EA 080 - Laboratório de Redes
Atividade de Aula 3 - Linguagem P4

Gabriela Surita
RA 139095

Resumo

Esta atividade introduz conceitos básicos do plano de dados de dispositivos de rede através da especificação de planos de dados usando a linguagem P4. A linguagem P4 é uma linguagem específica de domínio para a especificação de planos de dados de dispositivos de rede, que permite configurações via software de camadas de rede após a implementação.

Dentre as tarefas, foram listados alguns exemplos de código da linguagem P4, a compilação e execução de testes de switches implementados na linguagem usando a ferramenta Mininet, e por fim foram estudados aspectos de tempo de execução de dispositivos de rede.

Introdução

Soluções de engenharia de computação implementadas em software, tem, em geral um ciclo de vida muito mais rápido e um *time to market*, ou tempo até o lançamento, muito menor que soluções equivalentes em hardware. Como consequência, soluções em software geralmente hardware multipropósito, e por consequência sacrificam desempenho.

Linguagens de descrição de hardware, como VHDL e Verilog, atreladas a tecnologias como FPGAs, tem como proposta permitir soluções de compromisso entre implementar soluções puramente a nível de software usando hardware programável, permitindo rápida prototipagem e manutenção. No entanto, estas soluções não fazem considerações sobre o domínio das quais o hardware é proposto. Como consequência, em geral o hardware é caro e também tem perdas de desempenho em relação à soluções ASIC.

No caso de redes de computadores, em especial sobre o problema de roteamento de pacotes, em geral o domínio é bastante restrito e podemos fazer considerações

mais fortes sobre o hardware e a linguagem de descrição de hardware utilizada. Assim surge a ideia de uma linguagem de descrição de Hardware específica para o domínio de redes.

A linguagem P4 permite expressar como pacotes são processados pelo plano de dados de um elemento de rede capaz de receber ou redirecionar pacotes, como switches, placas de redes ou roteadores. O nome P4 é uma alusão a “Programming Protocol-independent Packet Processors”, nome dado ao artigo que propõe a primeira versão do protocolo.

Objetivos

Nesta atividade, propõe-se estudar conceitos sobre básicos sobre plano de dados de dispositivos de rede através da descrição de dispositivos de redirecionamento de pacotes (ou switches) usando a linguagem P4 e a ferramenta de simulação de redes mininet.

Propõe-se:

1. Descrever um plano de dados básico.
2. Estudar a implementação de algumas operações comuns executadas por switches comerciais.
3. Estudar aspectos de tempo de execução de switches.

Execução

Exemplos básicos

Abaixo se encontram alguns exemplos de blocos de código em linguagem P4 usados em um roteador simples:

1. Parse dos cabeçalhos Ethernet e IPv4 de pacotes

```
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;
    }
```

2. Procura de destino IPv4 na tabela de roteamento usando longest prefix match

```
    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        // ...
    }

    apply {
        if (hdr.ipv4.isValid()) {
            ipv4_lpm.apply();
        }
    }
```

3. Atualização de endereços MAC fonte e destino

```
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
```

4. Decremento do campo time-to-live (TTL) do pacote

```
    hdr.ipv4.ttl = hdr.ipv4.ttl - 3;
```

5. Definição da porta de saída (egress) do pacote

```
    standard_metadata.egress_spec = port;
```

6. Deparse/escrita dos cabeçalhos internos do pacote de saída

```
    apply {
        packet.emit(hdr.ethernet);  // Cabeçalho Ethernet
        packet.emit(hdr.ipv4);      // Cabeçalho Ipv4
        }
    }
```

Prototipando usando Mininet

Executando a topologia de exemplo compilando o switch especificado, obtemos uma interface interativa do Mininet. Podemos executar os comandos send.py e receive.py em nós diferentes da rede para observar o comportamento dos pacotes roteados pela rede. O resultado dos pacote enviado por h1 e o recebido por h2 encontra-se abaixo.

```
mininet> h2 ./receive.py &
mininet> h1 ./send.py 10.0.2.2 "Hello P4"
WARNING: No route found for IPv6 destination :: (no default route?)
sending on interface h1-eth0 to 10.0.2.2
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 00:00:00:00:01:01
  type      = 0x800
###[ IP ]###
      version   = 4L
      ihl   = 5L
      tos   = 0x0
      len   = 48
      id    = 1
      flags =
      frag  = 0L
      ttl   = 64
      proto = tcp
      chksum= 0x63c5
      src   = 10.0.1.1
      dst   = 10.0.2.2
      \options   \
###[ TCP ]###
      sport = 51609
      dport = 1234
      seq   = 0
      ack   = 0
      dataofs   = 5L
      reserved  = 0L
      flags = S
      window= 8192
      chksum= 0x3646
      urgptr= 0
      options   = []
###[ Raw ]###
            load  = 'Hello P4'
mininet>
sniffing on h2-eth0
got a packet
###[ Ethernet ]###
  dst       = 00:00:00:00:02:02
  src       = 00:00:00:02:02:00
  type      = 0x800
###[ IP ]###
      version   = 4L
      ihl   = 5L
      tos   = 0x0
      len   = 48
      id    = 1
      flags =
      frag  = 0L
```

```
      ttl   = 58
      proto = tcp
      chksum= 0x69c5
      src   = 10.0.1.1
      dst   = 10.0.2.2
      \options   \
###[ TCP ]###
      sport = 51609
      dport = 1234
      seq   = 0
      ack   = 0
      dataofs   = 5L
      reserved  = 0L
      flags = S
      window= 8192
      chksum= 0x3646
      urgptr= 0
      options   = []
###[ Raw ]###
            load  = 'Hello P4'
```

