Minicurso
Introdução à Linguagem P4
Teoria e Prática

Luis Fernando U. Garcia (UFES)
Rodolfo S. Villaça (UFES)
Moisés R. N. Ribeiro (UFES)
Regis F. T. Martins (UFSCar)
Fábio L. Verdi (USFCar)
Cesar Marcondes (UFSCar)

Sumário

Evolução das Redes

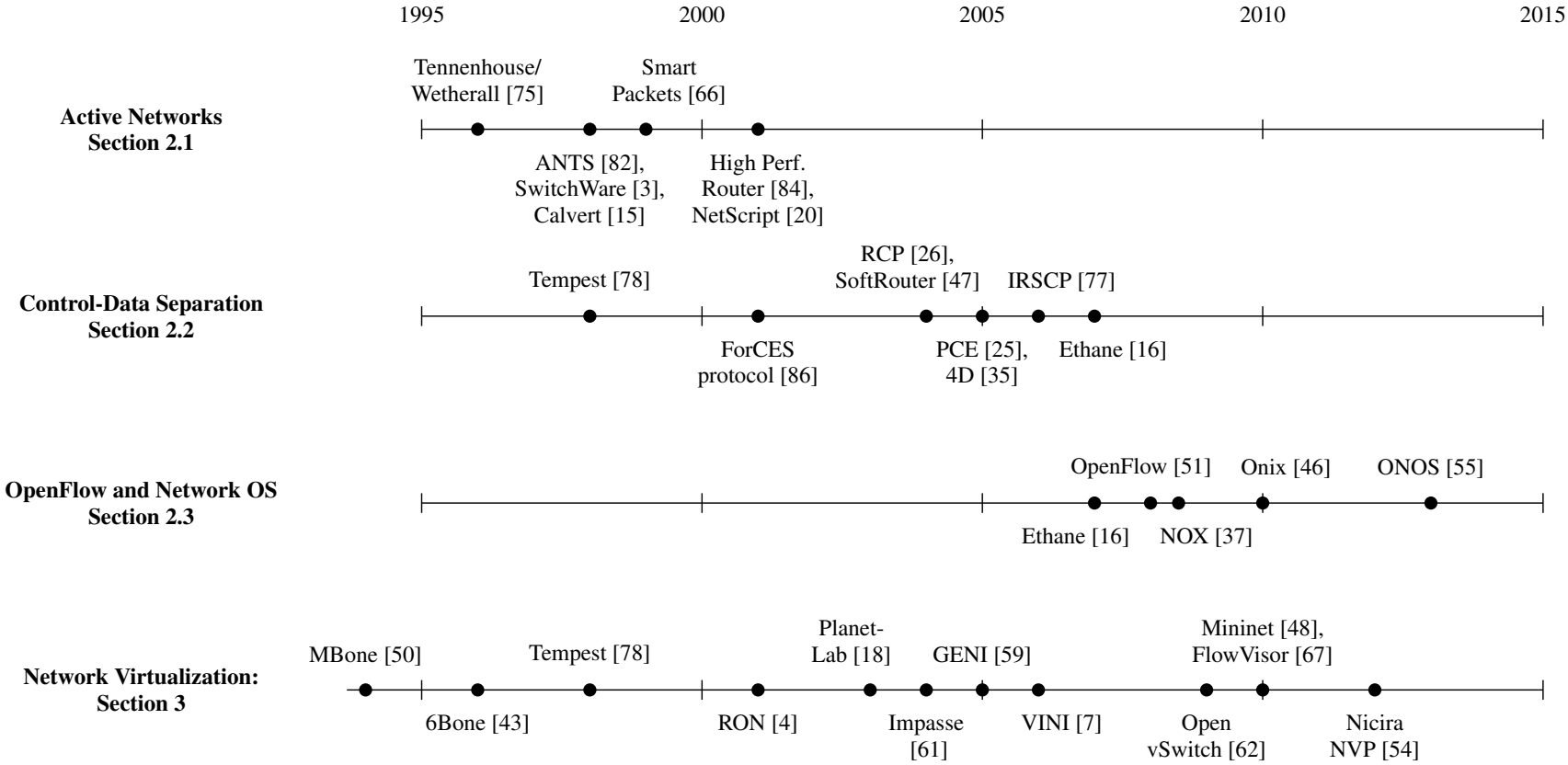
Redes Tradicionais

- Grande volume de dados;
- Múltiplos serviços para suportar;
- Aplicações e Serviços de natureza dinâmica.

Necessidade de tornar as redes flexíveis e programáveis

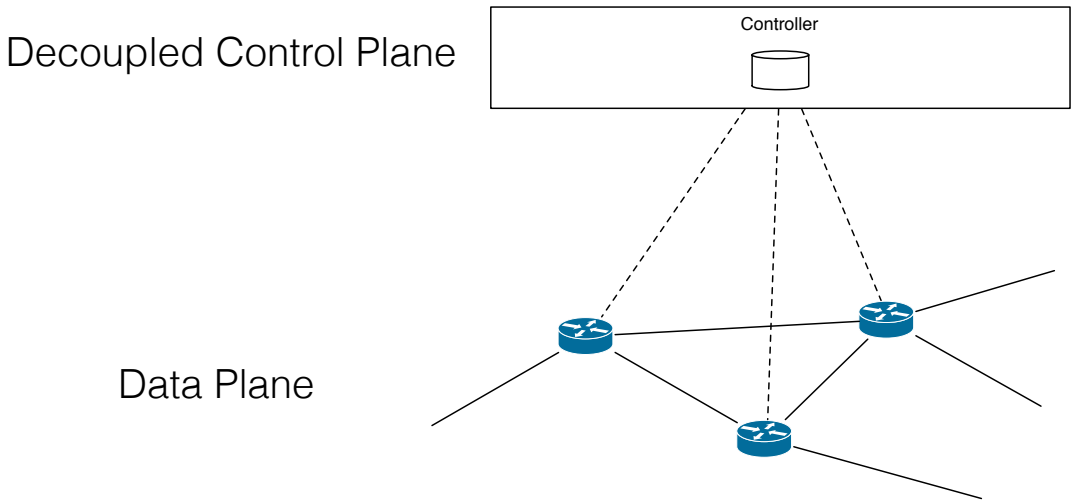
Introdução

Evolução das Redes Programáveis



Redes Definidas por Software

ONF: Uma arquitetura emergente, dinâmica, gerenciável e adaptável, que permite a separação entre o controle da rede e as funções de encaminhamento.



Introdução
Openflow

- Separação entre plano de controle e plano de dados;
- Centralização do controle;
- Controle baseado em fluxos.

OF Version	Release Date	Match fields
1.0	Dec 2009	12
1.1	Feb 2011	15
1.2	Dec 2011	36
1.3	Jun 2012	40
1.4	Oct 2013	41
1.5	Dec 2014	44

- Limitado aos protocolos existentes;
- Campos de cabeçalho fixos;
- Sem suporte para protocolos personalizados;
- Ex: NVGRE, VXLAN, STT.

Processamento dos pacotes: bottom-up

- Funções encapsuladas nos ASIC's dos dispositivos;
- Funções Fixas;
- Elevado tempo de espera para implementação de funções novas;
- Colocar novas ideias em produção depende dos vendedores/fabricantes;
- Impedimento para inovação.

