Lab 6: Introdução a programação de dispositivos com a linguagem P4

Objetivos

Este laboratório possui como objetivo geral introduzir o conceito de planos de dados programáveis por meio da linguagem P4 (Programming Protocol-independent Packet Processors - https://p4.org (https://p4.org)). Os objetivos específicos deste laboratório incluem:

• Permitir ao aluno melhorar a sua compreensão do funcionamento de equipamentos de rede, principalmente no que diz respeito ao plano de dado em relação a funções básicas de análise de cabeçalhos (parsing), endereçamento e comutação de pacotes.

• Abordar os princípios de projeto de sistema embarcado com foco no plano de dados (datapath) de dispositivos de redes e sua programação através da linguagem P4 versão 16.

• Desenvolver atividades práticas com a linguagem P4.

O material desta atividade foi adaptado a partir do tutorial P4 disponível no endereço https://github.com/p4lang/tutorials (https://github.com/p4lang/tutorials)

Contexto

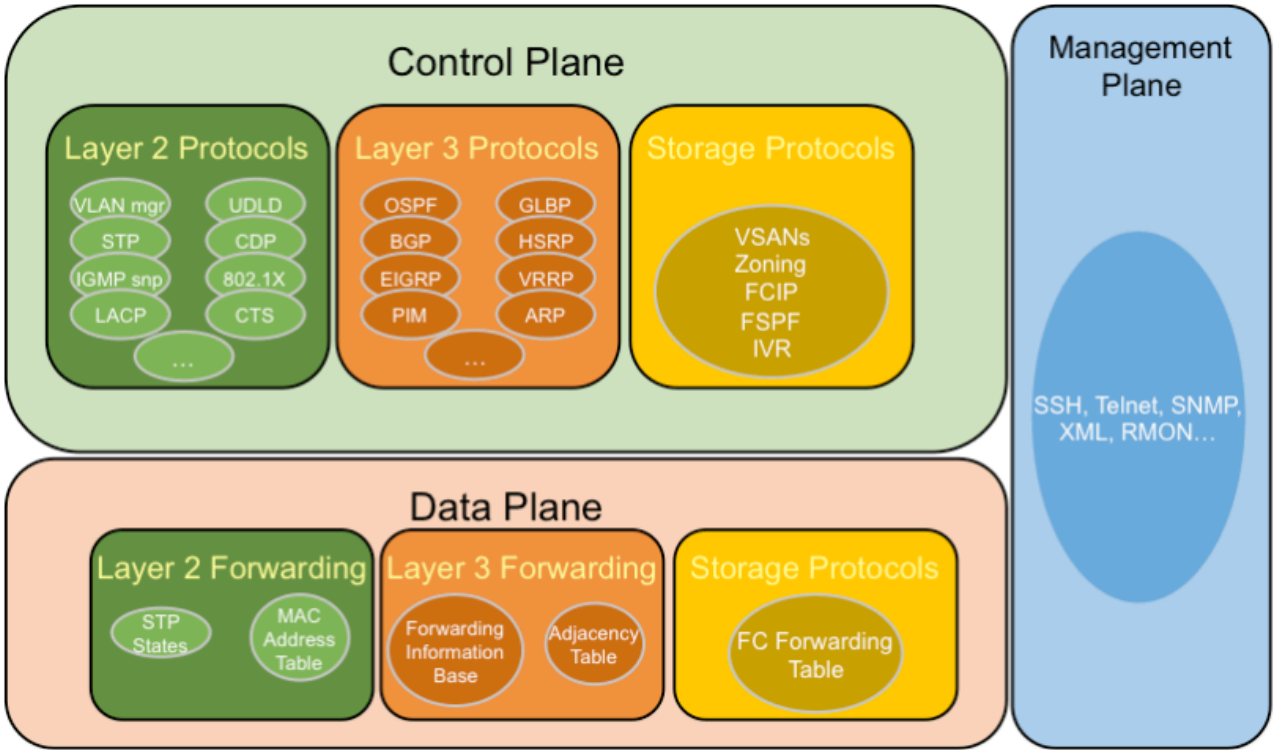
Arquitetura tradicional de equipamento de rede

Analisando uma arquitetura clássica de roteador (veja Fig. 1) nos permite observar que esta se trata de um modelo formado basicamente por dois planos distintos: o software de controle e o hardware dedicado ao processamento de pacotes. O plano de controle é encarregado de implementar protocolos de roteamento e tomar decisões tais como escolha da melhor rota, transferindo essas decisões (ex: tabela de encaminhamento) para o plano de dados através de uma API proprietária.

A única interação externa que um roteador pode receber ocorre através de interfaces de configuração, como por exemplo Web, SNMP, CLI. Isto limita o uso do dispositivo às funcionalidades programadas pelo fabricante dos circuitos integrados (ASIC - Application Specific Integrated Circuits) como funções fixas

implementadas em hardware por questão de desempenho e custo. Por ser uma implementação em hardware, o usuário do dispositivo não consegue instalar ou alterar as funcionalidades do plano de dados que faz o processamento de pacotes em altas taxas de transmissão.

