        *      WARNING: Authorised Access Only     *
        *******************************************


bfshell>
```

From that point the dataplane has activated its P4 **CPU_PORT** which is in our case **ens1** Linux interface that appeared when you activate **bf_kpkt** modules.

Now the idea is to stitch **ens1 CPU_PORT** to freeRtr dataplane port and make sure that control plane / dataplane in band communication can occur. The secret sauce main ingredient is **pcapInt** tool. In essence **pcapInt** is a freeRtr tool that will allow you to bind all the packet from an interface over UDP socket tunnel.

pcapInt between WEDGE100BF32X and freeRtr

```
root@rare:~# /opt/freertr/bin/pcapInt.bin ens1 20666 193.49.159.159 20666 193.49.159.237
```

So, the line above can be read as: tunnel all packet from **ens1** interface over UDP tunnel whose source is 193.49.159.237:20666 (bf_switchd@p4 switch) and destination is 193.49.159.159:20666 (freeRtr@server)

pcapInt between WEDGE100BF32X and freeRtr output

```
binded to local port 193.49.159.237 20666.
will send to 193.49.159.159 20666.
pcap version: libpcap version 1.10.0 (with TPACKET_V3)
opening interface ens1
serving others
> d
iface counters:
                      packets                bytes
received                    0                    0
sent                        0                    0

>
```

DAV0001 hardware configuration file:

remote-dav0001-hw.txt

```
hwid remote-wedge100bf32x
port 22000 64000
tcp2vrf 2323 p4 23
tcp2vrf 2001 p4 22
tcp2vrf 9080 p4 9080
int eth0 eth 0000.0bad.c0de 193.49.159.159 20666 193.49.159.237 20666
```

DAV0001 software configuration file:

remote-dav0001-hw.txt

```
hostname DAV0001
buggy
!
logging buffered debug 10240
logging file debug /var/log/freertr.log
logging rotate 655360000 /var/log/freertr.old
!
prefix-list all4
 sequence 10 permit 0.0.0.0/0 ge 0 le 0
 exit
!
prefix-list all6
 sequence 10 permit ::/0 ge 0 le 0
 exit
!
vrf definition inet
 exit
!
vrf definition oob
 exit
```

```
!
vrf definition p4
 description P4 VRF _NEVER_EVER_ CONFIGURE IT
 exit
!
router ospf4 1
 vrf inet
 router-id 10.1.1.1
 traffeng-id 0.0.0.0
 area 0 enable
 exit
!
interface loopback0
 no description
 vrf forwarding inet
 ipv4 address 10.1.1.1 255.255.255.255
 no shutdown
 no log-link-change
 exit
!
interface ethernet0
 description CPU_PORT _NEVER_EVER_ CONFIGURE IT
 no shutdown
 no log-link-change
 exit
!
interface sdn10
 description frontpanel port 10/0
 mtu 1500
 macaddr 0031.755e.0363
 lldp enable
 vrf forwarding inet
 ipv4 address 10.1.12.1 255.255.255.0
 router ospf4 1 enable
 router ospf4 1 area 0
 router ospf4 1 cost 10000
 no shutdown
 log-link-change
 exit
!
interface sdn15
 description frontpanel port 15/0
 mtu 1500
 macaddr 0056.2158.6249
 lldp enable
 vrf forwarding inet
 ipv4 address 10.1.14.1 255.255.255.0
 router ospf4 1 enable
 router ospf4 1 area 0
 no shutdown
 log-link-change
 exit
!
interface sdn7
 description frontpanel port 7/0
 mtu 1500
 macaddr 0040.784a.0b38
 lldp enable
 vrf forwarding inet
 ipv4 address 10.1.13.1 255.255.255.0
 router ospf4 1 enable
 router ospf4 1 area 0
 no shutdown
 log-link-change
 exit
!
proxy-profile oob
 vrf oob
 exit
!
server telnet oob
```