Necessidade de dispositivos de Rede programáveis

- CPUs, FPGAs
- PISA Chips (Protocol Independent Switch Architecture)
  - Processamento de pacotes com programabilidade completa do *parser* e lógica genérica para *match-action*;
  - Processador de domínio específico;
  - Comparável com ASICs em relação ao tamanho e custo.

Introdução

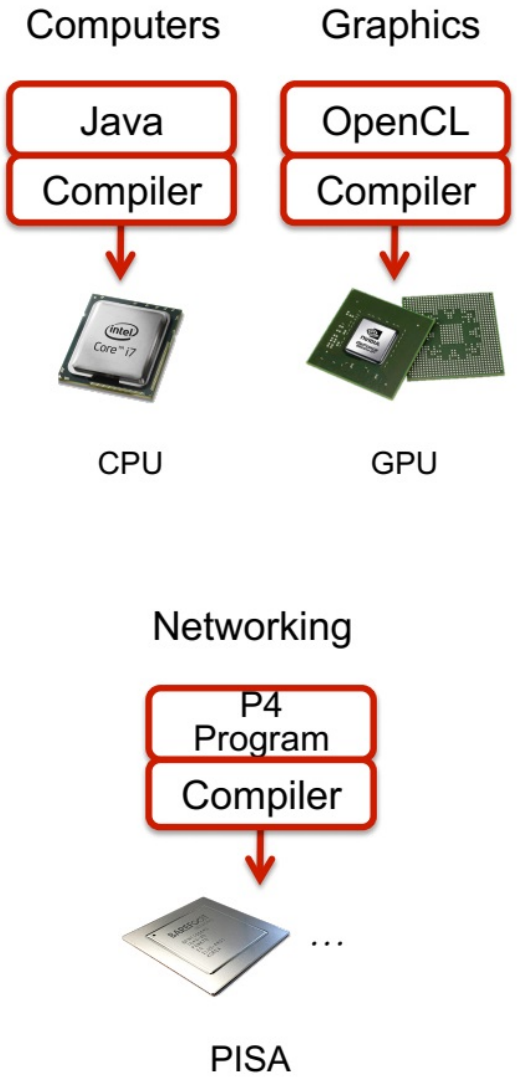
Benefícios da programabilidade do plano de dados

- O controle total da programabilidade do plano de dados permite que um dispositivo de rede programável se comporte exatamente como desejado;
- Capacidade de adicionar novas funções, uma vez que o dispositivo é completamente programável;
- Exclusividade e Inovação, pois os usuários têm a capacidade de implementar protocolos personalizados.

- Eficiência, uma vez que, considerando que os dispositivos atualmente têm capacidade limitada e funções fixas, muitos dessas não são usadas e, portanto, consomem recursos sem nenhuma finalidade;
- A confiabilidade constitui outro benefício, porque é possível não usar funções implementadas por terceiros;
- O monitoramento no plano de dados permite olhar para dentro do dispositivo de forma personalizada.

Programabilidade - Ideia

Implementar mecanismos flexíveis para analisar pacotes e campos de cabeçalho correspondentes (arbitrários) por meio de uma interface comum, em vez de estender repetidamente o padrão OpenFlow.


Computers
Graphics
Java
Compiler
OpenCL
Compiler
CPU
GPU
Networking
P4
Program
Compiler
...
PISA

Programabilidade

Atualmente, indústria e academia convergem em uma nova linguagem de programação de domínio específico, denominada P4 (*Programming Protocol-Independent Packet Processors*)

- Reconfigurabilidade: está relacionada com a habilidade do programador redefinir o processamento dos pacotes nos dispositivos de rede;
- Independência do Protocolo: define que os dispositivos de rede não devem ser amarrados a formatos de pacotes específicos;
- Independência do Alvo: relaciona-se com o fato de que o compilador da linguagem de programação de redes deve gerar uma descrição independente do dispositivo.

P4

P4 é uma linguagem de domínio específico para expressar como os pacotes são processados pelo plano de dados de um dispositivo de rede programável

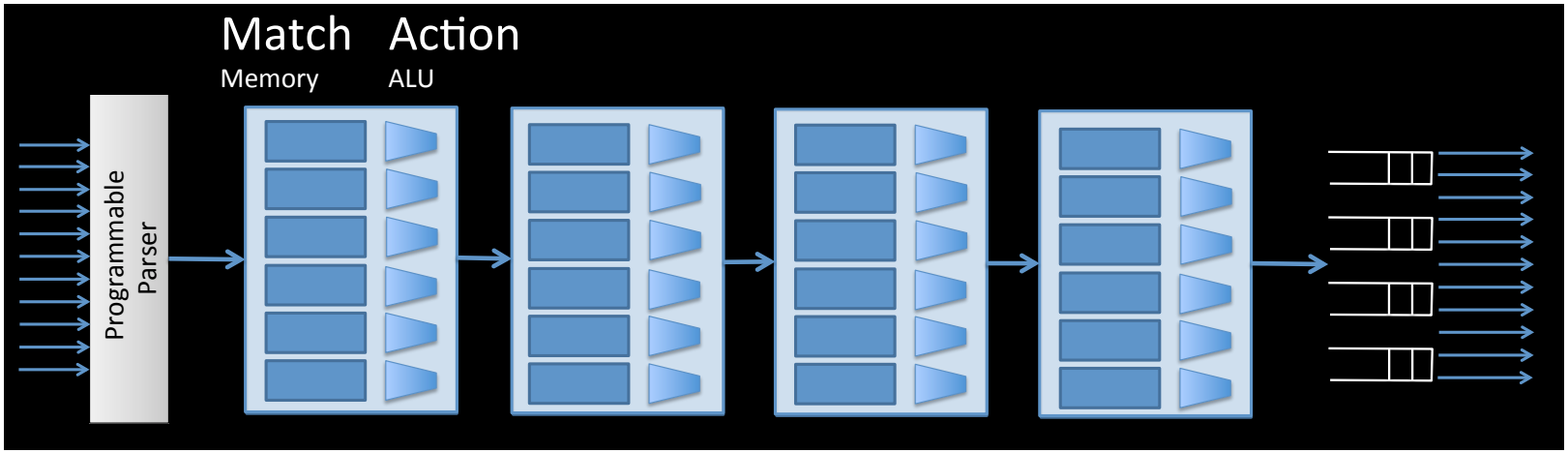
A Linguagem de Programação P4

Breve Historia

- Foi publicado o primeiro artigo no SIGCOMM em 2014 [Bosshart et al. 2014] quando surgiu a primeira versão, a $P4_{14}$ (1.0.x);
- A versão mais recente da linguagem, $P4_{16}$ (1.2), foi lançada em 2016 com mudanças significativas, dentre elas:
  - Tipagem forte;
  - Capacidade de indicar uma ordem na qual as ações devem ser executadas;
  - Modelo flexível visando a independência de alvos (ASICs,FPGA, NICs, software *switches*).