O Plano de dados, majoritariamente proprietário, dedicado e fixo (veja Fig. 2), torna difícil a definição de novas características para o processamento de pacotes, o que prejudica a inovação. Por exemplo, para que se modifique o plano de dados, um novo design de todo o conjunto de hardware deve ser realizado, para que então um novo ASIC seja produzido, distribuído e instalado: um processo de alto custo e longa duração. Daí a necessidade da proposta de um plano de dados flexível, cujo funcionamento possa ser alterado conforme a necessidade, isto é, um plano de dados programável.



Plano de dados programável

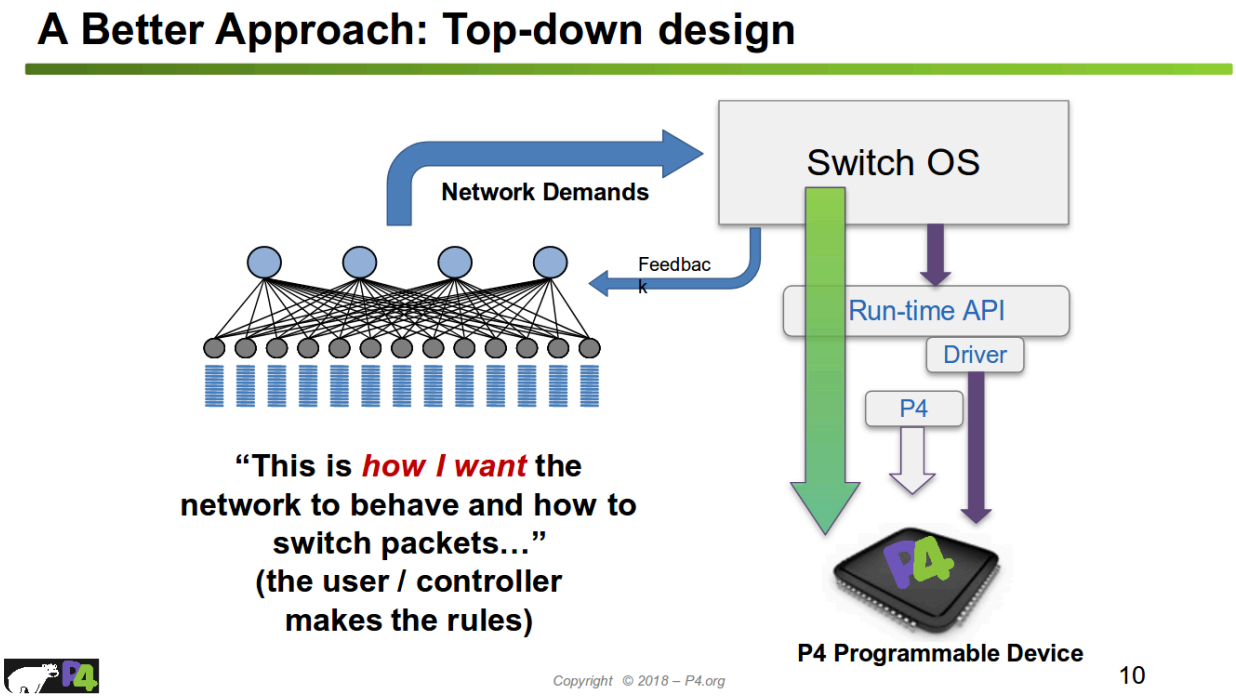
Buscando a flexibilidade para definição de comportamentos específicos no tratamento do encaminhamento de pacotes em redes programáveis (veja Fig. 2), a linguagem P4 abstrai a programação do plano de dados de redes, almejando três objetivos:

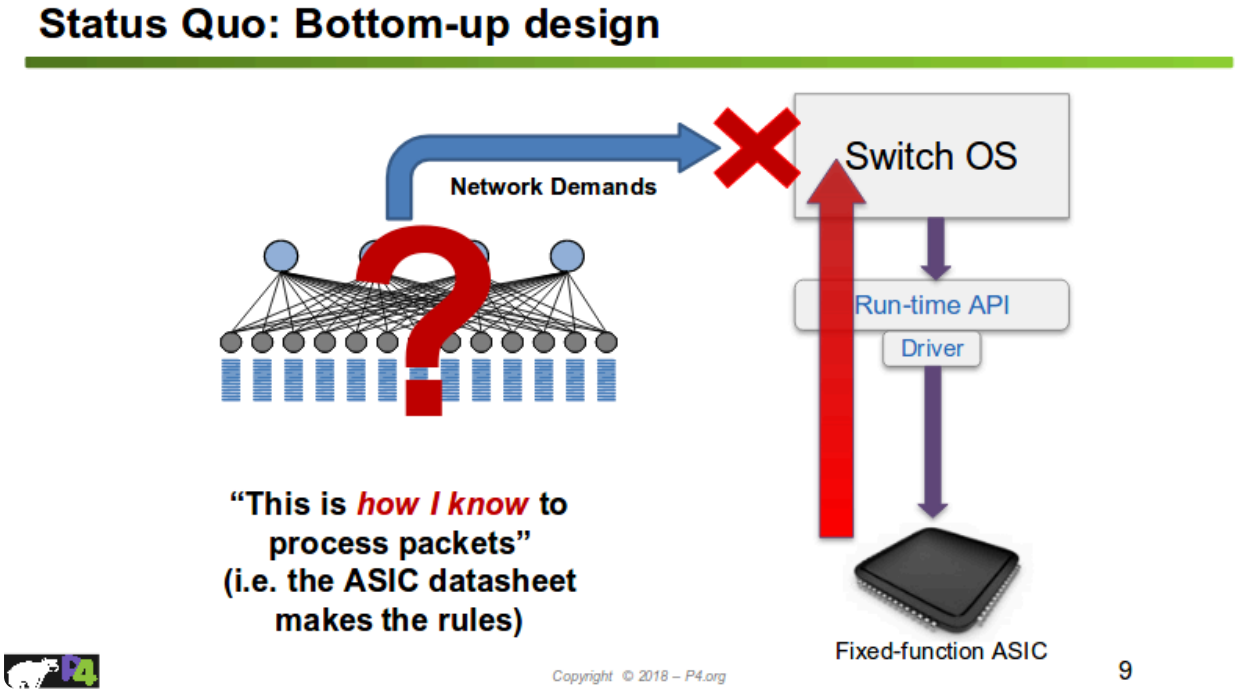
• Independência do protocolo: O dispositivo deve ser capaz de analisar qualquer tipo de protocolo desde que este seja declarado anteriormente no programa . Como por exemplo, análise de cabeçalho e ações usando o modelo

