P4 Bootcamp - Labs Guide

Vladimir Gurevich

November 2015

Environment Introduction

Lab VM

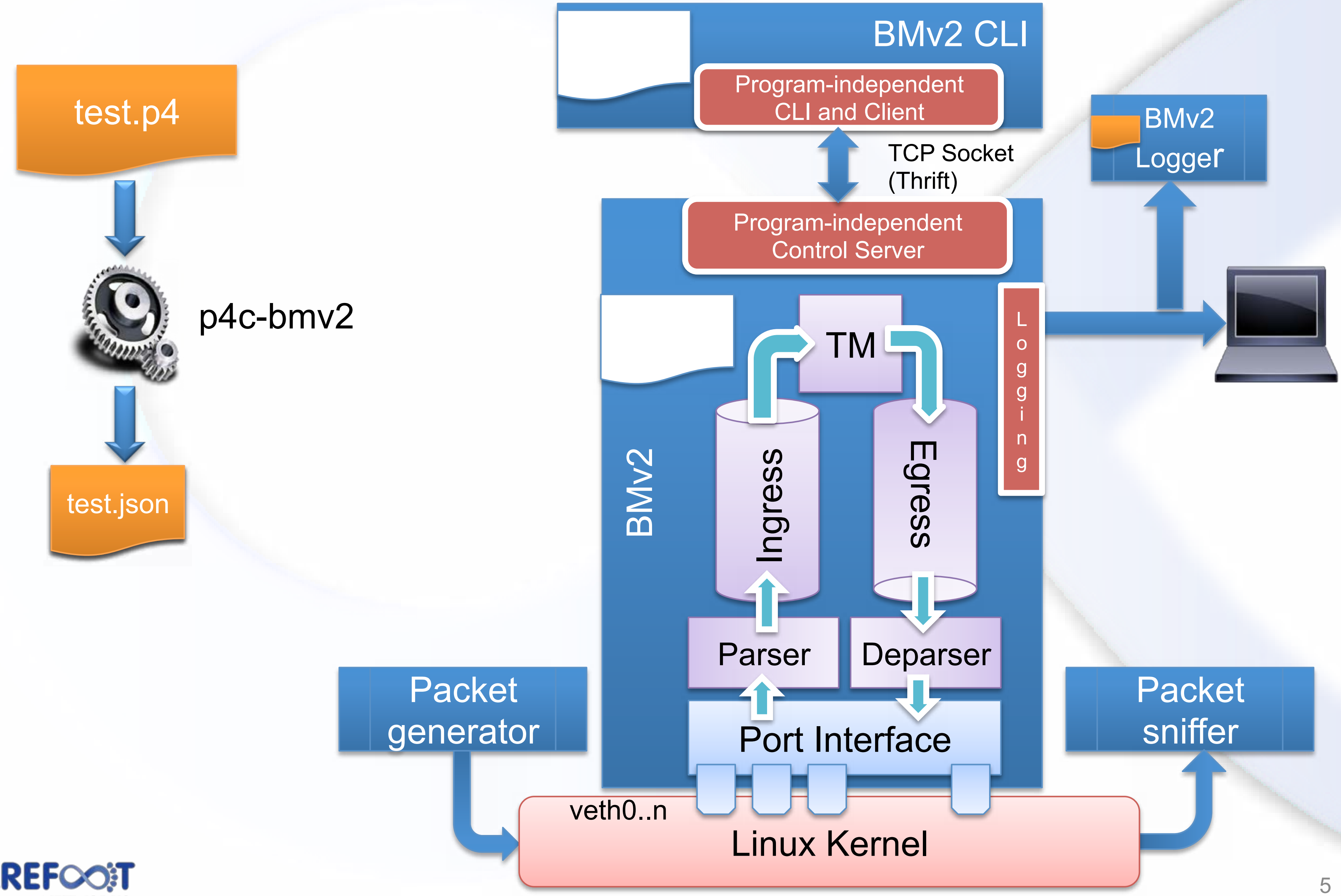
- **Operating System: Ubuntu 14.04**
- **Shipped as OVA (Open Virtual Appliance)**
- **Needs to be imported into Virtual Box**
  - File → Import Appliance…
- **Download link**
  - https://drive.google.com/file/d/0BxHYRsv-PNVvaEpZYnJ5LTAyZkk
- **User Name: ubuntu**
- **Password: ubuntu**