```
 security protocol telnet
 exec logging
 no exec authorization
 no login authentication
 login logging
 vrf oob
 exit
!
server telnet p4
 security protocol telnet
 exec logging
 no exec authorization
 no login authentication
 login logging
 vrf p4
 exit
!
server p4lang p4
 export-vrf inet 1
 export-port sdn15 12 10 0 0 0
 export-port sdn10 52 10 0 0 0
 export-port sdn7 176 10 0 0 0
 interconnect ethernet0
 vrf p4
 exit
!
client proxy oob
client name-server 1.1.1.1
client time-server europe.pool.ntp.org
client time-zone CET
!
end
```

remote-dav0001-hw.txt

```
root@vep1425:~# java -jar /rtr/rtr.jar routersc remote-dav0001-hw.txt remote-dav0001-sw.txt
```

remote-dav0001-hw.txt

```
root@vep1425:~# java -jar /rtr/rtr.jar routersc remote-dav0001-hw.txt remote-dav0001-sw.txt


#######                          ###################
 ##  ##                                   ##
 ##   # ## ###   #####   #####  ## ###  ## ## ###
 ## #    ### ## ##   ## ##   ##  ### ## ##  ### ##
 ####    ##  ## ####### #######  ##  ## ##  ##  ##
 ## #    ##     ##      ##       ##     ##  ##
 ##      ##     ##   ## ##   ##  ##     ##  ##
####     ##      #####   #####   ##     ##  ##

freeRouter v21.9.23-cur, done by cs@nop.

place on the web: http://www.freertr.net/
license: http://creativecommons.org/licenses/by-sa/4.0/
quote1: make the world better
quote2: if a machine can learn the value of human life, maybe we can too
quote3: be liberal in what you accept, and conservative in what you send
quote4: the beer-ware license for selected group of people:
cs@nop wrote these files. as long as you retain this notice you
can do whatever you want with this stuff. if we meet some day, and
you think this stuff is worth it, you can buy me a beer in return

info cfg.cfgInit.doInit:cfgInit.java:662 booting
info cfg.cfgInit.doInit:cfgInit.java:806 initializing hardware
info cfg.cfgInit.doInit:cfgInit.java:812 applying defaults
info cfg.cfgInit.doInit:cfgInit.java:819 applying configuration
warning ifc.ifcEthTyp.propagateState:ifcEthTyp.java:334 interface sdn15 change to up
warning ifc.ifcEthTyp.propagateState:ifcEthTyp.java:334 interface sdn10 change to up
warning ifc.ifcEthTyp.propagateState:ifcEthTyp.java:334 interface sdn7 change to up
info cfg.cfgInit.doInit:cfgInit.java:849 boot completed
welcome
line ready
DAV0001#
```

Now it is the time to establish the interface between the control plane and the dataplane

remote-dav0001-hw.txt

```
root@rare:~# /opt/freertr/bin/bf_forwarder.py --freerouter-address 193.49.159.159 --freerouter-port 9080 --
bfruntime-address 193.49.159.237:50052
```

basically you can read this command as follow:

- bf_forwarder.py is running 193.49.159.236
- it is binding freeRtr control plane running at 193.49.159.159
- and P4 dataplane running at 193.49.159.237

The output below is showing you a successful binding operation and the entries creation during control plane and dataplane communication.

remote-dav0001-hw.txt

```
bf_forwarder.py running on: WEDGE100BF32X
GRPC_ADDRESS: 193.49.159.237:50052
P4_NAME: bf_router
CLIENT_ID: 0
Subscribe attempt #1
Subscribe response received 0
Received bf_router on GetForwarding on client 0, device 0
Binding with p4_name bf_router
Binding with p4_name bf_router successful!!
bf_switchd started with no SNMP export
```