de Match+Action típico do protocol Openflow;

• Independência de arquitetura: O programa P4 deve abstrair a camada física, isto é, deve ser o mesmo independente do equipamento alvo (target) no qual está sendo aplicado, mesmo que diferentes tecnologias SW/HW sejam utilizadas. Dessa forma, as peculiaridades de cada equipamento devem ser tratadas pelo próprio compilador;

• Reconfiguração em campo: O processamento dos pacotes pode ser alterado uma vez implantado, até mesmo em tempo de execução.

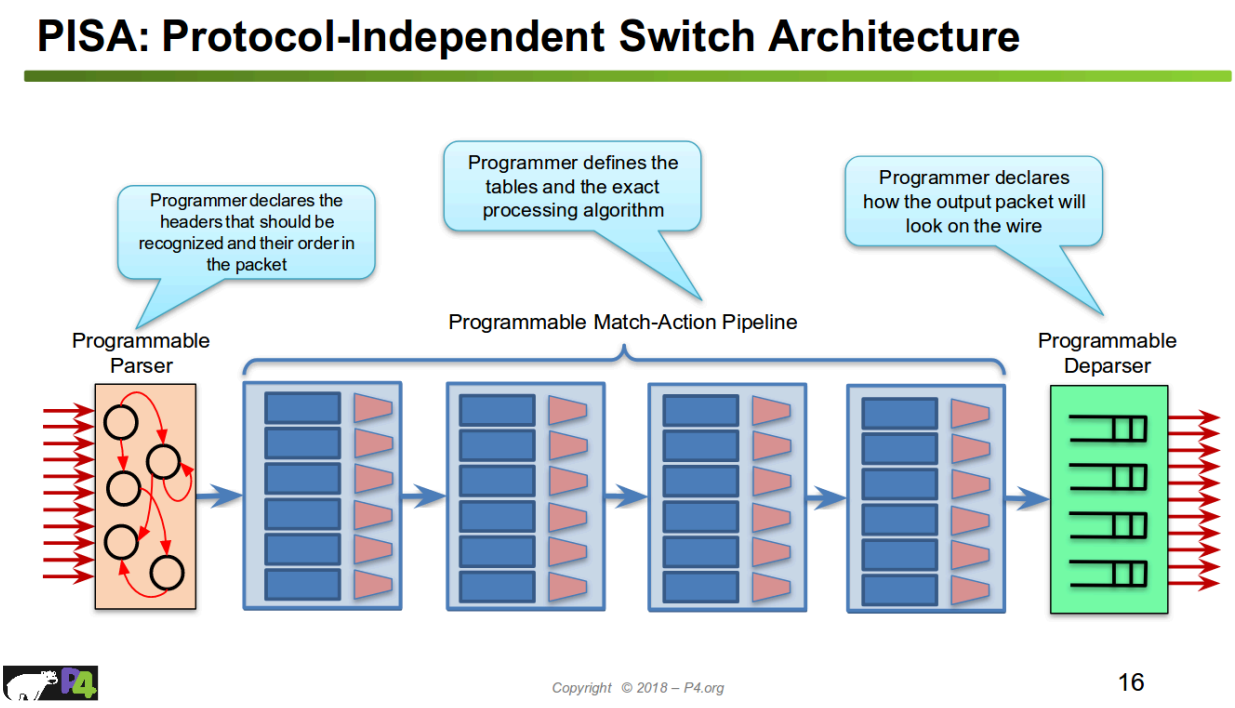


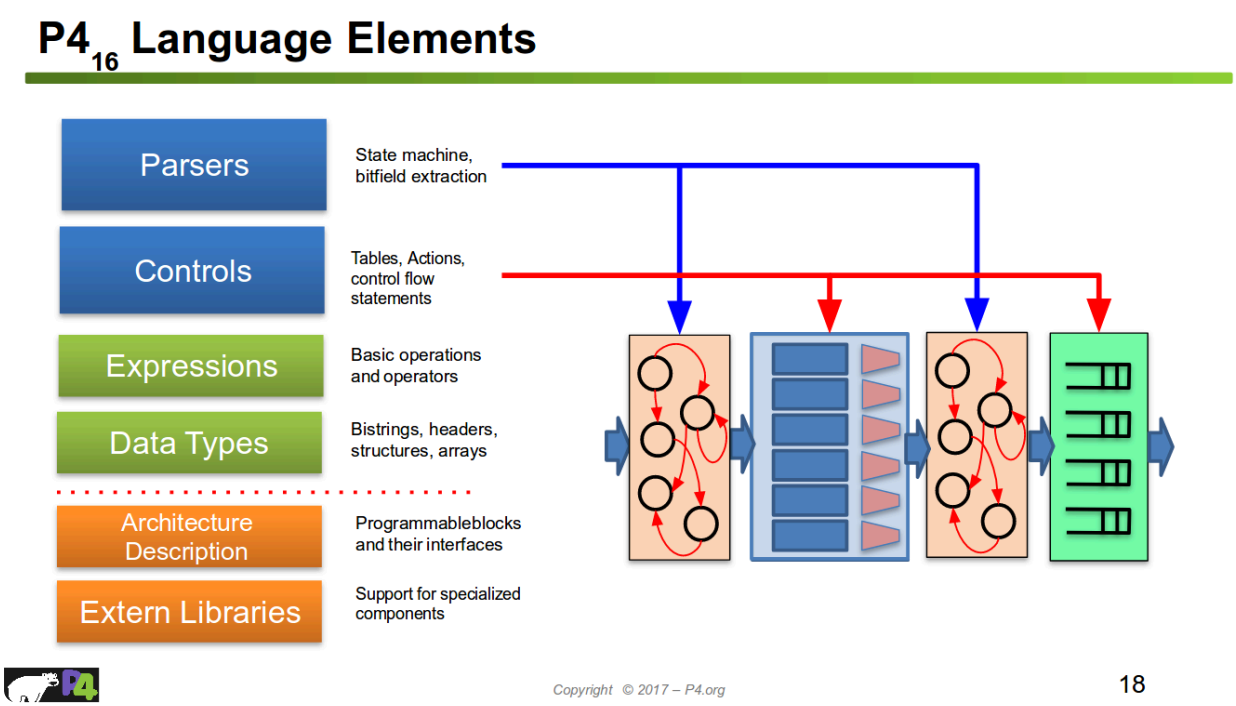


A linguagem P4 se baseia na arquitetura PISA (Protocol-Independent Switch Architecture) caraterizado por hardware baseado em um pipeline que permite (re)configuração usando o modelo de Match+Action (veja Fig. 3). Dessa forma, diferentemente dos ASICs tradicionais com funções fixas, a arquitetura provê grande flexibilidade sem comprometer a performance (chips com capacidade na mesma ordem de Tb/s suportados em ASICs).