A Linguagem de Programação P4

Arquitetura de Switches Independentes de Protocolo (PISA)



- Arquitetura para encaminhamento programável de pacotes. Componentes principais:
  - Parser;
  - Match-action;
  - Deparser;

- Parser Programável: o *parser* tem por função identificar, em uma cadeia (stream) de bits, os cabeçalhos que encontram-se presentes no pacote. Ou seja, permite especificar o formato para processamento e identificar os protocolos contidos no pacote;
- Match-Action: nessa etapa faz-se um *match*, ou seja a correspondência entre um campo e um valor, e executa-se uma ação de acordo com as entradas que encontram-se associadas a esses *matches* nas tabelas;

- Deparser: nessa etapa monta-se de novo o pacote, para enviá-lo como um *stream* de bits para a saída;
- Metadados: os metadados contém informações necessárias ao processamento mas que não estão no pacote, por exemplo, a identificação da porta de ingresso de um determinado pacote em um *switch*;

Fluxo dos pacotes na arquitetura PISA:

O *parser* identifica os cabeçalhos presentes em cada pacote ingressando no dispositivo. Realiza uma busca (*lookup*) em cada tabela *match-action* por um subconjunto de campos de cabeçalho e aplica as ações correspondentes ao primeiro *matching* encontrado na tabela.

- *Headers*: A definição do cabeçalho descreve a sequência e estrutura de uma série de campos de bits definidos pelo programador. Inclui especificações de largura, restrições de tipos e valores.
- *Parsers*: A definição de um *parser* especifica como identificar (reconhecer) cabeçalhos ou sequências de cabeçalho válidos nos pacotes.

- *Tables*: As tabelas *match-action* representam o mecanismo para realizar o processamento dos pacotes. As tabelas associam ações (*actions*) aos pacotes de acordo com o *matching* realizado nos *parsers*.
- *Actions*: P4 suporta construção de ações complexas construídas usando primitivas simples e independentes de protocolo.

Em $P4_{16}$ podem-se encontrar 5 categorias principais de elementos da linguagem:

- Tipos de Dados: P4 tem um conjunto de tipos de dados básicos que servem para a construção de tipos mais complexos, tais como *arrays*, *headers* e *structures*;
- Expressões: constituídas por uma lista de operadores e operações básicas;

- Elementos de Fluxo de Controle: são elementos e estruturas para expressar o fluxo de dados entre as tabelas *match-action*;
- *Parsers*: elementos de programação de máquina de estados e extração de campos;
- *Externs*: conjunto de bibliotecas para suporte de funções especializadas.

Elementos da linguagem P4

Headers

O *Header* constitui o mais importante dos tipos de dados. Pode conter os tipos básicos: *bit*, *int* e *varbit*. A declaração de um cabeçalho é dada pela sintaxe a seguir:

```
header HeaderName {
    fields.......
}
```

Por exemplo, pode-se definir o cabeçalho Ethernet da seguinte forma:

```
header Ethernet_h {
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16> etherType;
}
```

Elementos da linguagem P4

Headers

A seguinte declaração de variável, ou instanciação, usa o tipo *Ethernet_h* definido anteriormente:

```
Ethernet_h ethernetHeader;
```

A Struct é uma estrutura composta. Nesse caso, a estrutura é formada por campos que são cabeçalhos (*headers*) definidos previamente. Um exemplo deste tipo de definição é o seguinte:

```
header Tcp_h { ... }
header Udp_h { ... }
struct Parsed_headers {
Ethernet_h ethernet;
Ip_h       ip;
Tcp_h      tcp;
Udp_h      udp;
}
```

Elementos da linguagem P4

Operadores e Expressões

Lista de operadores e conjunto de expressões parecidas com outras linguagens de uso geral, como C: `+`, `-`, `*`, `<<`, `>>`, `&`, `|`, `~`, `^`, `&&`, `||`, `!`, `<`, `<=`, `>`, `>=`, `==`, `!=`

Operação match-action:

- Controles
  - Parecidos com funções em C;
  - Sem *loops* e recursão;
  - Possibilidade de uso de declarações *if* e/ou declarações do tipo *switch*;
  - Um bloco de controle é declarado com um nome, parâmetros e uma sequência de declarações de constantes, variáveis, ações e tabelas.

Elementos da linguagem P4

Match-action

Operação *match-action*:

- Tabelas
  - Associa as chaves com com ações específicas (*matching*);
  - Utiliza a sentença *apply* para associar o *matching* às ações que estão contidas em uma tabela;
- Ações
  - São exemplos de ações primitivas: *add_to_field*, *drop*, *add_header*, *modify_field*, *remove_header*

```
control ingress
{
apply(nome_tabela);
}
control egress {
}
```

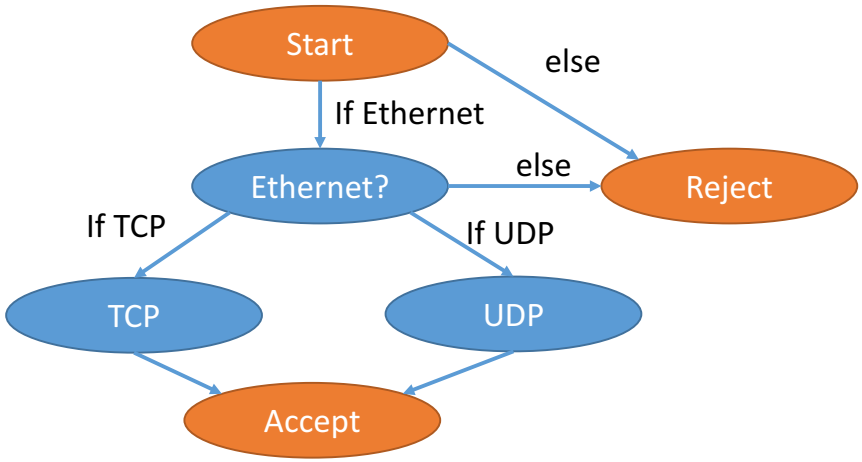
Elementos da linguagem P4

Parser

São funções especiais escritas em um estilo de máquina de estados finita que basicamente buscam converter ou mapear fluxos de bits em cabeçalho.
Possui um conjunto de estados predefinidos:

- start;
- accept;
- reject.

Os estados definidos pelo programador descrevem como o pacote será processado pelo *parser*.



Seções de um Programa P4

Declaração de Dados

Basicamente, um programa P4 começa com a definição do cabeçalho (*header*). Exemplo: quadro (frame) Ethernet.

- Definição de tipos;
- Organização dos campos de endereço destino, endereço de origem e ethertype.

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

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

Seções de um Programa P4

Declaração de Dados

Definição de um cabeçalho IPv4:

```
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;
}
```

Seções de um Programa P4

Parser Programável

Lógica do parser: identificar a transição e extrair os dados dos cabeçalhos Ethernet e IPv4 em um pacote. Exemplo:

Seções de um Programa P4

Parser Programável

```
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;
    }
```

Tabelas:

- Contém as chaves com as quais procura-se fazer o *matching* assim como as ações associadas a essas chaves;
- Os tipos de *matching* são: exact, lpm, ternary;
- Ações são basicamente funções compostas por ações primitivas;
- Exemplo: em um parser IPv4, pode-se fazer o *matching* com o endereço IP de destino (via *Longest Prefix Matching* - lpm) e as ações possíveis são *setnhop* (definir o próximo salto) e *drop* (descarte).