- **Do not forget to install VirtualBox additions!**
  - Devices → Insert Guest Additions CD Image…

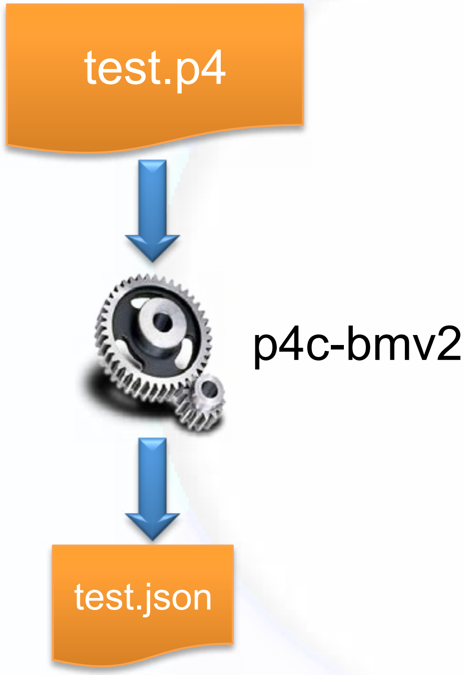
Basic Workflow

- How Everything “clicks” together

Basic Workflow

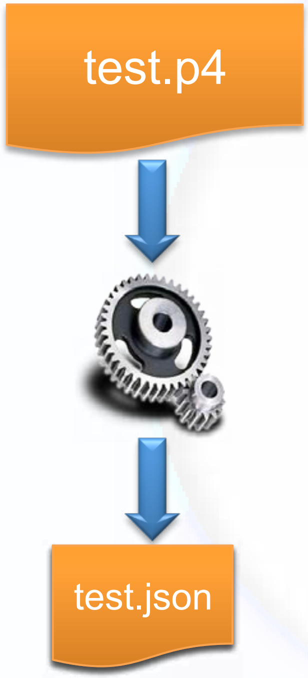


Step 1: P4 Program Compilation



```
$ p4c-bmv2 --json test.json test.p4
```

Step 2: Preparing veth Interfaces

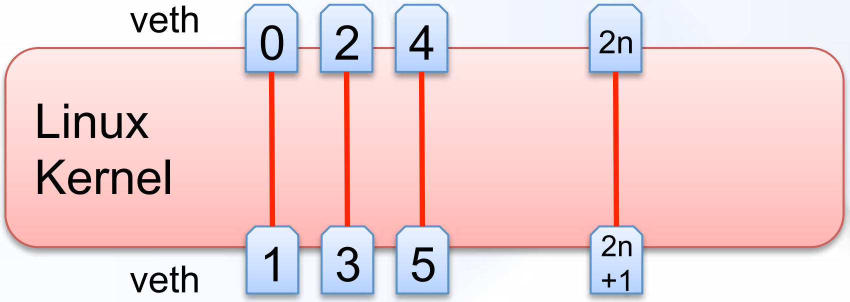


test.p4

test.json

```
$ sudo ~/tutorial/examples/veth_setup.sh

# ip link add name veth0 type veth peer name veth1
# for iface in “veth0 veth1”; do
     ip link set dev ${iface} up
     sysctl net.ipv6.conf.${iface}.disable_ipv6=1
     TOE_OPTIONS="rx tx sg tso ufo gso gro lro rxvlan txvlan rxhash”
     for TOE_OPTION in $TOE_OPTIONS; do
         /sbin/ethtool --offload $intf "$TOE_OPTION”
     done
  done
```