Na linguagem P4 os principais blocos programáveis são parsers e controls (veja Fig. 4). Parsers são implementados como máquinas de estado com o propósito de extrair campos de cabeçalhos de pacotes de forma definida pelo usuário. Controls são responsáveis pelo processamento principal de pacotes em estruturas match-action. Aqui as regras de tabelas e ações são definidas para manipular cabeçalhos de pacotes e meta-dados (metadata). O programador/usuário é responsável por realizar a codificação de parsers e controls por meio de diferentes tipos de estruturas de dados e expressões providas pela linguagem P4 (1) (https://opennetworking.org/wp-content/uploads/2020/12/p4-cheat-sheet.pdf).

Em contrapartida, as estruturas de definição de arquitetura e bibliotecas externas podem ser definidas pelo fabricante de equipamento onde o programa P4 irá ser executado. Enquanto externs já são estruturas especializadas que definem rotinas como caixas-pretas que o programa P4 pode utilizar em sua execução.





A estrutura de um programa escrito na linguagem P4 (veja Fig. 5) possui diferentes elementos, dentre eles:

• Headers: representam estruturas de dados que referenciam campos nos cabeçalhos de pacotes

• Parser: responsáveis pela análise e extração dos campos de pacotes;

• Deparser: responsáveis pela modificação final de cabeçalhos de pacotes;