Seções de um Programa P4

Tabelas


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

Seções de um Programa P4

Preenchimento das Tabelas

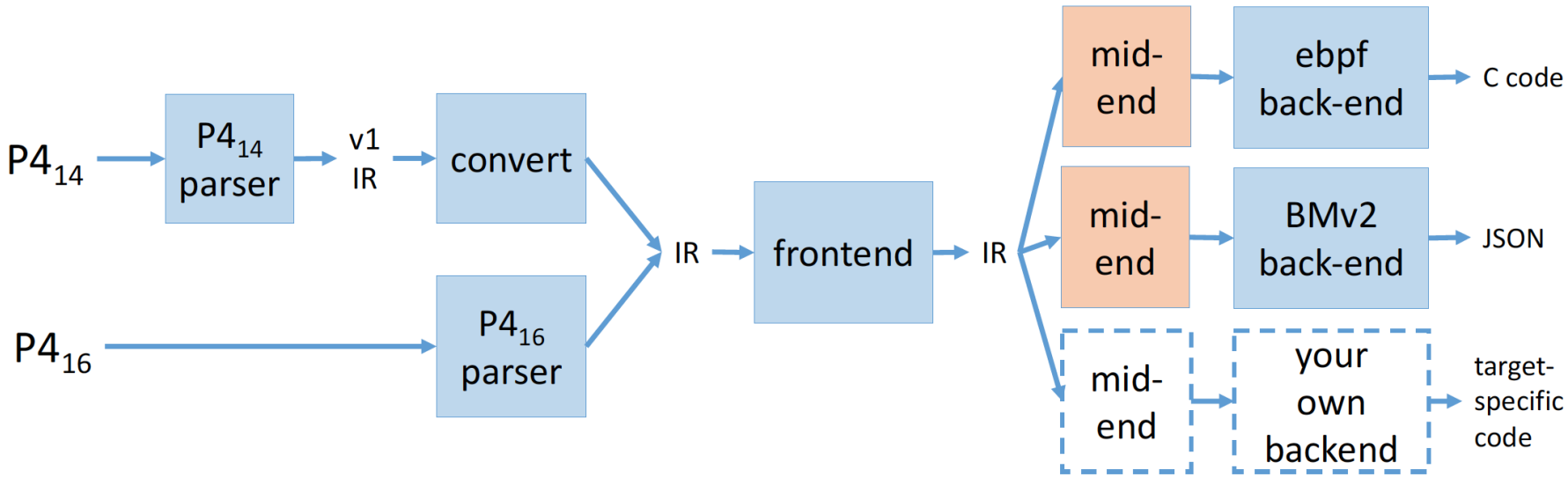
- O BMv2 vem com uma ferramenta CLI para operação em tempo de execução: `./simple_switch_CLI --thrift-port [port_number]`;
- *bmv2/tools/runtime_CLI.py* é uma maneira simples de adicionar entradas para tabelas *MMatch-action*, cujo formato é o seguinte:

```
table_set_default <table name> <action name> <action
    parameters>
table_add <table name> <action name> <match fields> => <
    action parameters> [priority]
table_delete <table name> <entry handle>
```

- A cada 2 anos são geradas novas especificações da linguagem e do compilador de referência;
- A versão corrente da linguagem P4, considerada estável, é a P4_16;
- Necessidade de manutenção da retro-compatibilidade, bem como permitir uma extensibilidade flexível;
- Suportar um modelo de abstração de hardware (*frontend*) único e estável.

Compilando Programas na Linguagem P4

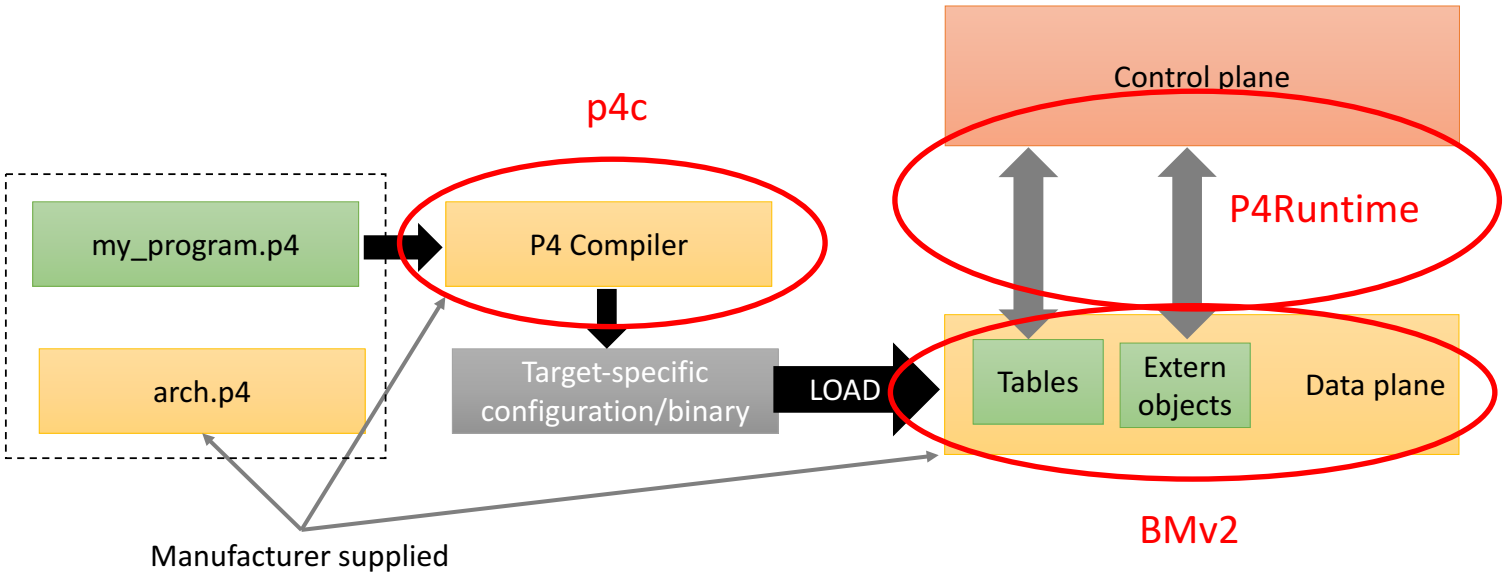
Fluxo do compilador



Compilando Programas na Linguagem P4

Fluxo para compilar um programa .p4

- Objetivo do compilador: mapear a descrição de como serão processados os pacotes (em programa ".p4") para um dispositivo alvo (*target*).
- Ferramentas:
  - p4c (compilador);
  - BMV2 (software *switch*, *target*)



Compilando Programas na Linguagem P4

Compilador p4c

BMv1

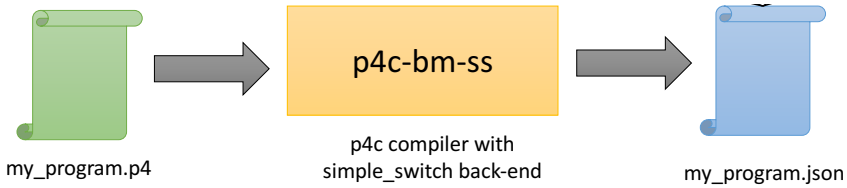
- Primeiro compilador: p4c-behavioral;
- Esse compilador gerava um código C para cada programa P4;
- Era novamente compilado para um executável usando o gcc.

