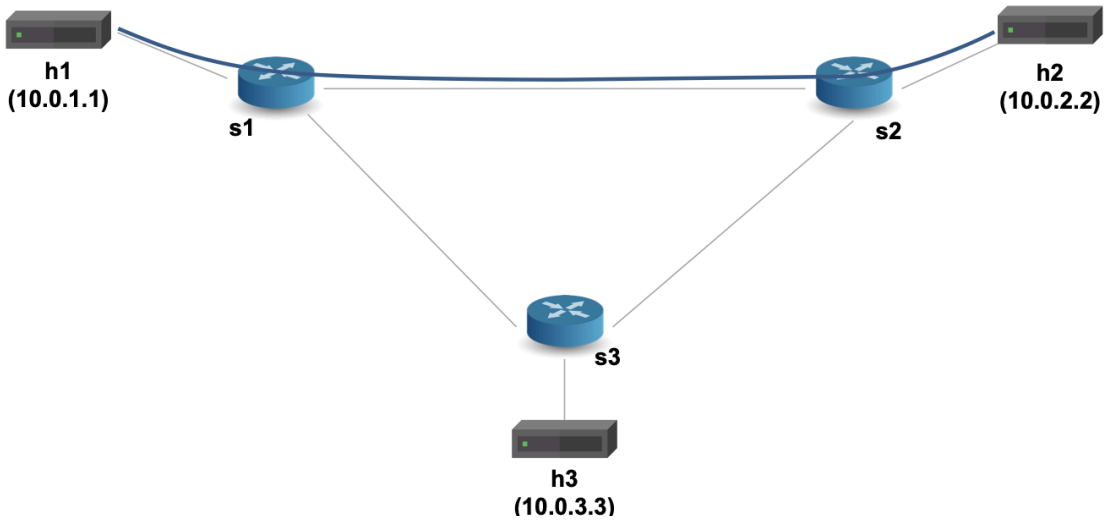
SEMESTRE 2020/1 (ERE)
AULA PRÁRICA: PROGRAMANDO EM P4

Objetivo da Aula Prática: Criando e Interagindo com Protocolos em P4

Você deverá implementar encaminhamento básico IPv4 em um arquivo P4 que está incompleto. Apenas como referência, esta será a topologia na qual você irá trabalhar, porém sua solução funcionará de maneira agnóstica à topologia.



Tutorial de apoio: https://p4.org/assets/P4_tutorial_01_basics.gslide.pdf

Passos:

1. Faça o download da VM da disciplina que está no Moodle;
2. Faça o download da pasta base para esta AP (“AP-Protocolos” no Moodle);
3. Entre na pasta “AP-Protocolos/exercises/AulaPratica”;
4. Identifique os “blocos” do programa **`basic.p4`** e as suas respectivas funções conforme vistas em aula anterior;
5. Nosso objetivo é dar vida a este arquivo **`basic.p4`** (que está incompleto). Será necessário programar os blocos básicos de um arquivo P4 para conseguir fazer *forwarding* de pacotes:
    6. Explicitamente, para cada pacote IPv4, iremos: (i) atualizar o endereço MAC destino e origem, (ii) decrementar o campo TimeToLive (TTL), e (iii) fazer um *lookup* em uma tabela de encaminhamento, enviando o pacote pela porta de saída correta;
7. Vamos começar implementando o *parser* para identificar o IPv4:
    8. O *parser* serve para ler cada pacote que entra no switch e identificar quais bytes pertencem a quais cabeçalhos;
9. Começamos adicionando um estado inicial. Como teremos que lidar apenas com as camadas acima da física, nosso estado inicial levará diretamente ao protocolo ethernet (todos outros protocolos serão descartados).
10. Adicione o seguinte trecho ao *parser*:

```
state start {
    transition parse_ethernet;
}
```

11. Para a camada de ethernet, nós queremos aceitar qualquer pacote de camada dois que seja ethernet, porém quando o cabeçalho IPv4 estiver presente, nós queremos carregá-lo em nosso programa;

12. Adicione o seguinte:

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

13. Finalmente, extraímos o cabeçalho IPv4 e finalizamos o *parser*, com qualquer transição indefinida resultando no descarte do pacote:

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

14. Seu *parser* deve estar desta maneira:

```
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;
    }

}
```

15. Com o *parser* devidamente implementado, iremos para o segundo passo para implementar o encaminhamento, que é aplicar os conceitos do passo 6;

16. No bloco chamado **`MyIngress`**, começaremos declarando uma tabela de lookup:

```
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = drop();
}
```

