阅读 p4-guide

摘要

这篇文章带你进入p4 guide 的大门

https://github.com/anonymity12/p4-guide

并完成demo1 的实验，后续 实验在其他文章中。

本文包括：

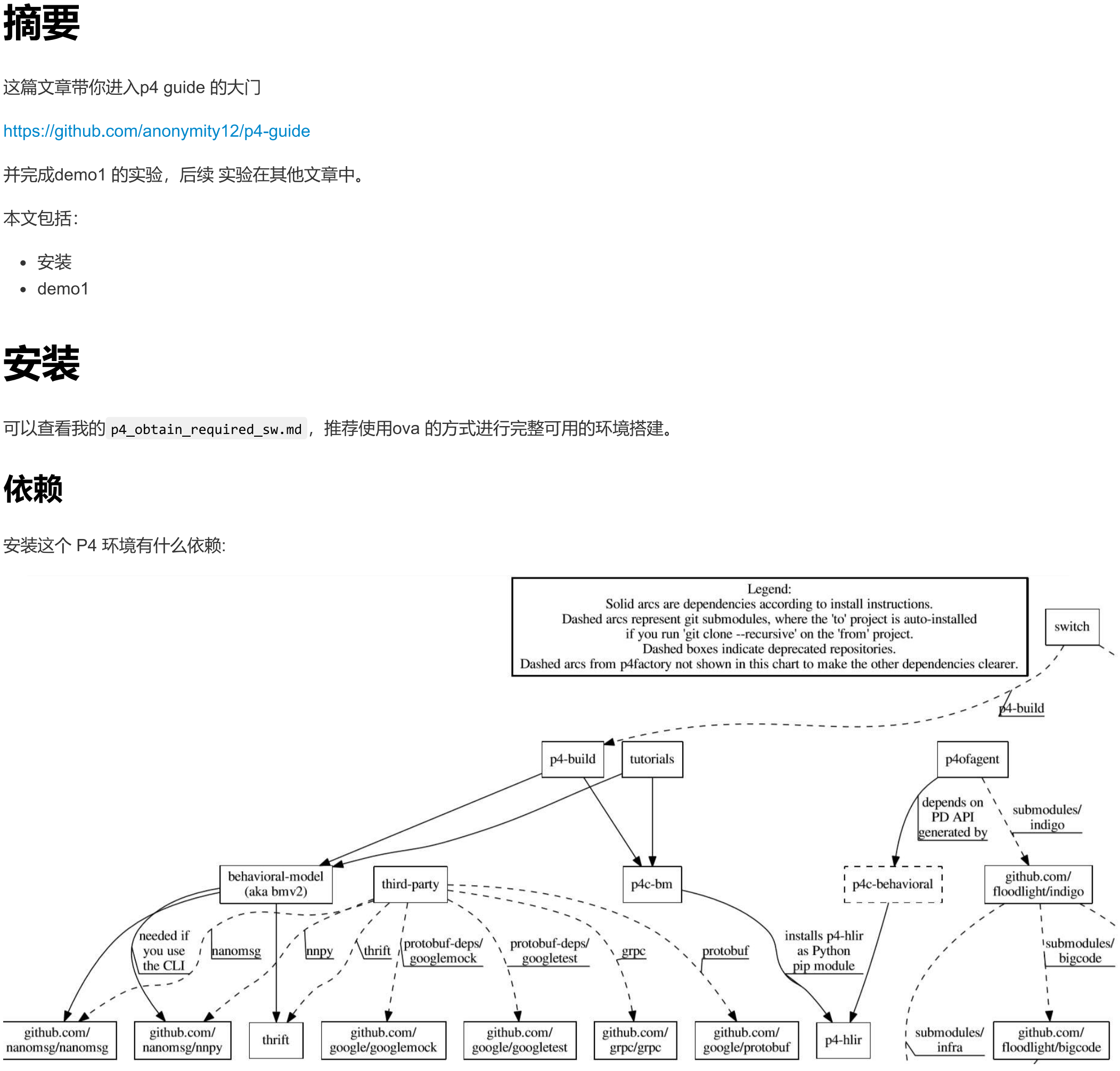
- 安装
- demo1

安装

可以查看我的`p4_obtain_required_sw.md`，推荐使用ova 的方式进行完整可用的环境搭建。

依赖

安装这个 P4 环境有什么依赖:



祥见： assets/dependencies.pdf

demo

https://github.com/anonymity12/p4-guide/blob/master/README-demos.md

demo1 简单的交换机

demo1 在做什么

在 ingress 方向

- 解析以太网头部，和ipv4 头部
- 进行LPM，得到l2ptr
- 利用l2ptr 得到出口的mac 和端口
- 减TTL

在 egress 方向

- 根据出口找 新的smac
- 重新计算ipv4 头部的checksum

https://github.com/anonymity12/p4-guide/blob/master/demo1/README.md

在这片文章里，我们讨论了：

- 编译源代码
- 运行，并要学会使用bmv2 行为模型
- 使用simple-switch 来建立一个虚拟的交换机
- 使用simple switch 的CLI， 并实践`table_add`命令，（这里的simple switch 是一个py session，扮演control panel的角色。）
- 并学习使用py 发包库 scapy 发送报文，用来验证交换机实际加载到的表

我们可以首先看一看 src code： `demo1.p4_16.p4`

https://github.com/anonymity12/p4-guide/blob/master/demo1/demo1.p4_16.p4

有这样的说法：

> P4编译生成配置和API，配置下发到硬件，然后API通过控制平面调动来下发表项。

这个代码 编译 完成后，就是生成了配置，也就是 （配置了/定义了） 匹配什么表，匹配之后，做什么action。

simple switch 进行的就是控制平面的下发 表项 行为

---

demo1中值得注意的，以及我学到的

demo1.p4_14.p4 or demo1.p4_16.p4 (same commands work for both)

这个部分是创建两张表，可以看到创建一个转发表，实际上有三个子表：

ipv4的，目的mac，出口源mac的

```
table_set_default ipv4_da_lpm my_drop
table_set_default mac_da my_drop
table_set_default send_frame my_drop

table_add ipv4_da_lpm set_l2ptr 10.1.0.1/32 => 58
table_add mac_da set_bd_dmac_intf 58 => 9 02:13:57:ab:cd:ef 2
table_add send_frame rewrite_mac 9 => 00:11:22:33:44:55
```

table 的名字

action

哪个key可以匹配到此action

action发生后的附加信息，比如nhop信息

这里是表格1

Another set of table entries to forward packets to a different output interface:

```
# Version with dotted decimal IPv4 address and : separators inside
# of hexadecimal Ethernet addresses.
table_add ipv4_da_lpm set_l2ptr 10.1.0.200/32 => 81
table_add mac_da set_bd_dmac_intf 81 => 15 08:de:ad:be:ef:00 4
table_add send_frame rewrite_mac 15 => ca:fe:ba:be:d0:0d

# Version with hex values instead of the above versions.
# Note: the prefix length after the / character must be decimal.
# I tried 0x20 and simple_switch_CLI raised an exception and
# exited.
table_add ipv4_da_lpm set_l2ptr 0x0a0100c8/32 => 0x51
table_add mac_da set_bd_dmac_intf 0x51 => 0xf 0x08deadbeef00 0x4
table_add send_frame rewrite_mac 0xf => 0xcafebabed00d
```

这里是表格2

可以看到每个表格的添加都由 3个 table_add 语句构成

注： 添加表格2 的另一种形式，16进制表达形式

demo1 中无法做到的：

demo1

A very simple program that only does these things:

- on ingress:
  - parse Ethernet and IPv4 headers (ignoring IP options)
  - perform a longest prefix match on the IPv4 DA, resulting in an 'l2ptr' value
  - l2ptr is an exact match index into a mac_da table, resulting in output BD (bridge domain), a new destination MAC address, and the output port.
  - decrement the IPv4 TTL field
- on egress:
  - look up the output port number to get a new source MAC address
  - calculate a new IPv4 header checksum

See the instructions in the file `README.md` if you want to use `simple_switch` using the Thrift API for communicating from a simple CLI "controller" process.