BMv2

- Compilado para uma representação JSON (p4c-bm);
- Este JSON é então carregado no alvo (BMv2) e as estruturas de dados são inicializadas para refletir o comportamento de encaminhamento desejado.

Compilando Programas na Linguagem P4

Fluxo de compilação



- Transformar o código P4 em uma representação JSON que pode ser consumida pelo software *switch*;
- Esta representação dirá ao BMv2 quais tabelas inicializar e como configurar o parser;
- `p4c-bm -json <path to JSON file> <path to P4 file>`
- Com o p4c-bm2-ss compila-se diretamente para o simple_switch
- `p4c-bmv2-ss -o program.json program.bmv2.p4`

Conclusões

- A separação entre o plano de controle e o plano de dados foi estabelecida pelo SDN;
- A programabilidade da rede ainda era papel quase exclusivo do plano de controle;
- A expressividade requerida para a programabilidade de rede no plano de dados sofria com a falta de bibliotecas e módulos específicos;

Conclusões

- Busca-se redução do tempo requerido prototipação de serviços de rede para acelerar os ciclos de inovação;
- Surge a linguagem P4 para, finalmente, habilitar programabilidade do plano de dados de dispositivos de rede;
- Um dos principais atrativos é que permite as descrições de funcionalidades de rede independentes da plataforma alvo.

Conclusões

- a expressividade da linguagem também viabiliza novas soluções para problemas clássicos: Estruturas de dados probabilísticas (sketches), Telemetria, Balançamento de carga (HULA), etc.
- número crescente de compiladores para a linguagem P4, o que pode ser um importante sinal de interesse da comunidade;
- A incorporação de programabilidade em P4 às interfaces de rede de servidores já é uma realidade via SmartNICs;

Conclusões

- Integração com o OvS: O projeto PISCES é um protótipo independente que pode ser programado a partir da linguagem P4

Parte Prática

Introdução

- Vamos colocar em prática o que vimos até agora através de alguns exercícios de programação em P4.
- Nós preparamos a imagem de uma VM com todo o software que você irá precisar para executar os exercícios. **Se você não fez o download da VM, por favor, nos avise!**
- Trabalhe no seu ritmo. Se você precisar de ajuda, peça a um dos instrutores ou a um dos seus vizinhos.

Começando

- **O username para a VM é student e a senha é student.**
- Os exercícios estão no diretório ~/labs do usuário student.
- As informações necessárias para executar os exercícios estão no material do curso, no capítulo de laboratórios

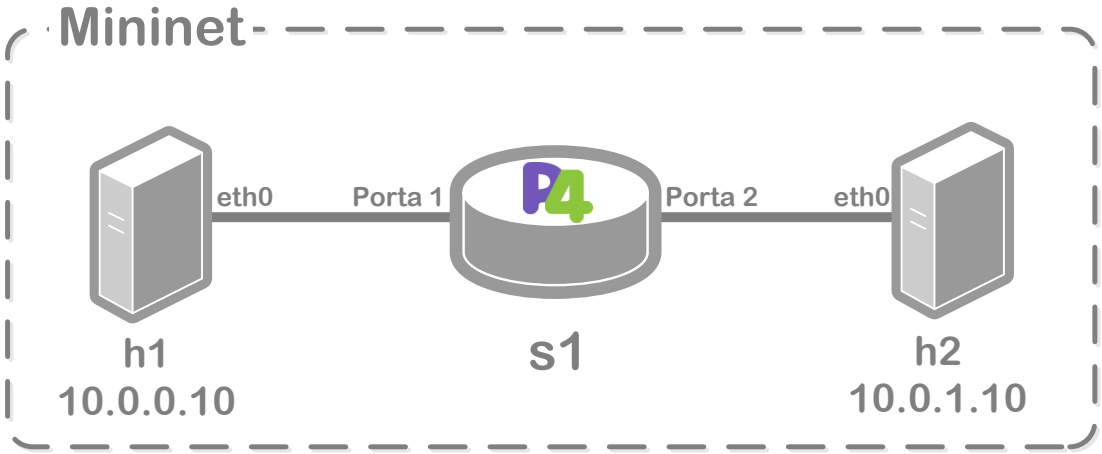
Visão Geral

- São propostos três exercícios práticos:
  - **Encaminhamento de Pacotes**: encaminhamento para IPv4.
  - **Calculadora**: implementação de uma calculadora.
  - **Roteamento de Origem**: esquema de roteamento determinado na origem do pacote.
- Para cada exercício haverá um código inicial e uma estrutura para teste do programa.
- Siga as instruções passo a passo no material do minicurso.

Atividade 1

Simple Route

O objetivo desta atividade é simplesmente encaminhar um pacote recebido de um *host* para outro. O diagrama apresentado na figura demonstra a topologia configurada para essa tarefa.



Atividade 1

Simple Route

Para simplificar ainda mais, deve ser implementado somente a lógica de encaminhamento para o protocolo IPv4. Para isso, o elemento de encaminhamento ou *switch*, deverá desempenhar as seguintes ações para cada pacote recebido:

- Atualização do endereço MAC de origem e destino;
- O decremento do valor do TTL (*Time To Live*) do pacote;
- Encaminhamento do pacote para a porta de saída apropriada.

Atividade 1

Simple Route

O *switch* P4 contará com uma única tabela, a qual será populada com regras estáticas implementadas pelo plano de controle usando CLI. Cada uma das regras mapeia um endereço IP para o endereço MAC e a porta de saída para o próximo salto. Essas regras já foram previamente implementadas, então será necessário somente a implementação da lógica do plano de dados através do código P4.

Atividade 1

Simple Route - Passo 1 - Execução Inicial

O diretório `/home/student/labs/task_1/p4src` contém o código P4 `task_1.p4`, que pode ser compilado. No entanto, em seu estado atual ele simplesmente descarta todos os pacotes que chegam até o dispositivo. O objetivo dessa atividade é completar esse código inicial de forma que o *switch* P4 faça o encaminhamento do pacote ao *host* correto. Primeiramente, compile o código de forma que está e execute o Mininet para testar o seu funcionamento.

Atividade 1

Simple Route - Passo 1 - Execução Inicial

Na linha de comando, dentro do diretório `/home/student/labs/task_1/` execute o *script*: `sudo ./run_task.sh`. Esse *script* irá compilar o programa `task_1.p4` e inicializar o Mininet com a topologia configurada para a atividade.
A instância do Mininet para essa atividade conta com um *switch* ($s1$) e dois *hosts* ($h1$ e $h2$). Os *hosts* possuem os seguintes endereços IPs associados: $h1$ - 10.0.0.10 e $h2$ - 10.0.1.10.

Atividade 1

Simple Route - Passo 1 - Execução Inicial

Uma vez que a instância do Mininet esteja carregada, obtem-se o *prompt* >. Abra então terminais para *h*1 e *h*2, respectivamente:

```
mininet> xterm h1 h2
```

Em cada um dos *hosts* existem aplicações baseadas em Python para enviar e receber mensagens. Inicie primeiramente a aplicação para receber mensagens no terminal do *host* 2 (*h*2).

```
sudo ./receive.py
```

Atividade 1

Simple Route - Passo 1 - Execução Inicial

Em seguida, inicie a aplicação para enviar mensagem a partir do *host* 1 (*h*1). A sintaxe do comando é `send.py <ip do receiver> <‘‘mensagem’’>`:

```
sudo ./send.py 10.0.1.10 "Olá mundo P4!"
```

A mensagem não será recebida.
Saia do emulador digitando `exit` na linha de comando.

```
mininet> exit
```

Atividade 1

Simple Route - Plano de Controle

O programa em P4 define o *pipeline* de processamento dos pacotes, mas as regras em cada uma das tabelas são inseridas através do plano de controle. Quando uma regra encontra um pacote correspondente, sua ação é invocada com os parâmetros fornecidos pelo plano de controle.
Nessa atividade, a lógica do plano de controle já foi implementada. Assim que o Mininet é instanciado, as regras para o processamento dos pacotes são instaladas nas tabelas do *switch* P4.