```
Generic table clearing: (Order not matters)
  Clearing Table pipe.eg_ctl.eg_ctl_vlan_out.tbl_vlan_out
  Clearing Table pipe.ig_ctl.ig_ctl_outport.tbl_vlan_out
  Clearing Table pipe.ig_ctl.ig_ctl_mcast.tbl_mcast6
  Clearing Table pipe.ig_ctl.ig_ctl_mcast.tbl_mcast4
  Clearing Table pipe.ig_ctl.ig_ctl_mpls.tbl_mpls_fib_decap
  Clearing Table pipe.ig_ctl.ig_ctl_vrf.tbl_vrf
  Clearing Table pipe.ig_ctl.ig_ctl_mpls.tbl_mpls_fib
  Clearing Table pipe.eg_ctl.eg_ctl_mcast.tbl_mcast
  Clearing Table pipe.eg_ctl.eg_ctl_nexthop.tbl_nexthop
  Clearing Table pipe.ig_ctl.ig_ctl_outport.tbl_nexthop
  Clearing Table pipe.ig_ctl.ig_ctl_ipv6.tbl_ipv6_fib_lpm
  Clearing Table pipe.ig_ctl.ig_ctl_ipv4.tbl_ipv4_fib_host
  Clearing Table pipe.ig_ctl.ig_ctl_ipv6.tbl_ipv6_fib_host
  Clearing Table pipe.ig_ctl.ig_ctl_vlan_in.tbl_vlan_in
  Clearing Table pipe.eg_ctl.eg_ctl_hairpin.tbl_hairpin
  Clearing Table pipe.ig_ctl.ig_ctl_ipv4.tbl_ipv4_fib_lpm
Bundle specific clearing: (Order matters)
  Clearing Table pipe.ig_ctl.ig_ctl_bundle.tbl_nexthop_bundle
  Clearing Table pipe.ig_ctl.ig_ctl_bundle.ase_bundle
  Clearing Table pipe.ig_ctl.ig_ctl_bundle.apr_bundle
Multicast specific clearing: (Order matters)
  Clearing Table $pre.mgid
  Clearing Table $pre.node
BfForwarder - Main
BfForwarder - Entering message loop
BfIfStatus - main
rx: ['state', '12', '1', '10', '0', '0', '0', '\n']
BfSubIfCounter - main
BfSubIfCounter - No active ports
BfIfStatus - No active ports
rx: ['mtu', '12', '1500', '\n']
rx: ['portvrf_add', '12', '1', '\n']
rx: ['tcpmss4in_add', '12', '0', '\n']
rx: ['tcpmss4out_add', '12', '0', '\n']
rx: ['state', '52', '1', '10', '0', '0', '0', '\n']
rx: ['mtu', '52', '1500', '\n']
rx: ['portvrf_add', '52', '1', '\n']
rx: ['tcpmss4in_add', '52', '0', '\n']
rx: ['tcpmss4out_add', '52', '0', '\n']
rx: ['state', '176', '1', '10', '0', '0', '0', '\n']
rx: ['mtu', '176', '1500', '\n']
rx: ['portvrf_add', '176', '1', '\n']
rx: ['tcpmss4in_add', '176', '0', '\n']
rx: ['tcpmss4out_add', '176', '0', '\n']
rx: ['myaddr4_add', '224.0.0.0/4', '-1', '1', '\n']
rx: ['myaddr4_add', '255.255.255.255/32', '-1', '1', '\n']
rx: ['myaddr6_add', 'ff00::/8', '-1', '1', '\n']
rx: ['myaddr4_add', '10.1.1.1/32', '-1', '1', '\n']
rx: ['myaddr4_add', '10.1.12.0/24', '52', '1', '\n']
rx: ['myaddr4_add', '10.1.12.1/32', '52', '1', '\n']
rx: ['myaddr4_add', '10.1.13.0/24', '176', '1', '\n']
rx: ['myaddr4_add', '10.1.13.1/32', '176', '1', '\n']
rx: ['myaddr4_add', '10.1.14.0/24', '12', '1', '\n']
rx: ['myaddr4_add', '10.1.14.1/32', '12', '1', '\n']
rx: ['mylabel6_add', '126184', '1', '\n']
rx: ['mylabel4_add', '835737', '1', '\n']
BfIfStatus - PORTS_OPER_STATUS[176] does not exist, adding it ...
BfIfStatus - PORTS_OPER_STATUS[52] does not exist, adding it ...
BfIfStatus - PORTS_OPER_STATUS[12] does not exist, adding it ...
tx: ['state','176','1','\n']
rx: ['neigh4_add', '32360', '10.1.13.3', '00:5b:09:4d:1e:40', '1', '00:40:78:4a:0b:38', '176', '\n']
rx: ['nhop2port_add', '32360', '176', '176', '\n']
rx: ['route4_add', '10.1.3.3/32', '32360', '10.1.13.3', '1', '\n']
rx: ['keepalive', '\n']
...
```

Verification

Once bf_forwader.py established the connection with the control plane and the dataplane, freeRtr should expose "session UP" message on console

remote control plane outout