17. Entrando em detalhes: queremos declarar uma **`table`** com um nome arbitrário (que em nosso caso será **`ipv4_lpm`**. A chave de entrada será o endereço destino (e será feito o matching "**`lpm`**" ou *longest prefix match*). A lista de ações possíveis para esta tabela é, para cada pacote: encaminhá-lo, ou descartá-lo, ou não fazer nada e seguir o programa. Opcionalmente temos um parâmetro **`size`** de número máximo de entradas, e também uma ação padrão caso não encontremos nenhum *match* para o endereço destino. Nota: estas ações serão definidas no próximo passo;

18. Agora, definiremos as ações que foram explicitadas na tabela anterior: primeiramente queremos ter declarado dentro do bloco **`ingress`** uma ação que apenas descarte os pacotes, usando a primitiva **`mark_to_drop()`**. Na sequência, vamos definir a ação que trata os pacotes IPv4: esta função recebe dois argumentos vindos da tabela, o MAC destino e a porta destino. Dentro da ação, nós iremos especificar a porta de saída, preencher o MAC destino e MAC origem, e finalmente, decrementar o TTL:

```
action drop() {
    mark_to_drop();
}

action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    standard_metadata.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

19. Estas todas foram as declarações do bloco **`Ingress`**. Porém, temos que incluir o bloco **`apply`**, que é o ponto de entrada de um bloco **`Ingress`** (e funciona como uma "main"):

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

20. Dentro do bloco **`apply`**, queremos, para cada pacote que seja válido, aplicar a nossa tabela **`ipv4_lpm`**. Caso o fluxo de controle incluísse outra logica, ela seria escrita dentro deste bloco **`apply`**;

21. Seu **`MyIngress`** deve estar assim:

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

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }

    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }

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

22. Finalmente, temos que configurar o *deparser*, que descarrega as estruturas do nosso programa de volta para o pacote. Necessariamente iremos fazer o **`emit`** de cada cabeçalho que parseamos e que queiramos que esteja no pacote final, na ordem que desejarmos:

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

23. Agora, seu arquivo **`basic.p4`** deve estar pronto para executar. Execute o comando "**`make`**" dentro da pasta AulaPratica e observe enquanto o mininet inicializa a topologia com seu programa;

24. Os comandos e entradas de tabela padrões foram escritos por nós de antemão. Caso queira adicionar comandos e entradas de tabela novos: para cada switch existe um arquivo **`commands-s*.txt`**, inclua um comando por linha e execute o arquivo

config.sh que fizemos por conveniência (para que vocês não precisem interagir com arquivos JSON em mais baixo nível);

25. Ao abrir o mininet, use o comando "**xterm h1 h2**";

26. Em um dos terminais, digite **ifconfig** e descubra o IP deste host, e logo após execute o arquivo **receive.py**;

27. Use o outro terminal para executar um "**send.py <ip> <mensagem>**" sendo **<ip>** o IP do primeiro terminal, se o pacote e mensagem aparecerem no receptor corretamente, há conectividade e seu programa funcionou!

28. Agora, let's get creative! Vamos usar a linguagem P4 para implementar funcionalidades além do encaminhamento, criando uma *blacklist hardcoded*. Saia do mininet executando "**exit**", e mude seu bloco **apply** para descartar qualquer pacote que venha de um dos IP do host h (use a representação hexadecimal do IP).

29. Adicione a condicional a seguir ao bloco **apply** do seu programa

```
apply {
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
        if (hdr.ipv4.srcAddr == 0x0a000101) {
            mark_to_drop();}
    }
```

30. Execute novamente o "**make**", e use o comando "**pingall**" no mininet para testar rapidamente a conectividade da rede inteira. O comando irá tentar, para cada host, se comunicar com todos os outros hosts, retornando uma matriz de conectividade (onde "X" é uma falha).

31. Perceba como sua modificação alterou a matriz de conectividade da rede.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X h3
h3 -> X h2
*** Results: 66% dropped (2/6 received)
```

32. **Problemas?** Revise os passos anteriores e corrija ou chame o monitor!

Se você chegou até aqui, parabéns! Mostre seu programa executando para o professor, e seu objetivo na aula de hoje terá sido cumprido.

Mas espere!!! Ainda há uma série de exemplos de aplicações de rede em P4 para serem explorados. Caso você ainda tenha tempo em aula, ou caso prefira fazer em casa, algumas atividades adicionais são sugeridas como exercício:

- A pasta "exercises" contém diversos exercícios feitos pelos criadores da linguagem P4 para demonstrar a utilidade da linguagem. Alguns exemplos:
  - Balanceamento de carga em P4
  - Fazendo uma calculadora em P4 (*in-network compute*)
  - Tunelamento em P4

- Experimente entrar nestas pastas e completar os passos do arquivo README.txt
- A solução de cada exercício está sempre na pasta “solution” do exercício (inclusive a desta aula prática!), mas caso tenha qualquer dúvida sobre a implementação, pergunte!