Use the alternate instructions in `README-p4runtime.md` if you want to use `simple_switch_grpc`. You may still use the Thrift API, but `simple_switch_grpc` was created with the intent of using the newer P4Runtime API for communicating from a controller process (in the example run, it is an interactive Python session acting as the controller).

想使用比较新的： simple_switch_grpc ， 可以通过 互动的 python session， 但是发现 如下错误： （python 的 `sys.path` 似乎不存在 我们的p4 python 库，导致的）

```
>>> import base_test as bt
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "base_test.py", line 43, in <module>
    from p4.v1 import p4runtime_pb2
ImportError: No module named v1
```

于是采用旧的： simple_switch， 此时参考

https://github.com/anonymity12/p4-guide/blob/master/demo1/README.md

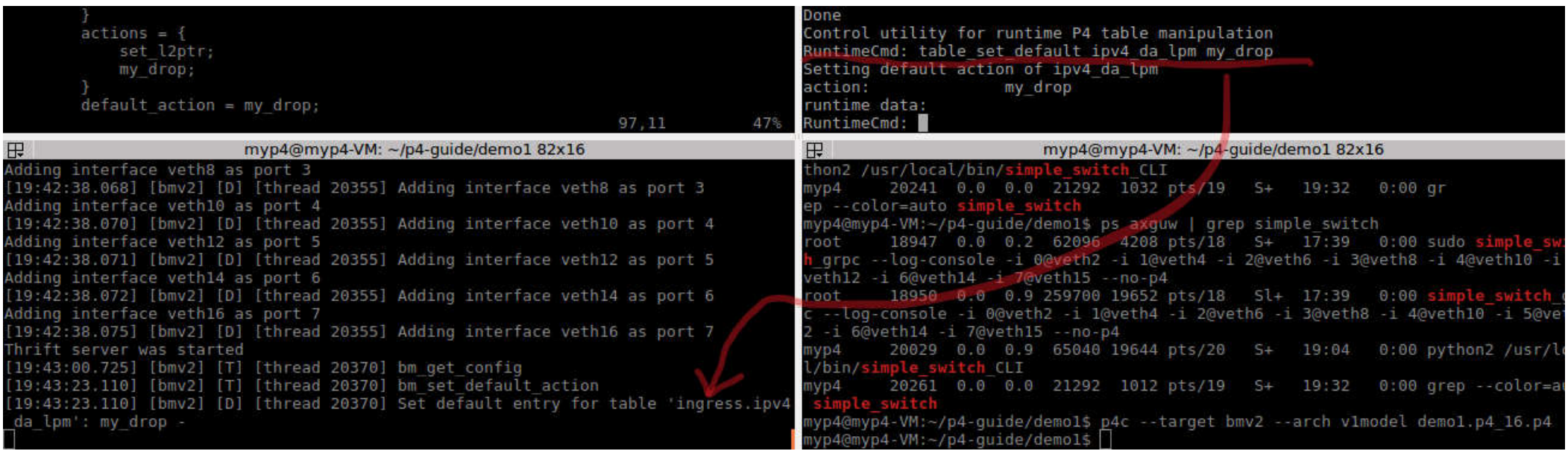
进行实验即可

数据层面添加table实验时的截图

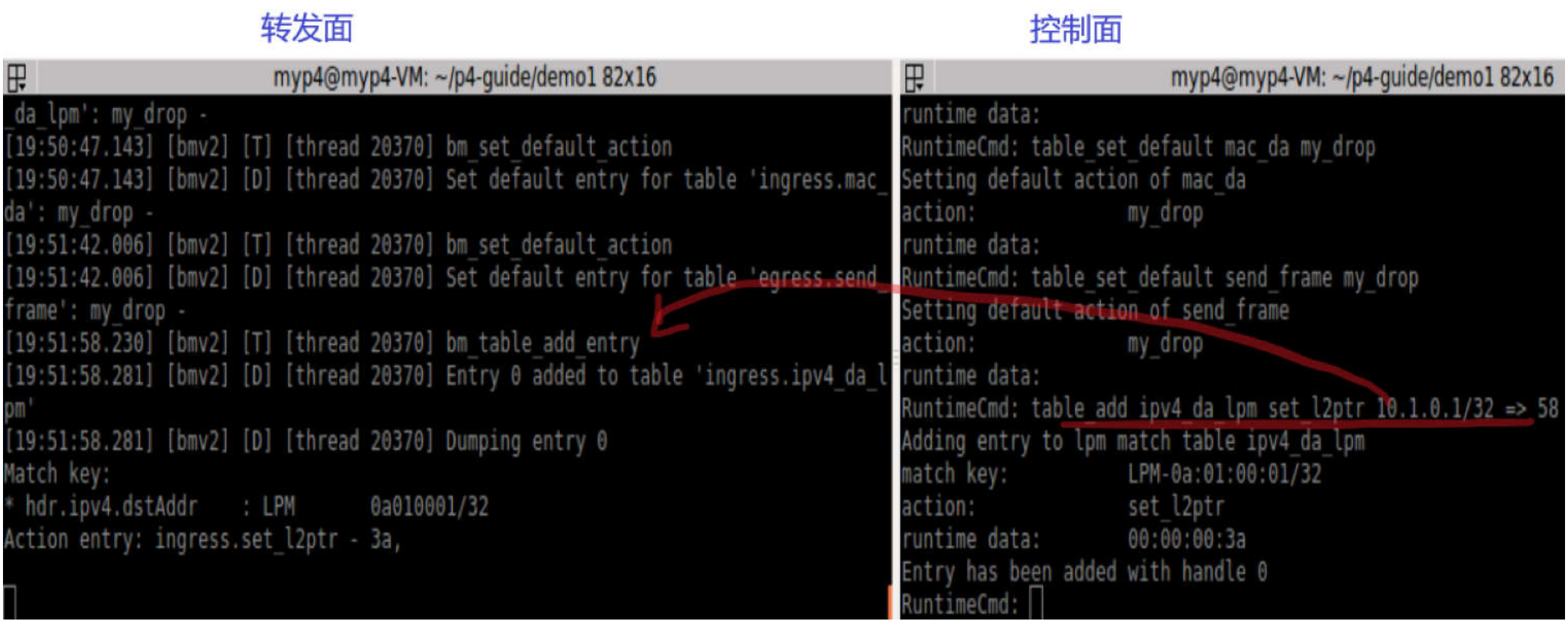
开启模拟器的控制层面的命令： `simple_switch_CLI`

然后在此cli 中执行 `table_add` 命令，控制层面下发 将会下发这个add的表项。

添加一个table 的 default action 成功



添加一个table 的 entry 成功



这个entry 的意思是：

- 我属于 table： ipv4_da_lpm
- 我包含一个key： 10.1.0.1/32
- 如果我发现有 什么匹配到了上述这个key 10.1.0.1/32 那么我们的action是 `set_l2ptr`
- 在执行action： `set_l2ptr` 时， 我会给action一个参数： `l2ptr` , 并且 `l2ptr` 的值是 `58` （ 这个 `58` 会被用作下一个表中的key）