```
welcome
line ready
DAV0001#warning serv.servP4lang.srvAccept:servP4lang.java:597 neighbor 193.49.159.236 up
```

Check that you are sending and receiving packet from **freeRtr@eth0** and **P4_switch@CPU_PORT**:

remote control plane outout

```
DAV0001#sh int
sum
interface  state  tx       rx           drop
loopback0  up     0        0            0
ethernet0  up     34946    12232        0
sdn10      up     10928+0  0+0          0+0
sdn15      up     10928+0  0+0          0+0
sdn7       up     12214+0  11930+14346  0+0
```

LLDP is your friend

Physical connectivity verification

```
DAV0001#show lldp
neighbor
interface  hostname  iface  ipv4       ipv6
sdn7       DAV0101   sdn6   10.1.13.3  null
```

In the above configuration we have enabled OSPFv4 adjacency via **sdn7** with **DAV0101@sdn6**

Control plane verification

```
DAV0001#sh ipv4 ospf 1
int
interface  neighbors
sdn10      0
sdn15      0
sdn7       1

DAV0001#show ipv4 ospf 1
neighbor
interface  area  address    routerid  state  uptime
sdn7       0     10.1.13.3  10.1.3.3  4      00:05:18

DAV0001#sh ipv4 route
inet
typ  prefix        metric  iface      hop        time
C    10.1.1.1/32   0/0     loopback0  null       00:08:13
O    10.1.3.3/32   110/1   sdn7       10.1.13.3  00:07:32
C    10.1.12.0/24  0/0     sdn10      null       00:08:12
LOC  10.1.12.1/32  0/1     sdn10      null       00:08:12
C    10.1.13.0/24  0/0     sdn7       null       00:08:12
LOC  10.1.13.1/32  0/1     sdn7       null       00:08:12
C    10.1.14.0/24  0/0     sdn15      null       00:08:12
LOC  10.1.14.1/32  0/1     sdn15      null       00:08:12
```

The important thing to note is that we are pinging from a control plane that is running on a different host as the dataplane. Nothing special here except that this seemed to be totally invisible from the operator point of view !

mandatory ping

```
DAV0001#ping 10.1.3.3 /vrf
inet
pinging 10.1.3.3, src=null, vrf=inet, cnt=5, len=64, tim=1000, gap=0, ttl=255, tos=0, fill=0, sweep=false,
multi=false, detail=false
!!!!!
result=100%, recv/sent/lost/err=5/5/0/0, rtt min/avg/max/total=1/1/2/9
DAV0001#
```

Discussion

This modular design and its property to run as fully disaggregated model is powerful as:

- each component are inter-changeable, connecting freeRtr to another dataplane is a seamless operation
- it can be the foundation of a decentralized architecture where the control plane is not attached to the dataplane
- Resiliency has of course to be thouroughly strenghtened

Conclusion

In this 2nd article you:

- had a showcase on how to implement a fully disaggregated RARE/freeRtr
- even if the control plane and the interface can be run almost anywhere, the dataplane still needs to be specific and adapted to the use case you planned to deploy
- **pcapInt** tool is a nitty gritty tool used to bind existing ports to a UDP socket tunnel.