Atividade 1

Simple Route - Plano de Controle

Essas regras estão definidas no arquivo `command_s1.txt`.

```
table_set_default ipv4_lpm drop
table_add ipv4_lpm ipv4_forward 10.0.0.10/32 => 00:04:00:00:00:00 1
table_add ipv4_lpm ipv4_forward 10.0.1.10/32 => 00:04:00:00:00:01 2
```

Atividade 1

Simple Route - Plano de Controle

O código P4 também define a interface entre o *pipeline* do *switch* e o plano de controle. Os comandos CLI no arquivo `command_s1.txt` referem-se a tabelas, chaves e ações, especificadas pelo nome e qualquer alteração que ocorra no programa P4 que adicione ou renomeie uma tabela, chave ou ação, precisa ser refletida nestes comandos.

Atividade 1

Simple Route - Passo 2: Implementação

O arquivo `task_1.p4` contém o código P4, com as principais partes da lógica a ser implementada, assinaladas com o comentários **PARA COMPLETAR**. A implementação a ser feita deverá seguir a estrutura proposta nesse arquivo, substituindo os comentários pelo código P4 necessário para que a implementação funcione.

Atividade 1

Simple Route - Passo 2: Implementação

O arquivo `task_1.p4` completo deve ter os seguintes componentes:

1 - Definições de cabeçalho para as camadas *ethernet* (`ethernet_t`) e IPv4 (`ipv4_t`).
2 - **PARA COMPLETAR:** Os *parsers* para *ethernet* e IPv4 que populem os campos de `ethernet_t` e `ipv4_t`.
3 - Pelo menos uma ação para descartar os pacotes não reconhecidos usando `mark_to_drop()`.

4 - **PARA COMPLETAR:** Uma ação (chamada `ipv4_forward`) que:

a) Determina a porta de saída para o próximo salto;

b) Atualiza o endereço *ethernet* de destino com o endereço do próximo salto;

c) Atualiza o endereço *ethernet* de origem, com o endereço *ethernet* do *switch*;

d) Decrementa o TTL.

5 - **PARA COMPLETAR:** Um controle que:

a) Defina a tabela para ler o endereço IPv4 de destino e invoque a ação `drop` ou a ação `ipv4_forward`;

b) Defina um bloco `apply` que referencie a tabela.

6 - **PARA COMPLETAR:** Defina um `deparser` que selecione a ordem que cada campo deve ser inserido no pacote que está saindo do *switch* e um instanciação do pacote composta por `parser`, `control` e `deparser`.

Eventualmente esse processamento também requer instâncias de verificação de *checksum* e controles. Isso não será necessário para essa tarefa e foram substituídas por controles vazios.

Atividade 1

Simple Route - Passo 3: Execução Final

Siga as instruções do Passo 1 para testar. Após as alterações de código as mensagens do *host h*1 deverão ser entregues ao *host h*2.

Atividade 1

Simple Route - Reflexões e Desafios

Os programas utilizados para testar a sua implementação não são muito robustos e não suportam o envio de um tráfego mais elevado. Qual seria outra forma de testar a implementação? Além disso, outras questões podem ser consideradas: Como você poderia melhorar o seu código para suportar vários saltos? Essa implementação é suficiente para substituir um roteador comercial? O que estaria faltando?

Atividade 1

Simple Route - Solução de Problemas

Existem vários problemas que podem acontecer durante o desenvolvimento da sua implementação:

- O código `task_1.p4` pode falhar durante a compilação. Nesse caso o *script* `run_task_1.sh` apresentará na tela o erro de compilação e será terminado.
- O código `task_1.p4` pode compilar, mas pode falhar ao carregar as regras do plano de controle contidas no arquivo `command_s1.txt`, que o *script* `run_task.sh` tentará instalar usando o CLI BMv2. Neste caso, o `run_task.sh` irá gerar mensagens de erro sobre a saída do CLI no diretório de *logs*. Utilize essa mensagem de erro para consertar a sua implementação.

Atividade 1

Simple Route - Solução de Problemas

- O código `task_1.p4` pode compilar e as regras do plano de controle descritas no arquivo `command_s1.txt` podem ser carregadas sem problemas, mas ainda assim, o *switch* P4 pode não conseguir processar os pacotes corretamente. O arquivo `/tmp/p4s.s1.log` contém informações detalhadas descrevendo como o pacote foi processado pelo *switch*. Este *log* pode ajudar a encontrar erros de lógica na sua implementação.

Reiniciando o Mininet

Alguns casos de falha podem ser causados devido a inicialização do ambiente. Nestes casos, reiniciar o Mininet pode ajudar. Utilize o comando `mn -c` para terminar as instâncias que ficaram travadas.

Atividade 2

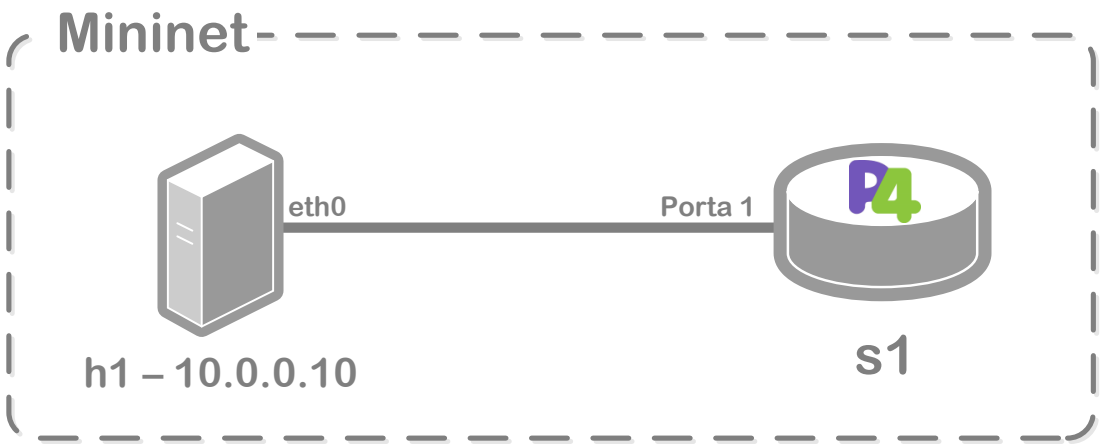
Implementando um calculadora

O objetivo desta atividade é implementar uma calculadora básica utilizando um cabeçalho customizado escrito em P4. O cabeçalho deve conter a operação a ser executada e dois operandos. Quando o *switch* receber o cabeçalho para calcular, ele deverá executar a operação com os operandos e retornar ao endereço de origem o resultado. Muito embora esta atividade pode não possua uma aplicação efetivamente prática, ela é muito útil para demonstrar o potencial da linguagem P4 para definir novos cabeçalhos.