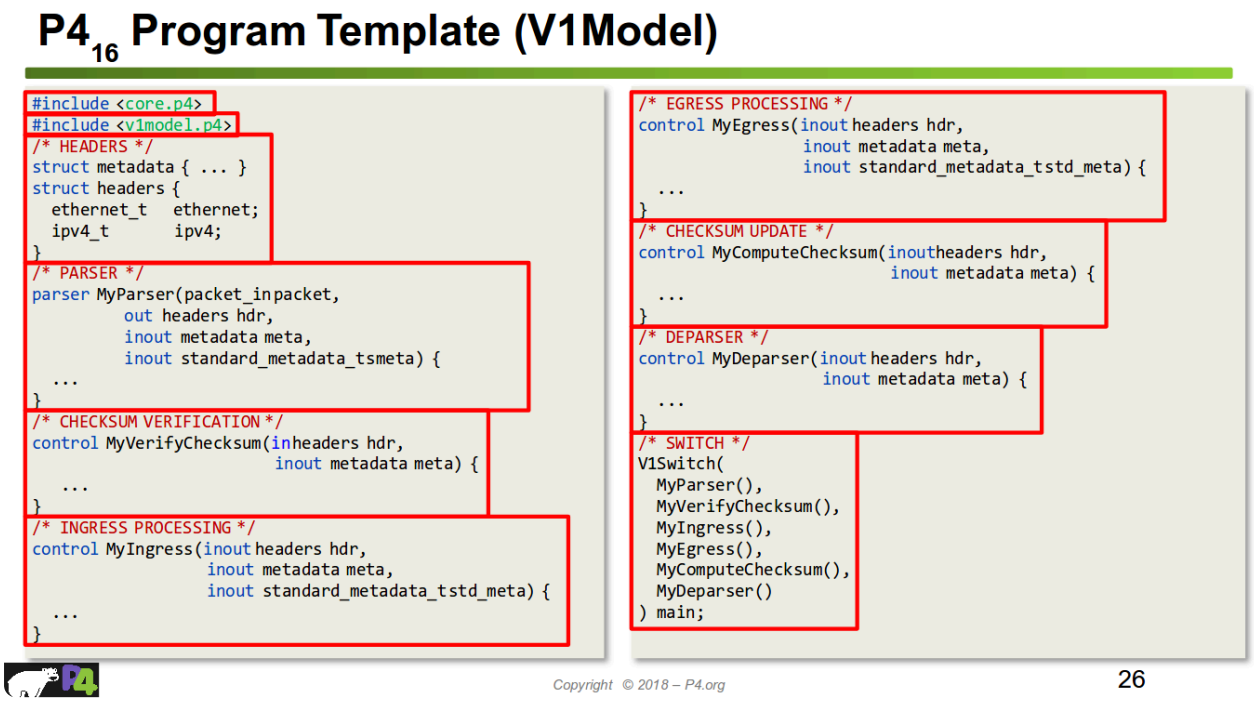
• Ingress: depois de passarem pelo parser pacotes são encaminhados a ingress onde podem ter ações aplicadas sobre eles dependendo de estruturas de tabelas, definindo assim filas e portas de saída para os pacotes;

• Egress: antes de deixar o pipeline, pacotes podem ter ações aplicadas sobre eles em egress, com a realização de modificações nos cabeçalhos de pacotes;

• Tables: mecanismo que faz o processamento de pacotes por meio de correspondên-
cias (key/matches) e suas respectivas ações associadas a serem executadas sobre os
pacotes;

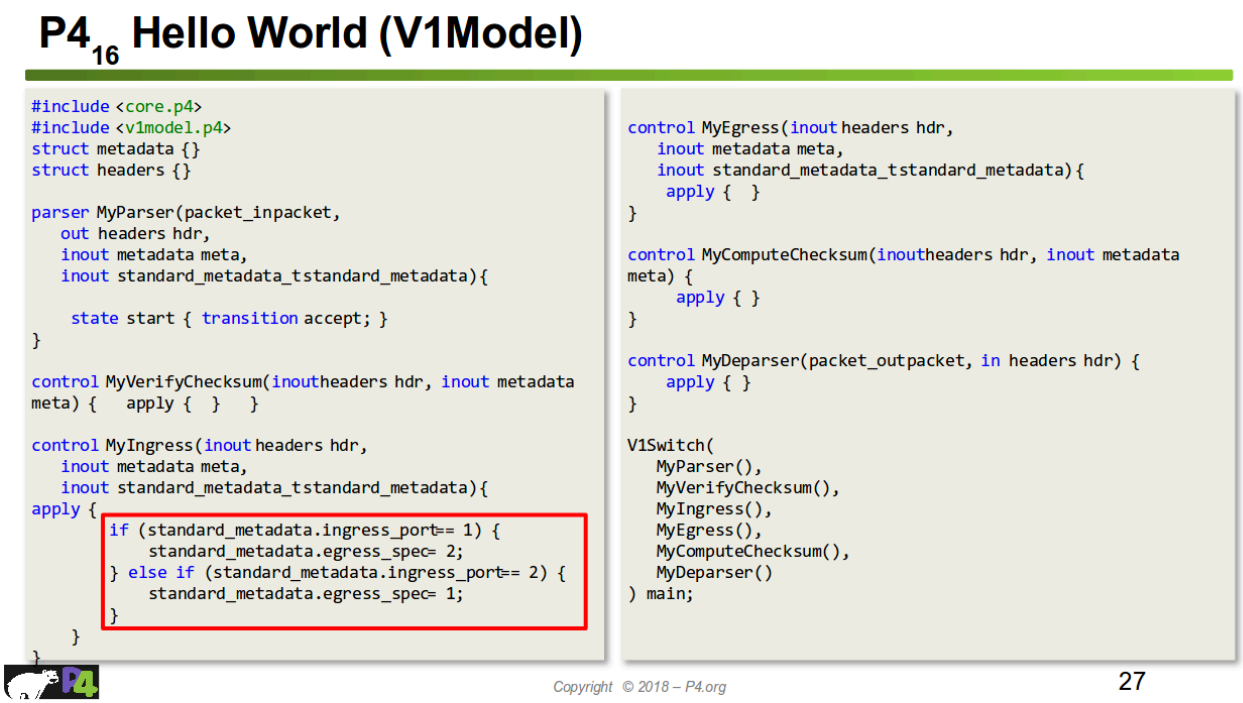
• Actions: conjunto de primitivas a serem aplicadas (possivelmente por funções customizadas) sobre pacotes;

- Control: estabelece o fluxo de processamento de pacotes do programa P4 utilizando os blocos previamente declarados.