veth

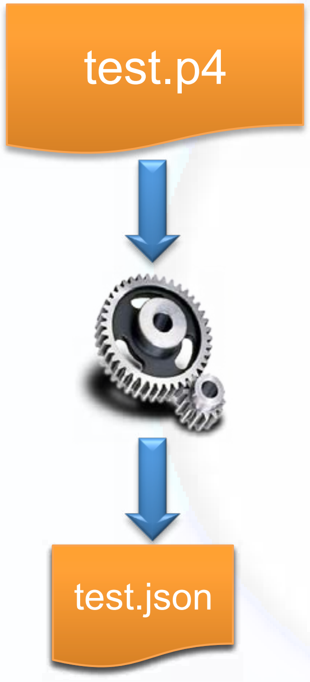
Linux Kernel

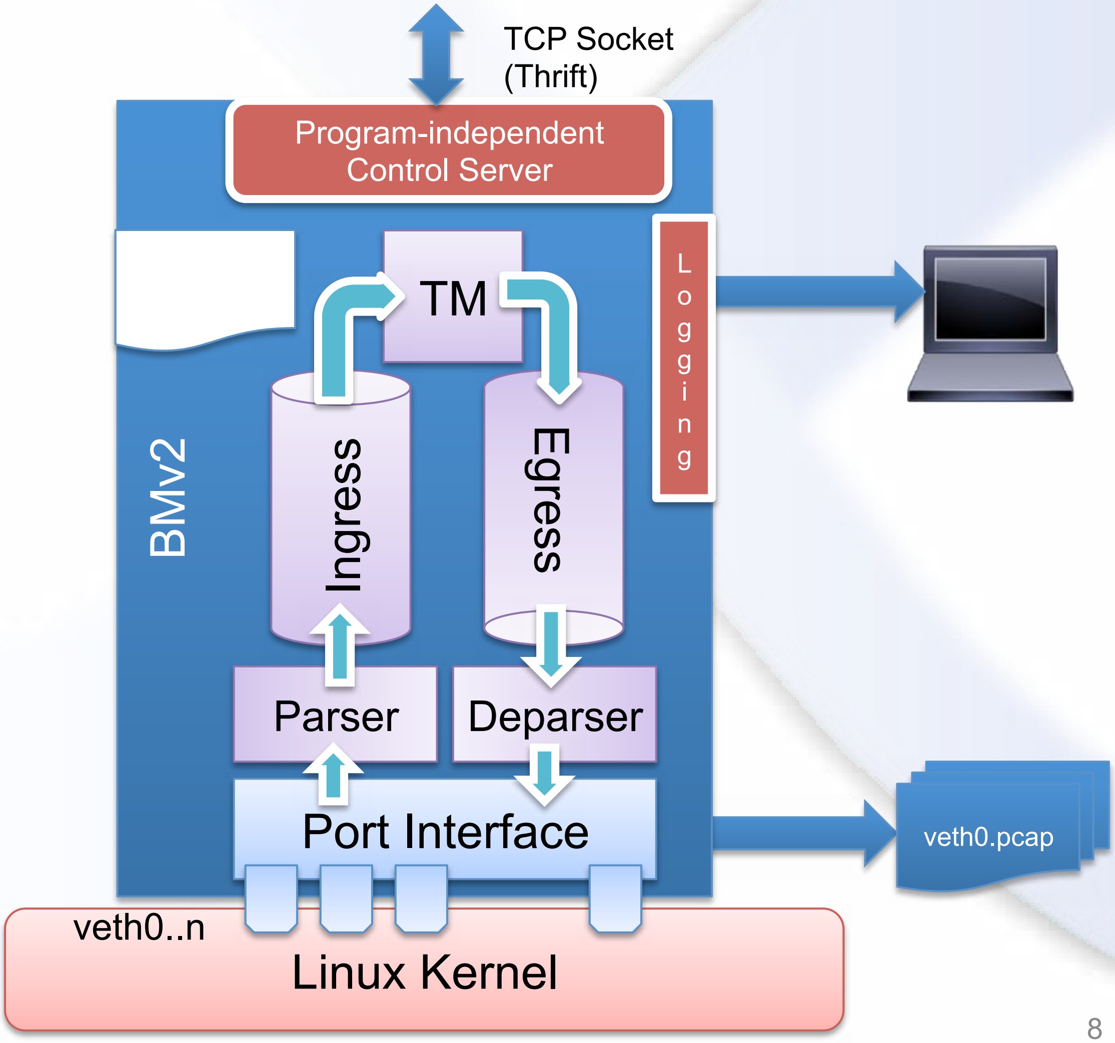
veth

BAREFOOT NETWORKS

Step 3: Starting the model

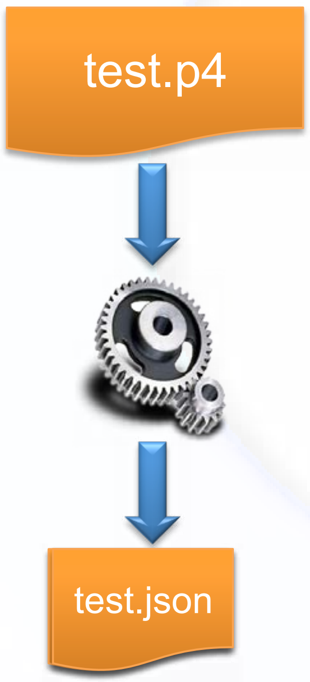
```
$ sudo simple_switch test.json  --log-console  \
       -i 0@veth0 -i 1@veth2 …                 \
       --thrift-port 9090
```