Atividade 2

Implementando um calculadora

O diagrama apresentado na figura demonstra a topologia configurada para essa tarefa.



.

Atividade 2

Implementando um calculadora - Passo 1: Execução Inicial

O diretório `/home/student/labs/task_2/p4src` contém o código P4 `task_2.p4`, e pode ser compilado. No entanto, em seu estado atual ele simplesmente descarta os pacotes que chegam até o *switch*. O objetivo dessa atividade é completar esse código inicial de forma que o *switch* P4 faça o cálculo de acordo com as informações recebidas no cabeçalho e devolva o pacote à sua origem.

Atividade 2

Implementando um calculadora - Passo 1: Execução Inicial

Primeiramente compile o código da forma que se encontra e execute o Mininet para testar o seu funcionamento.

Na linha de comando, dentro do diretório `/home/student/labs/task_2/` execute o *script*: `sudo ./run_task.sh`. Esse *script* irá compilar o programa `task_2.p4` e inicializar o Mininet já com a topologia configurada para a atividade.

A instância do Mininet para essa atividade conta com um *switch* ($s1$) e um *host* ($h1$). O *host* tem o seguintes endereço IP associado: h1 - 10.0.0.10

Atividade 2

Implementando um calculadora - Passo 1: Execução Inicial

Para essa atividade um pequeno programa em Python foi escrito para testar o funcionamento da sua calculadora. O programa será carregado no *host* *h*1 diretamente no *prompt* e permitirá a execução dos testes. O *script* proverá um novo *prompt*, no qual você poderá escrever expressões matemáticas básicas. Essa aplicação vai verificar a expressão e preparar um pacote com o operador e os operandos correspondentes, e então, enviará esse pacote para o *switch* P4.

Atividade 2

Implementando um calculadora - Passo 1: Execução Inicial

Quando o *switch* retornar o pacote com o resultado da operação, a aplicação imprimirá na tela o valor. Uma vez que a lógica em P4 ainda não foi implementada, você deve ver uma mensagem de erro na tela, conforme exemplo a seguir:

```
> 1+1
Didn't receive response
>
```

Atividade 2

Implementando um calculadora - Passo 2: Implementação

Para a implementação da calculadora, será necessário definir um cabeçalho customizado (figura) e definir a lógica no *switch* P4 para verificar o cabeçalho, efetuar a operação indicada, escrever o resultado no cabeçalho e retornar o pacote ao seu emissor.

.

Atividade 2

Implementando um calculadora - Passo 2: Implementação

O seguinte formato de cabeçalho foi utilizado para essa atividade:

```
P    o caracter ASCII 'P' (0x50)
4    o caracter ASCII '4' (0x34)
Versao e atualmente a 0.1 (0x01)
Op e a operacao a ser efetuada:
'+' (0x2b) Resultado = Operando A + Operando B
'-' (0x2d) Resultado = Operando A - Operando B
'&' (0x26) Resultado = Operando A & Operando B
'|' (0x7c) Resultado = Operando A | Operando B
'^' (0x5e) Resultado = Operando A ^ Operando B
```

Atividade 2

Implementando um calculadora - Passo 2: Implementação

Assume-se que o cabeçalho a ser calculado será carregado sobre o protocolo *ethernet*, e que será usado o tipo de *ethernet* 0x1234 para indicar a presença desse cabeçalho. Dado o que foi aprendido até agora, o objetivo é implementar o código P4 para fazer os cálculos. Não existe lógica do plano de controle, então só é necessário a implementação da lógica para o plano de dados.

Atividade 2

Implementando um calculadora - Passo 2: Implementação

Uma implementação correta deverá verificar os cabeçalhos customizados, executar as operações matemáticas, escrever o resultado no campo apropriado e retornar o pacote ao emissor do mesmo.

Atividade 2

Implementando um calculadora - Passo 3: Execução Final

Siga as instruções do Passo 1. Agora, ao invés de uma mensagem de erro, deverá aparecer o resultado correto da operação.

```
> 1+1
2
>
```

Atividade 3

Source Router - Implementação Inicial

O objetivo desta atividade é implementar um esquema de roteamento estabelecido na origem do pacote. Com o roteamento na origem, o *host* de origem especifica para qual porta cada *switch* na rede deverá encaminhar o pacote em cada salto. O *host* de origem estabelece uma série de portas de saída no cabeçalhos do pacote. Neste exemplo, será introduzida a lista das portas para encaminhamento após o cabeçalho *ethernet* e será estabelecido um *etherType* especial para indicar a presença dessa lista. Cada *switch* deverá tomar um item dessa lista e encaminhar o pacote de acordo com o número da porta especificada.

Atividade 3

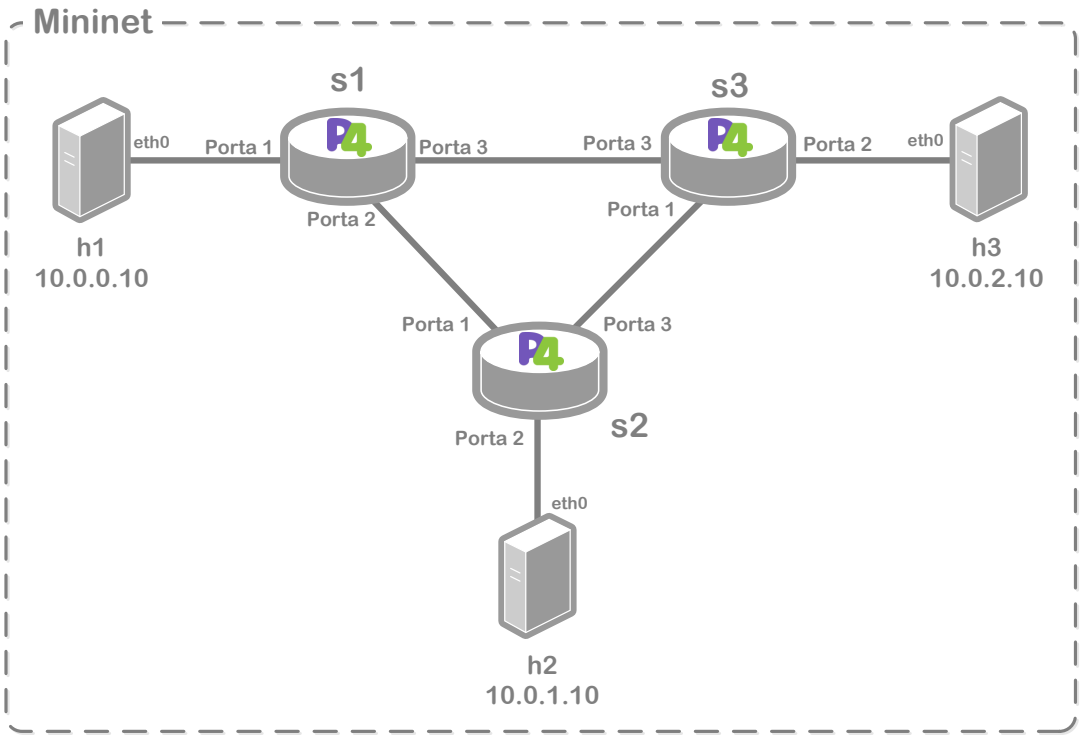
Source Router - Implementação Inicial

O *switch* P4 deve verificar a lista de roteamento de origem. Cada item tem um *bit* de fim de lista e o número da porta. O *bit* de fim lista será igual a 1 somente no último item da lista. Então, no ingresso do pacote, é necessário tomar um item da lista e configurar a porta de saída de acordo com o valor do item tomado. Note que no último salto o pacote deve ser revertido ao *etherType* `TYPE_IPV4` antes de ser entregue.