Em um exemplo de programa P4 (veja Fig. 6), temos: (1) a estrutura de controle do switch “V1Switch”2 determinando todos os blocos de control sequencialmente pelos quais os pacotes serão tratados; (2) MyParser realizando o accepts de todos os pacotes recebidos pelo switch; (3) MyIngress realizando a simples alteração de portas de encaminhamento de pacotes (onde pacotes recebidos na porta 1 são encaminhados para porta 2 e vice-versa) (4) e os outros blocos de control não realizando nenhuma operação sobre os pacotes. Note que em cada um dos blocos de controle a função apply() realiza a chamada de ações a serem realizadas em pacotes.

> Obs.: O termo switch, assim como em Openflow, é usado na comunidade P4 para fazer referência ao equipamento de rede de maneira genérica e não deve ser associado ao termo tradicional de um switch de camada 2 com as funções clássicas de computação e aprendizado de endereços MAC / Ethernet.



Materiais necessários

Para realizar esta atividade o aluno necessitará dos seguintes pacotes

No Linux, precisamos de:

- VirtualBox
- Virtual Machine (VM) do P4

No Windows, precisamos de:

- VirtualBox
- Virtual Machine (VM) do P4

A VM do P4 pode ser baixada no link https://drive.google.com/uc?id=1lYF4NgFkYoRqtskdGTMxy3sXUV0jkMxo&export=download

(https://drive.google.com/uc?id=1lYF4NgFkYoRqtskdGTMxy3sXUV0jkMxo&export=download)

> Atenção: Os equipamentos devem ter 25GB ou mais livres de espaço em disco para importar a VM.

Configuração

A VM roda o SO Ubuntu 16.04 e dispõe de todas as funcionalidades do Mininet e da linguagem P4 já instaladas. Além disso, a VM possui todos os arquivos e pacotes necessários para utilização nas atividades do laboratório. Dessa forma,

o aluno não precisará instalar quaisquer ferramentas para realizar esta atividade.

Todas as dicas e configurações mostradas para VirtualBox podem ser realizadas pelo próprio aluno em um computador pessoal.

Primeiro acesso à Máquina Virtual:

> Para acessar a VM do P4 utilize as seguintes credenciais:
> Login: p4
> Password: p4

Após o acesso a maquina virtual, abra o terminal e faça o download do projeto Github contendo os exercicios necessários.:

```
git clone https://github.com/FranciscoVogt/P4-lab-2023.git
```

FAQ - Potenciais problemas

Sempre que você estiver com algum problema na execução do laboratório, retorne ao FAQ para verificar se a solução não está aqui.

> Execute sempre o comando make clean entre uma execução e outra

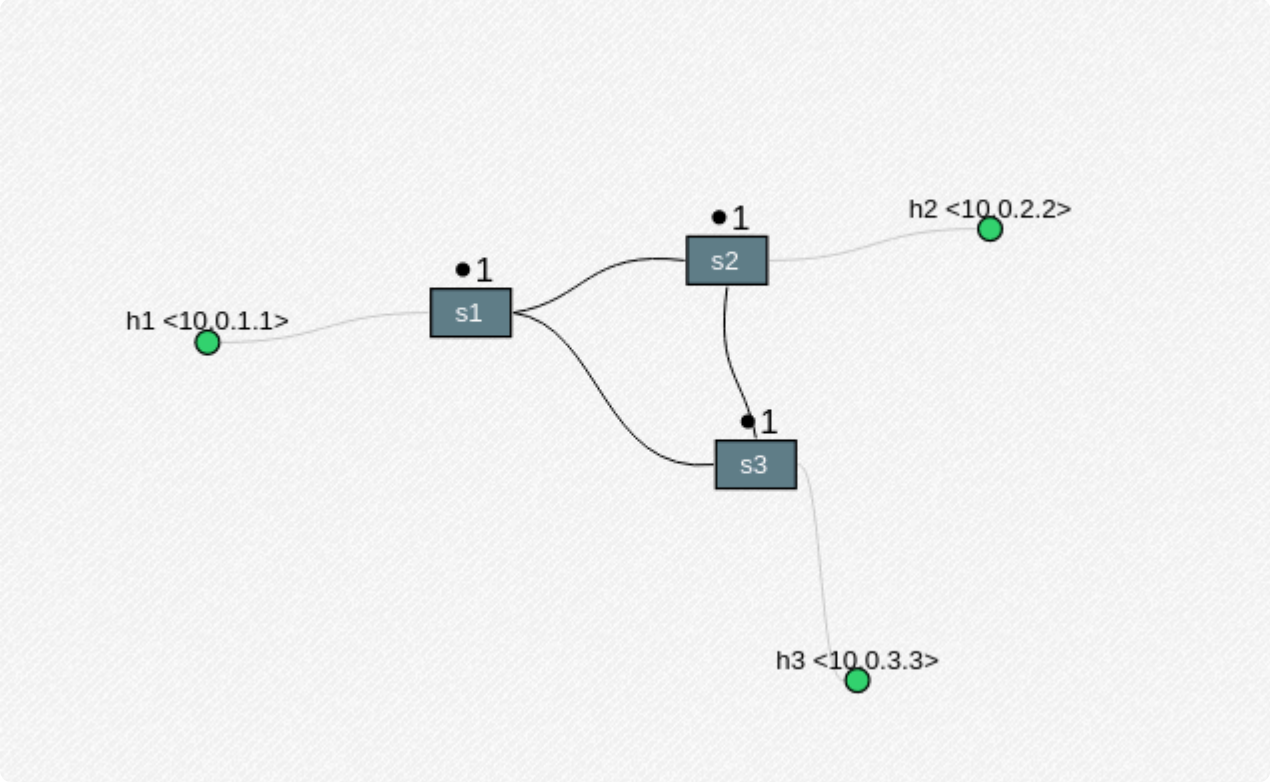
> No caso de um erro similar a:
> Exception: Error creating interface pair (s1-eth2,s2-eth2): RTNETLINK answers: File exists
>
> Execute o comando sudo mn -c ou make clean