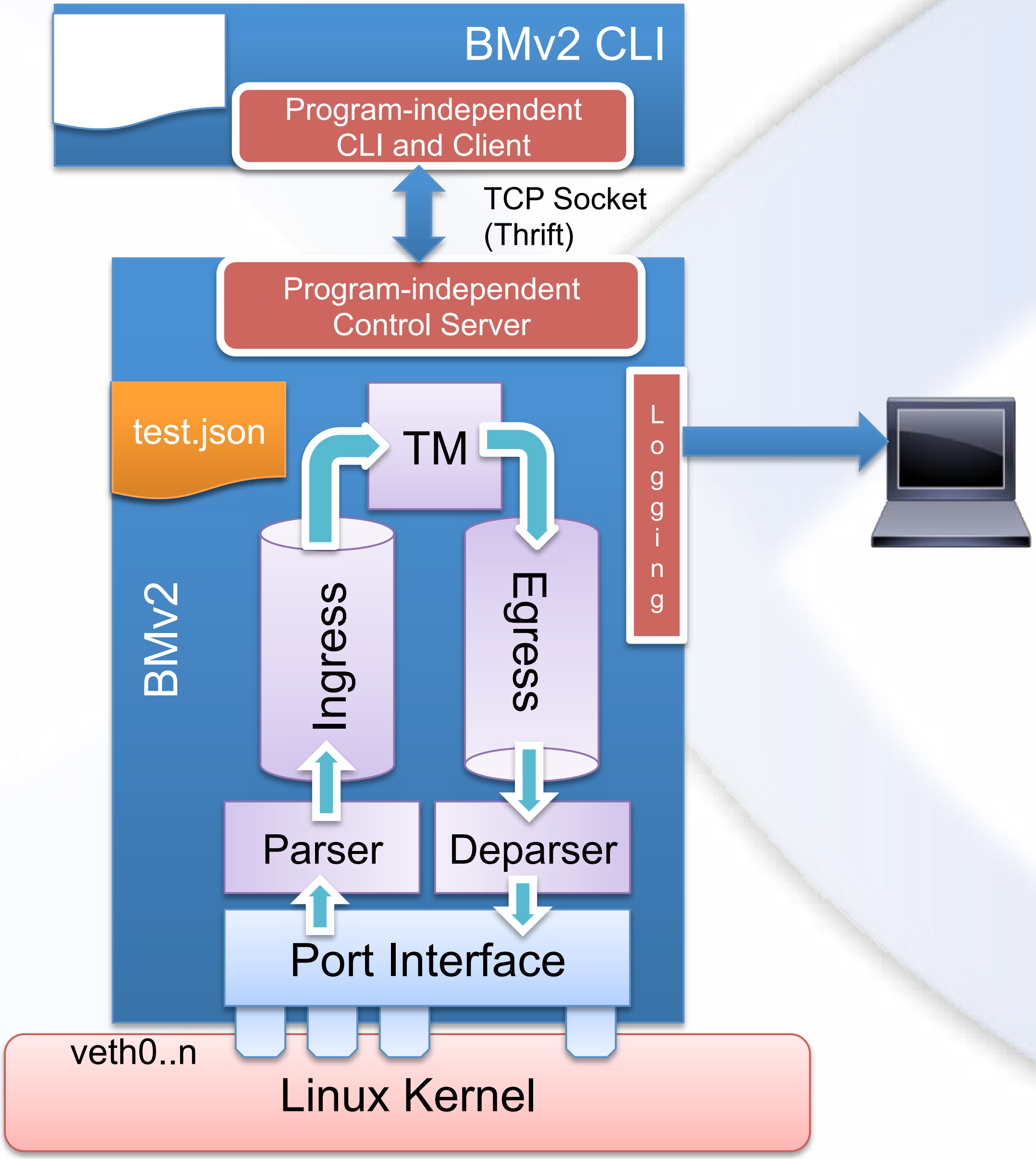




Step 4: Starting the CLI

```
$ sswitch_CLI --json test.json
```





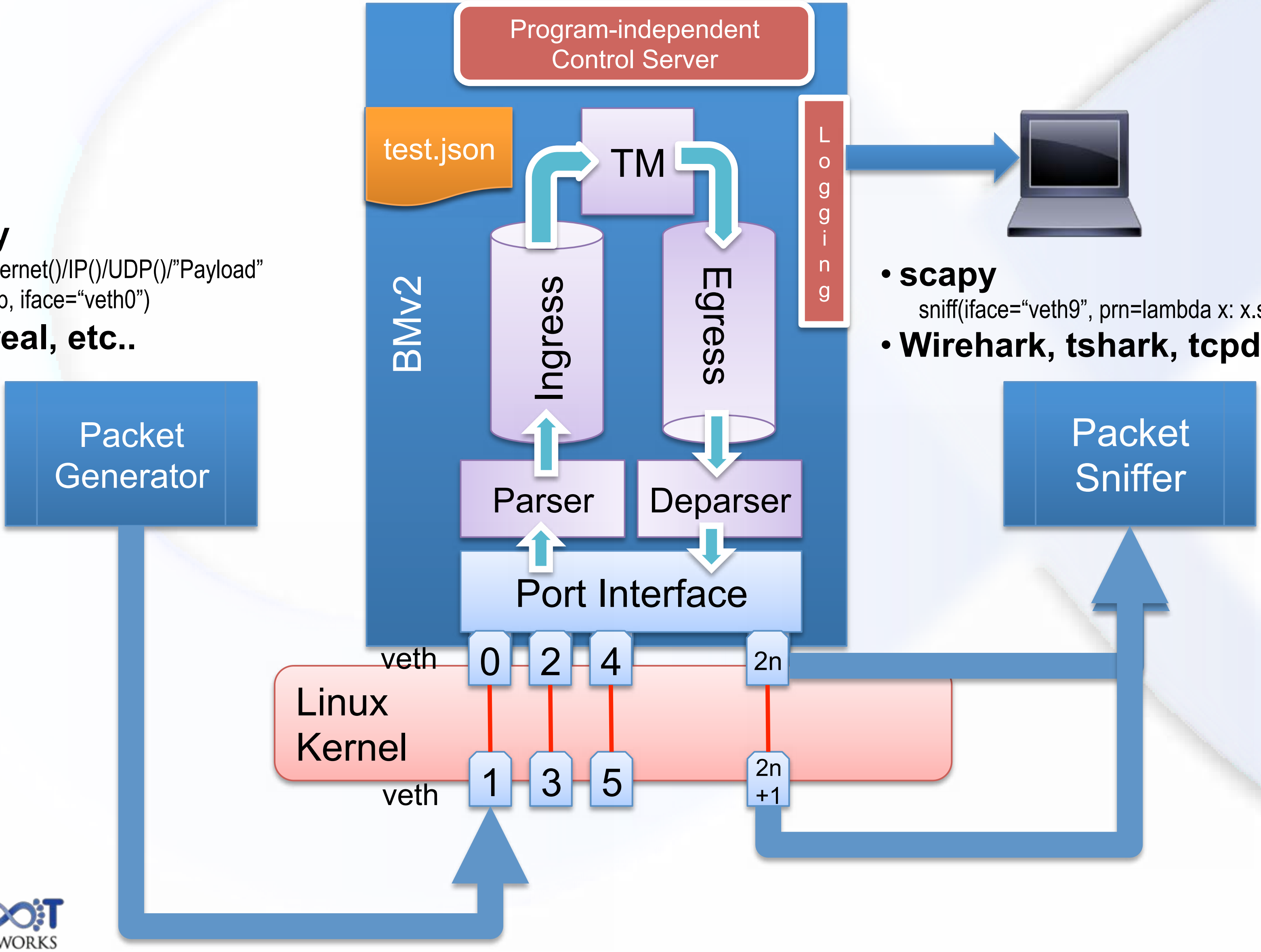
Step 5: Sending and Receiving Packets

- **scapy**

  p = Ethernet()/IP()/UDP()/"Payload"
  sendp(p, iface="veth0")
- **Ethereal, etc..**