Atividade 3

Source Router

O diagrama apresentado na figura demonstra a topologia configurada para essa tarefa.



.

Atividade 3

Source Router - Passo 1: Execução

O diretório `/home/student/labs/task_3/p4src` contém o código P4 `task_3.p4`, que pode ser compilado, no entanto, em sua versão original, simplesmente descarta os pacotes que chegam até ele. O objetivo dessa atividade é completar esse código inicial, de forma que o *switch* P4 faça o encaminhamento do pacote ao *host* correto.

Primeiramente, compile o código de forma em que se encontra e execute o Mininet para testar o seu funcionamento.

Atividade 3

Source Router - Passo 1: Execução

Na linha de comando, dentro do diretório `/home/student/labs/task_3/` execute o *script*: `sudo ./run_task.sh`. Esse *script* irá compilar o código `task_3.p4` e inicializar o Mininet já com a topologia configurada para a atividade.

A instância do Mininet para essa atividade conta com três *switches* ($s1$, $s2$ e $s3$) e três *hosts* ($h1$, $h2$ e $h3$). Os *hosts* possuem os seguintes endereços IPs associados: $h1$ - 10.0.0.10, $h2$ - 10.0.1.10 e $h3$ - 10.0.2.10.

Atividade 3

Source Router - Passo 1: Execução

Em seguida, inicie a aplicação para enviar mensagem a partir do *host* 1 (*h*1). A sintaxe do comando é `send.py <ip do receiver>`:

```
sudo ./send.py 10.0.1.10
```

Digite então a lista de roteamento de origem, digamos, `2 3 2 2 1`. Essa lista deveria fazer com que o pacote percorra uma rota passando por: *h*1, *s*1, *s*2, *s*3, *s*1, *s*2 e *h*2. Entretanto o pacote não será recebido por *h*2.

Atividade 3

Source Router - Passo 1: Execução

Saia do *prompt* do *script* Python digitando `q` e então digite `exit` para sair do `xterm` e voltar a linha de comando do Mininet. Então saia do Mininet digitando `exit` na linha de comando.

```
mininet> exit
```

A mensagem não foi entregue porque o *switch* está programado com o código P4 original em `task_3.p4`, que descarta todos os pacotes recebidos. O objetivo dessa tarefa é estender o código P4 para que os pacotes consigam alcançar o seu destino.

Atividade 3

Source Router - Passo 2: Implementação

O arquivo `task_3.p4` contém o código P4 com indicações das partes a serem implementadas, assinaladas com o comentários **PARA COMPLETAR**. A implementação a ser feita deverá seguir a estrutura proposta nesse arquivo, substituindo os comentários pelo código P4 necessário para que a implementação funcione.

Atividade 3

Source Router - Passo 2: Implementação

O arquivo `task_3.p4` completo deve ter os seguintes componentes:

1 - Definições de cabeçalho para as camadas *ethernet* (`ethernet_t`), IPv4 (`ipv4_t`) e o roteamento de origem (`srcRoute_t`).
2 - **PARA COMPLETAR:** Os *parsers* para *ethernet*, IPv4 e roteamento de origem que populem os campos de `ethernet_t`, `ipv4_t` e `srcRoute_t`.
3 - Pelo menos uma ação para descartar os pacotes, usando `mark_to_drop()`.

4 - **PARA COMPLETAR:** Uma ação (chamada `srcRoute_nhop`) que irá:

a) Determinar a porta de saída para o próximo salto;

b) Remover o primeiro item da lista roteamento de origem.

5 - Um controle com um bloco de aplicação que:

a) Verifica a existência das rotas de origem.

b) **PARA COMPLETAR:** Verifica se o `ethernet.etherType` deve ser alterado, caso o pacote esteja sendo enviado ao último *host*.

Atividade 3

Source Router - Passo 2: Implementação

6 - **PARA COMPLETAR:** Chamar a ação `srcRoute_nhop`
7 - Um `deparser` que selecione a ordem que cada campo deve ser inserido no pacote que está saindo do *switch*.
8 - Um instanciação do pacote composta por `parser`, `control` e `deparser`.

Eventualmente esse processamento também requer instâncias de verificação de *checksum* e controles de recálculo. Isso não será necessário para essa tarefa e foram substituídas com controles vazios.

Atividade 3

Source Router - Passo 3: Execução Final

Siga as instruções do Passo 1. Agora as mensagens do *host h*1 deverão ser entregues ao *host h*2.

Verifique o `TTL` do cabeçalho IP. A cada salto o `TTL` será decrementado. A sequência de portas `2 3 2 2 1` força o pacote e fazer uma volta a mais na rede (*loop*), então o `TTL` deverá ser 59 no *host h*2.

Atividade 3

Source Router - Reflexões

- É possível alterar a lógica do código P4 para manipular o encaminhamento IPv4 e o roteamento de origem ao mesmo tempo?
- Como você poderia melhorar seu código para permitir que o primeiro *switch* adicionasse a rota ao pacote, tornando o roteamento de origem transparente para todos os *hosts*?

Atividade 3

Source Router - Solução de Problemas

Existem vários problemas que podem acontecer durante o desenvolvimento da sua implementação:

- O código `task_3.p4` pode falhar durante a compilação. Nesse caso o *script* `run_task_3.sh` apresentará na tela o erro de compilação e será terminado.
- O código `task_3.p4` pode compilar, mas falhar ao carregar as regras do plano de controle descritas no arquivo `command_s<número_do_switch>.txt` que o *script* `run_task.sh` tentará instalar usando o CLI BMv2. Neste caso, o `run_task.sh` irá registrar a saída do CLI no diretório de *logs*. Utilize essa mensagem de erro para consertar a sua implementação.

Atividade 3

Source Router - Solução de Problemas

- O código `task_3.p4` pode compilar, e as regras do plano de controle descritas no arquivo command_s1.txt podem ser carregadas sem problemas, mas ainda assim, o *switch* P4 pode não conseguir processar os pacotes corretamente. O arquivo `/tmp/p4s.s1.log`, contém informações detalhadas descrevendo como o pacote foi processado pelo *switch*. Este *log* pode ajudar a encontrar erros de lógica na sua implementação. Um captura de pacotes também será gerada para as interfaces de entrada e saída de cada um dos *switchs* no formato `<switch-name>-<interfa ce-name>.pcap`. Utilize o comando `tcpdump -r <filename> -xxx` e imprima a saída em hexadecimal (*hexdump*) dos pacotes capturados.

Reiniciando o Mininet

Alguns casos de falha podem ser causados devido a inicialização do ambiente. Nestes casos, reiniciar o Mininet pode ajudar. Utilize o comando `mn -c` para terminar as instâncias que ficaram travadas.