> No caso de erro de compilação, e voce tiver convicção de que o seu código esta correto, exclua as pastas "build, logs e pcaps" de dentro da pasta do exercicio em questão. Também, é recomendado que você exclua elas SEMPRE antes de cada nova execução do comando "make run"

Atividades práticas

Nas atividades a seguir iremos trabalhar na análise, no design e na implementação de um programa P4 para o encaminhamento básico de pacotes entre hosts de uma topologia emulada no Mininet (veja A figura abaixo).

A figura representa a topologia que será utilizada em todos os exercicios.



Atividade 1: Análise do programa P4

Após iniciar a maquina virtual, e fazer o download do repositório github (veja a seção de configuração), acesse a pasta do exercicio 1 com o comando:

```
$ cd /home/p4/P4-lab-2023/exercises/ex1
```

Nesta pasta abra o arquivo

```
$ sudo subl basic.p4
```

> Analisando o programa aberto indique (referenciando a linha ou colando parte do código) onde ocorrem as seguintes operações:
> - Definição dos cabeçalhos (headers)
> - Parser dos cabeçalhos
> - Processamento de ingresso
> - Processamento de egresso
> - Deparser dos cabeçalhos
> - Calculo do checksum

Adicionalmente, comente brevemente o que acontece em cada um desses "blocos", no contexto desse código em especifico. Por exemplo, quais headers são definidos? o que o parser está fazendo? o que o bloco de ingress está fazendo? etc.

Atividade 2: Executando o código P4, e fazendo pequenas modificações.

Agora, vamos executar o código do exercicio 2. Para isso, acesse o diretório do exercicio 2 com o seguinte comando:

```
$ cd /home/p4/P4-lab-2023/exercises/ex2
```

execute o comando:

```
make
```

Este comando realiza a compilação do programa basic.p4, e utiliza o código compilado para definir o switch base a ser executado pela plataforma Mininet. Ou seja, os switches mostrados na Fig. 2 serão uma instância deste programa P4.

Em seguida o script realiza a configuração das tabelas de encaminhamento de cada um dos switches (s1, s2, s3) utilizando os arquivos “s[1,2,3]-runtime.json”.

Após a execução do comando “$ make”, no console mostrado da plataforma Mininet execute:

```
mininet> xterm h1 h2
```

para abrir os consoles externos dos hosts h1 e h2.
Na janela do xterm de h2 execute o comando:

```
./receive.py
```

E na janela do xterm de h1 execute o comando:

```
./send.py 10.0.2.2 "mensagem"
```

Onde dentro das aspas, você pode substituir o "mensagem", pela mensagem que voce desejar.

Estes programas realizam respectivamente a captura de pacote em h2 e o envio de 1 pacote de h1.

> Com base no comportamento dos switches, explique:
> • quais os valores campos Ethernet (src e dst) do pacote enviado em h1 e recebido em h2, eles são iguais ou diferentes? está correto?
> • quais os valores de TTL do pacote enviado e recebido? estão corretos? Por quê?

> Referencia sobre TTL que pode ajudar a responder a questão acima: https://pt.wikipedia.org/wiki/Time_to_Live (https://pt.wikipedia.org/wiki/Time_to_Live)

Agora vamos modificar o codigo P4 para que o TTL seja alterado a cada salto.

Para isso, devemos modificar a action "ipv4_forward", que realiza o encaminhamento dos pacotes quando temos um match na tabela.

Então, finalize a execução do código, e dite o arquivo basic.p4, adicionando as seguintes linhas na action "ipv4_forward":

```
hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
hdr.ethernet.dstAddr = dstAddr;
hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
```

> Note que foram adicionadas 3 linhas na action, 1 modificando o TTL, e outras duas modificando os endereços ethernet. Qual a função das linhas que modificam os endereços ethernet? por que voce acha que elas foram adicionadas?

Agora, após salvar o arquivo, vamos repetir o processo de envio do pacote. Então, de o comando exit no mininet, e execute novamente a topologia com o comando make, abra os terminais com o xterm, e execute os scripts para receber e enviar pacotes (mesmo processo feito anteriormente).