- **scapy**

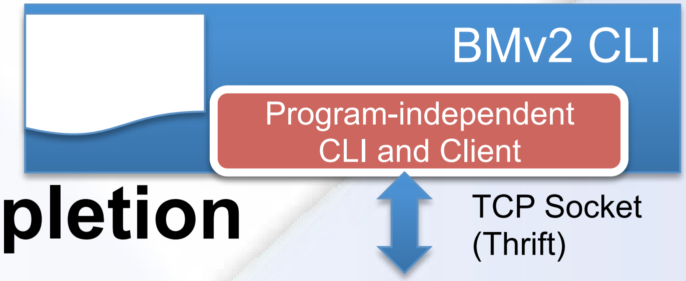
  sniff(iface="veth9", prn=lambda x: x.show())
- **Wirehark, tshark, tcpdump**



Using the CLI

- **Programming the device**

Basic Info



BMv2 CLI

Program-independent CLI and Client

TCP Socket (Thrift)

- **Simple CLI written in Python**
  - Based on the standard `cmd` module
- **Interactive shell with autocompletion**
- **Simple scripting**
  - Feed a list of commands on STDIN
- **Generic commands for various P4 objects**
  - P4 object definitions are loaded from the JSON file
- **Additional commands for the fixed APIs**
- **No state**
  - Simple translation of commands to Thrift messages
  - Can be restarted (or crashed) without disturbing the model
- **Multiple instances can be started**
  - To communicate with multiple models via separate connections

Getting Help

Getting the list of commands

```
RuntimeCmd: help
Documented commands (type help <topic>):
========================================
counter_read           show_tables
counter_reset          swap_configs
help                   table_add
load_new_config_file   table_delete
mc_mgrp_create         table_dump
mc_mgrp_destroy        table_indirect_add
mc_node_associate      table_indirect_add_member_to_group
mc_node_create         table_indirect_add_with_group
mc_node_destroy        table_indirect_create_group
mc_node_dissociate     table_indirect_create_member
mc_node_update         table_indirect_delete
mc_set_lag_membership  table_indirect_delete_group
meter_set_rates        table_indirect_delete_member
mirroring_add          table_indirect_modify_member
mirroring_delete       table_indirect_remove_member_from_group
register_read          table_indirect_set_default
register_write         table_indirect_set_default_with_group
set_queue_depth        table_info
set_queue_rate         table_modify
shell                  table_set_default
show_actions           table_show_actions
```

Getting the command help

```
RuntimeCmd: help table_add
Add entry to a match table:
table_add <table name> <action name> <match fields> => <action parameters> [priority]
```

Working with Tables

```
RuntimeCmd: show_tables
m_filter                       [meta.meter_tag(exact, 32)]
m_table                        [ethernet.srcAddr(ternary, 48)]

RuntimeCmd: table_info m_table
m_table                        [ethernet.srcAddr(ternary, 48)]
********************************************************************************
_nop
[]m_action                       [meter_idx(32)]

RuntimeCmd: dump_table m_table
m_table:
0: aaaaaaaaaaaa &&& ffffffffffff => m_action - 0,
SUCCESS

RuntimeCmd: table_add m_table m_action 01:00:00:00:00:00&&&01:00:00:00:00:00 => 1 0
Adding entry to ternary match table m_table
match key:           TERNARY-01:00:00:00:00:00 &&& 01:00:00:00:00:00
action:              m_action
runtime data:        00:00:00:05
SUCCESS
entry has been added with handle 1

RuntimeCmd: table_delete 1
```

