P4上手实验文档

傅亮

一. 了解P4语言

- P4 学习笔记（一）- 导论
- P4 学习笔记（二）- 基础语法和 Parser
- P4 学习笔记（三）- 控制逻辑与完整的工作流
- P416 Language Specification

二. 搭建简单的P4实验环境

所需组件：

1. behavioral-model(P4交换机的软件模拟环境)
2. p4c(P4语言编译器)
3. tutorials(包含P4教程)

1. 安装流程

首先准备一台ubuntu20.04的虚拟机，硬盘推荐20G以上

1.1 安装p4c

在ubuntu20.04中，直接使用以下命令安装p4c

```
. /etc/os-release
echo "deb http://download.opensuse.org/repositories/home:/p4lang/xUbuntu_${VERSI
curl -L "http://download.opensuse.org/repositories/home:/p4lang/xUbuntu_${VERSIO
sudo apt-get update
sudo apt install p4lang-p4c
```

1.2 安装behavioral-model

这里推荐手动安装，behavioral-model中有详细的安装教程，这里仅做简要介绍。

首先安装运行所需的工具包

```
sudo apt-get install -y automake cmake libgmp-dev \
    libpcap-dev libboost-dev libboost-test-dev libboost-program-options-dev \
    libboost-system-dev libboost-filesystem-dev libboost-thread-dev \
    libevent-dev libtool flex bison pkg-config g++ libssl-dev
```

接着安装thrift和nanomsg，直接运行behavioral-model/ci/目录下的install-nanomsg.sh和install-thrift.sh脚本即可。

所需组件安装完成后，开始编译behavioral-model

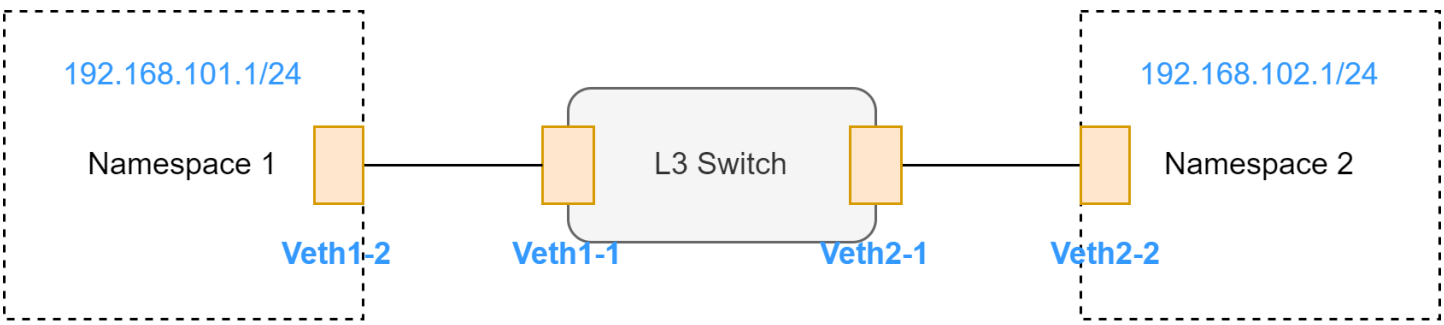
```
./autogen.sh
./configure
make
[sudo] make install  # if you need to install bmv2
```

1.3 （可选）安装P4 tutorials

在基本运行环境安装完成以后，可以试着下载并运行P4 tutorials中的各种P4 demo，理解P4不同场景下的应用。

2. 搭建网络拓扑

在本次实验中，我们将用P4实现一个简单的L3交换机，为了方便网络拓扑的构建，我们使用namespace和veth将发包端，收包端和交换机实现在同一台虚机上，网络拓扑图如下：



2.1 搭建流程

- 新建veth

```
sudo ip link add veth1-1 type veth peer name veth1-2
sudo ip link add veth2-1 type veth peer name veth2-2
```

- 新建namespace：ns1、ns2

```
sudo ip netns add ns1
sudo ip netns add ns2
```

- 将veth1-2放进ns1，veth2-2放进ns2

```
sudo ip link set veth1-2 netns ns1
sudo ip link set veth2-2 netns ns2
```

- 配置IP

```
sudo ip addr add 192.168.101.254/24 dev veth1-1
sudo ip addr add 192.168.102.254/24 dev veth2-1
sudo ip netns exec ns1 ip addr add 192.168.101.1/24 dev veth1-2
sudo ip netns exec ns2 ip addr add 192.168.102.1/24 dev veth2-2
```

- 启动veth

```
sudo ip link set up veth1-1
sudo ip link set up veth2-1
sudo ip netns exec ns1 ip link set up veth1-2
sudo ip netns exec ns2 ip link set up veth2-2
```