> Qualquer problema na execução, retorne a seção FAQ

> Novamente, com base no comportamento dos switches, explique:
> • quais os novos valores campos Ethernet (src e dst) do pacote enviado em h1 e recebido em h2, eles são iguais ou diferentes? qual a explicação para isso?
> • quais os novos valores de TTL do pacote enviado e recebido? qual a diferença entre eles? por que?

Atividade 3 - Adicionando suporte ao TCP e UDP

Na atividade anterior, podemos observar que ao enviarmos pacotes pela topologia, quando esses pacotes chegam ao destino eles contém o header TCP. No entanto, se verificarmos o código P4, o header TCP não está definido, assim como o UDP.

Isso ocorre pois no P4 precisamos apenas fazer o parser sequencial dos headers que queremos trabalhar (modificar ou usar para encaminhamento), e o restante será tratado como o payload do pacote (e se manterá igual).

Naquele caso, estavamos trabalhando apenas com os headers ethernet e ipv4, e como o TCP vem após eles, não precisamos fazer o parser dele.

Nesta atividade, incluiremos o suporte ao TCP e UDP, para que seja possível, por exemplo, modificar suas portas de origem e destino.

Então, o primeiro passo é acessar o diretorio do exercicio 3:

```
$ cd /home/p4/P4-lab-2023/exercises/ex3
```

Abra o código P4, e vamos começar a edita-lo.

Primeiramente, vamos definir o Header TCP, para que possamos processa-lo, então, na definição dos headers (e.g., apos o header IPV4), adicione o seguinte trecho de código:

```
header tcp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4>  dataOffset;
    bit<3>  res;
    bit<3>  ecn;
    bit<6>  ctrl;
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;
}
```

Além disso, adicione voce mesmo o header UDP.

> Dica, adicione o nome similar ao tcp "udp_t". O header udp tem apenas 4 campos, essa referencia pode ajudar: https://en.wikipedia.org/wiki/User_Datagram_Protocol (https://en.wikipedia.org/wiki/User_Datagram_Protocol)

Agora devemos adicionar os headers TCP e UDP a nossa lista de headers. Modifique a estrutura struct headers para:

```
struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
    tcp_t      tcp;
    udp_t      udp;
}
```

Após isso, devemos atualizar o parser, para que nosso parser consiga entender o header TCP ou UDP.

Altere o state parse_ipv4 para:

```
state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol) {
                6: parse_tcp;
         default: accept;
        }
    }
```

Dessa forma, caso o protocolo contido em IPv4 protocol for 6 (codigo em decimal para o protocolo tcp) o nosso codigo irá fazer o parser do header TCP.

> Adicione voce mesmo o codigo do UDP (dentro do bloco transition select), e um estado similar ao do TCP (exemplo: parse_udp).

Agora devemos criar os estados para o parser dos dois protocolos, tcp e udp. Para isso, abaixo do estado parse_ipv4, adicione os seguintes estados:

```
state parse_tcp {
        packet.extract(hdr.tcp);
        meta.tcpLen = hdr.ipv4.totalLen - 20;
        transition accept;
       }

state parse_udp {
        packet.extract(hdr.udp);
        transition accept;
       }
```

Pronto, agora nosso código é capaz de ler entender os headers UDP e TCP. Execute o mesmo processo do exercicio anterior (make, xterm h1 h2...) para executar a topologia e enviar um pacote de h1 para h2.

> O pacote foi recebido? Qual voce acha que é o motivo?

Apesar de termos feito o parser do header TCP e UDP, nao fizemos o seu deparser, que é a inclusão do header novamente no pacote. Ou seja, quando o pacote é recebido, apenas removemos o header.

Vamos alterar então o deparser do nosso código para incluir o header tcp e udp novamente. Adicione os seguintes comandos no deparser:

```
packet.emit(hdr.tcp);
packet.emit(hdr.udp);
```

Além disso, vamos atualizar o checksum para o TCP, entao no bloco MyCompute checksum, apos o update do checksum do IPv4, adicione as seguintes linhas:

```
update_checksum(
            hdr.tcp.isValid(),{  hdr.ipv4.srcAddr,
                                 hdr.ipv4.dstAddr,
                                 8w0,
                                 hdr.ipv4.protocol,
                                 meta.tcpLen,
                                 hdr.tcp. srcPort,
                                 hdr.tcp.dstPort,
                                 hdr.tcp.seqNo,
                                 hdr.tcp.ackNo,
                                 hdr.tcp.dataOffset,
                                 hdr.tcp.res,
                                 hdr.tcp.ecn,
                                 hdr.tcp.ctrl,
                                 hdr.tcp.window,
                                 hdr.tcp.urgentPtr},
                                 hdr.tcp.checksum,
            HashAlgorithm.csum16);
```

> Opcional: procure na internet e adicione a atualização do checksum para o UDP também.

Para verificarmos que está tudo correto, no bloco apply do ingress, vamos alterar o destination port do TCP e verificar o resultado. Entao, no bloco apply dentro do MyIngress, adicione o seguinte trecho de codigo:

```
if(hdr.tcp.isValid()){
    hdr.tcp.dstPort = 4321;
}
```

Pronto, agora execute novamente a topologia, e envie o pacote de h1 para h2 novamente.

> Qual foi o resultado? qual destination port na origem e no destino (h1 e h2)?

Congratulations

Agora você sabe um pouco sobre P4!!