也就是说table_add 的语法是：

```
table_add table_name action key => action_data
```

- [x] 0116 scapy in p4 suite vbox

添加表项之后，发送报文 by scapy

使用 scapy 的代码如下：

```
sudo scapy

fwd_pkt1=Ether() / IP(dst='10.1.0.1') / TCP(sport=5793, dport=80)
drop_pkt1=Ether() / IP(dst='10.1.0.34') / TCP(sport=5793, dport=80)

# Send packet at layer2, specifying interface
sendp(fwd_pkt1, iface="veth2")
sendp(drop_pkt1, iface="veth2")

fwd_pkt2=Ether() / IP(dst='10.1.0.1') / TCP(sport=5793, dport=80) / Raw('The quick brown fox jumped over the lazy dog.')
sendp(fwd_pkt2, iface="veth2")
```

```
43 [20:15:29.883] [bmv2] [T] [thread 20361] [0.0] [cxt 0] demo1.p4_16.p4(110) Primitive hdr.ethernet.dstAddr = dma
   c
44 [20:15:29.883] [bmv2] [T] [thread 20361] [0.0] [cxt 0] demo1.p4_16.p4(111) Primitive standard_metadata.egress_s
   pec = intf
45 [20:15:29.883] [bmv2] [T] [thread 20361] [0.0] [cxt 0] demo1.p4_16.p4(112) Primitive hdr.ipv4.ttl = hdr.ipv4.tt
   l - 1
46 [20:15:29.883] [bmv2] [D] [thread 20361] [0.0] [cxt 0] Pipeline 'ingress': end
47 [20:15:29.883] [bmv2] [D] [thread 20361] [0.0] [cxt 0] Egress port is 2
48 [20:15:29.883] [bmv2] [D] [thread 20364] [0.0] [cxt 0] Pipeline 'egress': start
49 [20:15:29.883] [bmv2] [T] [thread 20364] [0.0] [cxt 0] Applying table 'egress.send_frame'
50 [20:15:29.883] [bmv2] [D] [thread 20364] [0.0] [cxt 0] Looking up key:
51 * meta.fwd_metadata.out_bd: 000009
52
53 [20:15:29.883] [bmv2] [D] [thread 20364] [0.0] [cxt 0] Table 'egress.send_frame': hit with handle 0
```

我们可以从上图（发送 到一个ip， which table key 中有 此目的ip的） 看到，simple_switch 的log 中，有明显的data panel的 变动，比如 我们在p4 src code 里写的table， action 都 有被回调。

对于没有目标ip 的报文，因为数据层面没有这个entry， 则会出现`miss`, 如下，两个表都 出现了miss； `ipv4_da_lpm`，`mac_da`

ps: 这两个表自动被加上 `ingress`的前缀，因为这两张表都是在 `ingress` 的流水线里进行apply 的。

```
[cxt 0] Table 'ingress.ipv4_da_lpm': miss
[cxt 0] Action entry is my_drop -
[cxt 0] Action my_drop
[cxt 0] demo1.p4_16.p4(64) Primitive mark_to_drop()
[cxt 0] Applying table 'ingress.mac_da'
[cxt 0] Looking up key:


[cxt 0] Table 'ingress.mac_da': miss
[cxt 0] Action entry is my_drop -
[cxt 0] Action my_drop
[cxt 0] demo1.p4_16.p4(64) Primitive mark_to_drop()
[cxt 0] Pipeline 'ingress': end
```

---

ps 后记

利用p4c进行编译的完整代码

```
p4c --target bmv2 --arch v1model --p4runtime-file demo1.p4_16.p4rt.txt --p4runtime-format text demo1.p4_16.p4
```

这 会有如下的`OUTPUT`:

demo1.p4_16.p4i

demo1.p4_16.json

demo1.p4_16.p4rt.txt

p4i 是预处理的中间文件，对于runtime 没有用。

simple_switch_grpc 与 行为表现模型`bm` 有关

```
sudo simple_switch_grpc --log-file ss-log --log-flush -i 0@veth2 -i 1@veth4 -i 2@veth6 -i 3@veth8 -i 4@veth10 -i 5@veth12 -i 6@veth14 -i 7@veth16 --no-p4
```

你会用到虚拟的以太网接口： `veth`

备用名词

用于和 controller 进行沟通的接口 API

- older Thrift API
- newer P4Runtime API