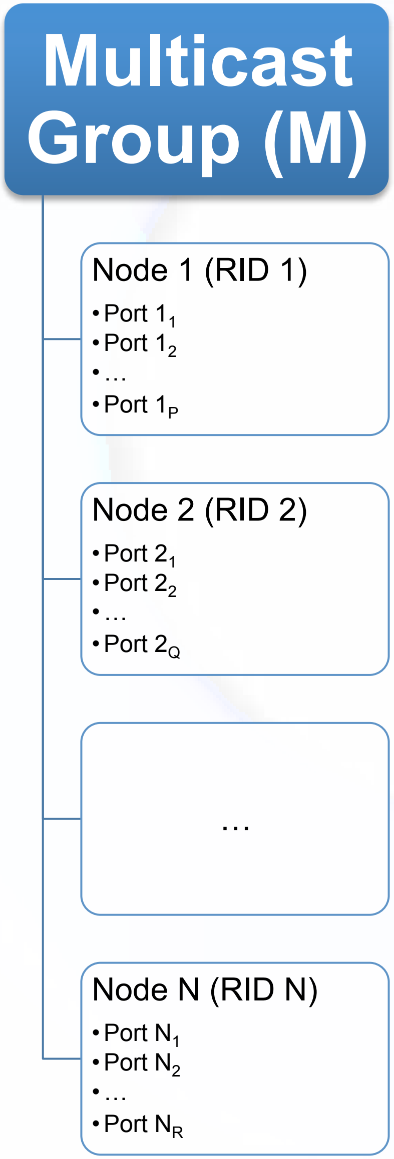
Value and mask for ternary matching. No spaces around “&&&”

Entry priority

“=>” separates the key from the action data

All subsequent operations use the entry handle

Packet Replication (Multicast)



Multicast Group (M)

Node 1 (RID 1)

- Port 1_1
- Port 1_2
- …
- Port 1_P

Node 2 (RID 2)

- Port 2_1
- Port 2_2
- …
- Port 2_Q

…

Node N (RID N)

- Port N_1
- Port N_2
- …
- Port N_R

```
RuntimeCmd: mc_mgrp_create 1
Creating multicast group 1
SUCCESS

RuntimeCmd: mc_node_create 10    1 2 3 4 5
Creating node with rid 10 , port map 111110 and lag map
SUCCESS
node was created with handle 1

RuntimeCmd: mc_node_create 12    10 9 4 6
Creating node with rid 12 , port map 10111000000000 and lag
map
SUCCESS
node was created with handle 2

RuntimeCmd: mc_node_associate 1 1
Associating node 1 to multicast group 1
SUCCESS

RuntimeCmd: mc_node_associate 1 2
Associating node 2 to multicast group 1
SUCCESS
```

A node can also be associated with multiple multicast groups

Managing Mirror Destinations

- **Mirror Destinations (Clone Specs) are used by P4 primitive actions:**
  - clone_ingress_pkt_to_ingress(clone_spec, field_list)
  - clone_ingress_pkt_to_egress(clone_spec, field_list)
  - clone_egress_pkt_to_ingress(clone_spec, field_list)
  - clone_egress_pkt_to_egress(clone_spec, field_list)
- **Clone spec is an integer number, representing a “special destination”**

```
RuntimeCmd: mirroring_add 12345 2
```

  - Packets set to clone spec 12345 will go to the switch port #2
  - Typical application: designating a certain port for CPU

Scapy – Packet Sniffer and Generator

- **Free Software**
  - ○ http://www.secdev.org/projects/scapy/
- **Implemented in Python**
- **Can be imported as a module**
- **Extensible**
  - ○ New packet formats are easily defined
- **Easy to use**
  - ○ Reasonable defaults everywhere
  - ○ Simple Syntax

Simple Examples

- **Creating a packet**
  - p = Ether()/Dot1Q()/IP()/UDP()/(“A” * 64)
  - p = Ether(src=“00:00:00:00:00:01”, dst=“ff:ff:ff:ff:ff:ff”) /
    Dot1Q(pri=6, vlan=23) /
    IP(src=“192.168.1.1”, dst=“192.168.1.255”) /
    UDP(sport=7, dport=7) / “Vladimir”
- **Packet display**
  - p.show()
  - p.show2()
  - hexdump(p)
- **Sending the packet**
  - sendp(p, iface=“eth0”, [count=100])
- **Sniffing**
  - sniff(iface=“eth0”, prn=hexdump)
  - sniff(iface=“eth0”, prn=lambda p: p.show())

Thank you