Podemos observar algumas diferenças entre os pacotes, por exemplo:

1. Os endereços de origem e destino (src e dst) Ethernet são diferentes. Isto ocorre porque a topologia de rede não prevê os dois hosts dentro de uma mesma subrede. No caso, o endereço de destino de h1 é s1, e o de origem de h2 é s2.

2. O TTL dos pacotes foi decrementado em 6.

Considerando o TTL recebido, podemos considerar que do ponto de vista de comportamento, o valor recebido está correto, pois a implementação previa um decremento de 3 em cada hop (temos dois hops s1 e s2). No entanto, essa implementação parece fora do padrão, pois é esperado em roteadores comerciais que cada hop decremente o TTL em apenas 1.

Configurações de tempo de execução

Uma próxima proposta é verificar a conectividade entre h1 e h2. Através de um comando ping, podemos observar que ela não é bem sucedida.

```
mininet> h1 ping h2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
^C
--- 10.0.2.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4050ms
```

Uma análise exploratória no código P4 não indica que existe um problema na declaração da camada de dados do dispositivo. O que nos leva a desconfiar do conteúdo das entradas das tabelas do switch.

No código, não existe nenhum mecanismo para edição de tais entradas, mas um protocolo de autodescoberta de dispositivos poderia, em tese, ser implementado usando a linguagem. No nosso exemplo, no entanto, as entradas são carregadas na inicialização através de arquivos JSON previamente declarados.

Podemos verificar, por exemplo, o conteúdo das tabelas do switch s1.

```
vagrant@p4:~/lab3/exercises/basic$ cat s1-runtime.json
{
  "target": "bmv2",
  "p4info": "build/basic.p4info",
  "bmv2_json": "build/basic.json",
  "table_entries": [
      {
      "table": "MyIngress.ipv4_lpm",
      "default_action": true,
      "action_name": "MyIngress.drop",
      "action_params": { }
      },
      {
      "table": "MyIngress.ipv4_lpm",
      "match": {
      "hdr.ipv4.dstAddr": ["10.0.1.1", 32]
      },
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
      "dstAddr": "00:00:00:00:01:10",
      "port": 1
      }
      },
      {
      "table": "MyIngress.ipv4_lpm",
      "match": {
      "hdr.ipv4.dstAddr": ["10.0.2.2", 32]
      },
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
      "dstAddr": "00:00:00:02:02:00",
      "port": 2
      }
      },
      {
      "table": "MyIngress.ipv4_lpm",
      "match": {
      "hdr.ipv4.dstAddr": ["10.0.3.3", 32]
      },
```

```
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
      "dstAddr": "00:00:00:03:03:00",
      "port": 3
      }
      }
  ]
}
```

Podemos verificar as entradas declaradas contra os endereços de IP e endereços MAC das máquinas. Vemos que claramente, neste caso, a declaração do endereço MAC de h1 estava errada.

```
mininet> h1 ip address | grep inet
      inet 127.0.0.1/8 scope host lo
      inet 10.0.1.1/24 brd 10.0.1.255 scope global h1-eth0
mininet> h2 ip address | grep inet
      inet 127.0.0.1/8 scope host lo
      inet 10.0.2.2/24 brd 10.0.2.255 scope global h2-eth0
mininet> h3 ip address | grep inet
      inet 127.0.0.1/8 scope host lo
      inet 10.0.3.3/24 brd 10.0.3.255 scope global h3-eth0
mininet> h1 ip address | grep ether
      link/ether 00:00:00:00:01:01 brd ff:ff:ff:ff:ff:ff link-netnsid 0
mininet> h2 ip address | grep ether
      link/ether 00:00:00:00:02:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
mininet> h3 ip address | grep ether
      link/ether 00:00:00:00:03:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

O erro pode ser corrigido alterando o bloco:

```
      {
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
      "dstAddr": "00:00:00:00:10:01",
      "port": 1
      }
```

Para:

```
      {
      "action_name": "MyIngress.ipv4_forward",
      "action_params": {
      "dstAddr": "00:00:00:00:01:01",
      "port": 1
      }
```

Reinicializando a rede, vemos que agora a conectividade está estabelecida.

```
mininet> h1 ping h2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=58 time=1.59 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=58 time=1.35 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=58 time=1.39 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=58 time=1.84 ms
^C
--- 10.0.2.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 1.357/1.548/1.841/0.193 ms

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Referências

[1] P4 Language Specification. Version 1.0.0 Available at https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html.