- 配置路由

```
sudo ip netns exec ns1 ip route add 0.0.0.0/0 via 192.168.101.254
sudo ip netns exec ns2 ip route add 0.0.0.0/0 via 192.168.102.254
```

配置完成，此时拓扑图中的L3 switch并没有运行，所以从ns1是ping不通ns2的，可以使用以下命令测试：

```
ip netns exec ns1 ping 192.168.102.1
```

3. 开始实验

3.1 Switch代码

创建一个.p4后缀的文件，并向其中添加代码

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

const bit<16> TYPE_IPV4 = 0x800;

/*************************************************************************
*********************** H E A D E R S  ***********************************
*************************************************************************/

typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}

struct metadata {
    /* empty */
}

struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
}

```

```
/*************************************************************************
*********************** P A R S E R  *******************************
*************************************************************************/

parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }

}

/*************************************************************************
************   C H E C K S U M    V E R I F I C A T I O N  *************
*************************************************************************/

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}


/*************************************************************************
**************  I N G R E S S  P R O C E S S I N G  *******************
*************************************************************************/

control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }
```

```
92
93      action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
94          standard_metadata.egress_spec = port;
95          hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
96          hdr.ethernet.dstAddr = dstAddr;
97          hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
98      }
99
100     table ipv4_lpm {
101         key = {
102             hdr.ipv4.dstAddr: lpm;
103         }
104         actions = {
105             ipv4_forward;
106             drop;
107             NoAction;
108         }
109         size = 1024;
110         default_action = drop();
111     }
112
113     apply {
114         if (hdr.ipv4.isValid()) {
115             ipv4_lpm.apply();
116         }
117     }
118 }
119
120 /*************************************************************************
121 ***************  E G R E S S   P R O C E S S I N G   *******************
122 *************************************************************************/
123
124 control MyEgress(inout headers hdr,
125                  inout metadata meta,
126                  inout standard_metadata_t standard_metadata) {
127     apply {  }
128 }
129
130 /*************************************************************************
131 *************   C H E C K S U M    C O M P U T A T I O N   **************
132 *************************************************************************/
133
134 control MyComputeChecksum(inout headers  hdr, inout metadata meta) {
135      apply {
136         update_checksum(
137         hdr.ipv4.isValid(),
138             { hdr.ipv4.version,
```

```
139                 hdr.ipv4.ihl,
140                 hdr.ipv4.diffserv,
141                 hdr.ipv4.totalLen,
142                 hdr.ipv4.identification,
143                 hdr.ipv4.flags,
144                 hdr.ipv4.fragOffset,
145                 hdr.ipv4.ttl,
146                 hdr.ipv4.protocol,
147                 hdr.ipv4.srcAddr,
148                 hdr.ipv4.dstAddr },
149             hdr.ipv4.hdrChecksum,
150             HashAlgorithm.csum16);
151     }
152 }
153
154 /*************************************************************************
155 ***********************  D E P A R S E R  ***************************
156 *************************************************************************/
157
158 control MyDeparser(packet_out packet, in headers hdr) {
159     apply {
160         packet.emit(hdr.ethernet);
161         packet.emit(hdr.ipv4);
162     }
163 }
164
165 /*************************************************************************
166 ***********************  S W I T C H  *******************************
167 *************************************************************************/
168
169 V1Switch(
170 MyParser(),
171 MyVerifyChecksum(),
172 MyIngress(),
173 MyEgress(),
174 MyComputeChecksum(),
175 MyDeparser()
176 ) main;
```

3.2 编译并运行switch

```
1 p4c --target bmv2 --arch v1model --std p4-16 xxx.p4
```

编译完成后，会在当前目录下生成名为xxx.json的文件，接下来即可使用behavioral-model运行此文件，请自行将xxx.json拷贝至simple_switch目录下

```
cd path_to_behavioral-model/target/simple_switch
simple_switch -i 1@veth1-1 -i 2@veth2-1 --thrift-port 9090 --nanolog ipc:///tmp/bm-1-log.ipc xxx.json --log-console
```

此时swich已经启动，接下来我们继续配置Swich的转发规则。

新开一个终端，运行以下命令

```
cd path_to_behavioral-model/target/simple_switch
simple_switch_CLI --thrift-port 9090
```

输入转发规则

```
table_add ipv4_lpm ipv4_forward 192.168.101.1/32 => xx:xx:xx:xx:xx:xx 1
table_add ipv4_lpm ipv4_forward 192.168.102.1/32 => xx:xx:xx:xx:xx:xx 2
```

（高亮的两处需要分别替换为veth1-2和veth2-2的mac地址，mac地址可使用命令ip netns exec ns1 ip a 查看，ns2同理）

3.3 测试连通性

再次用ns1 ping ns2，此时即可正常ping通，命令如下：

```
ip netns exec ns1 ping 192.168.102.1
```

可以检查simple_switch的输出，理解switch在转发中的工作流程