

Tutorial de linguagem P4

Slides: <https://bit.ly/p4d2-2018-spring>

Configuração

do software → Baixe a VM ou copie do pendrive → Importe a VM para o VirtualBox ou VMware Fusion → Inicialize a VM e faça login como usuário "p4" com a senha "p4" → Abra o terminal → cd ~/tutorials; git pull



Metas

- **Aprenda a linguagem P4**

- ÿ Aplicações tradicionais

- Novas aplicações

Aprenda ferramentas de software P4

- ÿ Compilador P4

- ÿ BMv2

- ÿ P4Runtime

- **Aprenda sobre tendências tecnológicas futuras**

- ÿ Palestra de Arvind Krishnamurthy (Washington)
 - ÿ Painel com representantes da ONF, Kaloom e Keysight

- **Networking (o outro tipo)**
- Divirta-se!

Introdução à Programação do Plano de Dados

Noções básicas de linguagem

Quebrar

Ferramentas de software e tempo de execução P4

Almoço

Palestra principal (Arvind Krishnamurthy)

Monitoramento e Depuração

Quebrar

Estruturas de Dados Avançadas

Painel de discussão

Recepção



Instrutores



Stephen Ibañez
Stanford



Brian O'Connor
ONF



Mina Arashloo
Princeton

Obrigado

- Sedef Ozcana (P4.org) •

Rachel Everman (Descalço) • Arvind

Krishnamurthy (Washington) • Painelistas

↳ Uyen Chau (ONF) ↳

Bochra Boughzala (Kaloom) ↳ Ben

Pfaff (VMware) ↳ Chris

Sommers (Keysight) • TAs ↳

Santiago Bautista (ENS/Cornell) ↳ Sean Choi

(Stanford) ↳ Theo Jepsen

(Lugano) ↳ Sarah Tollman

(Stanford)





O que é Programação de Plano de Dados?

- Por que programar o Data Plane?

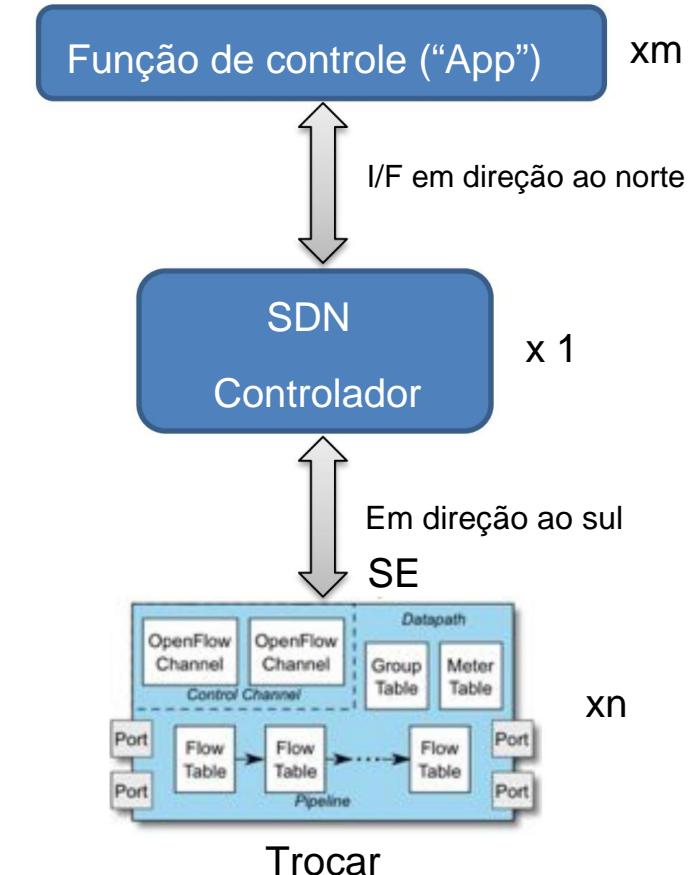
Rede definida por software (SDN)

- **Principais contribuições**

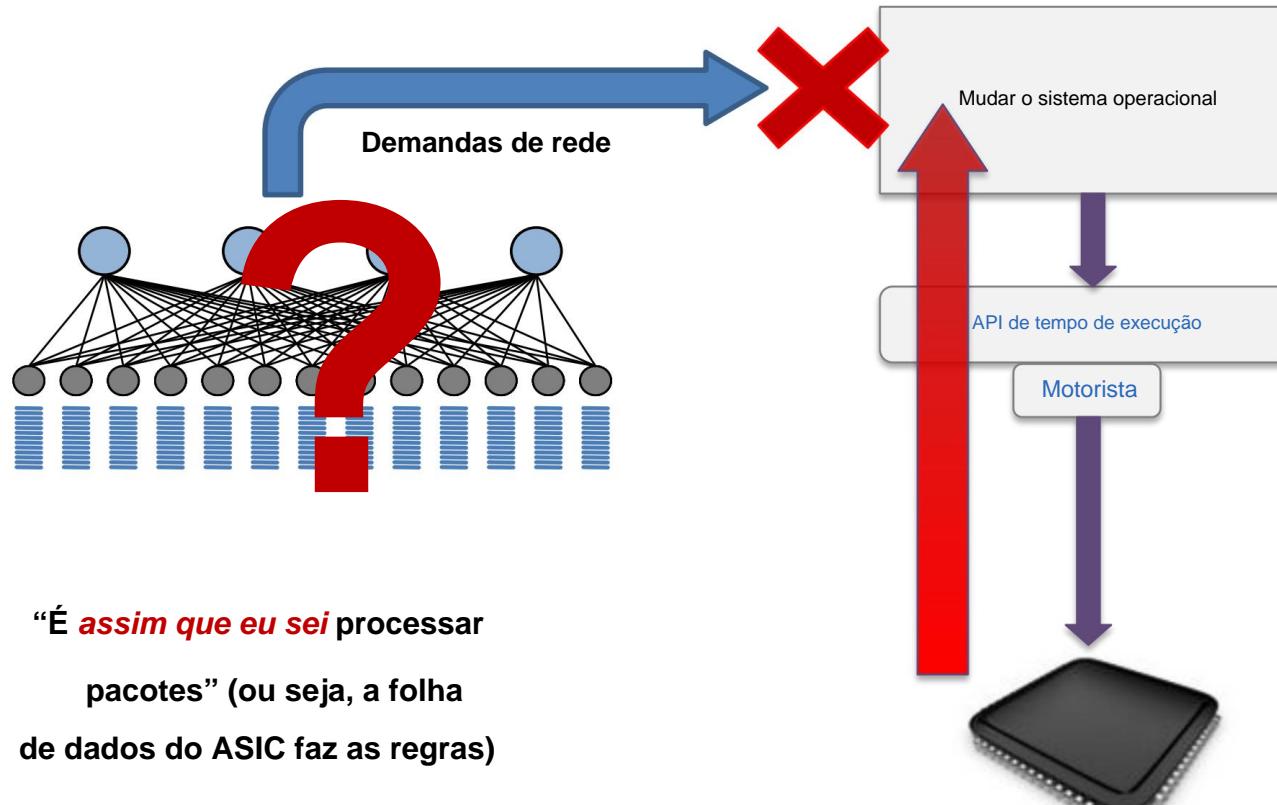
- ÿ OpenFlow = protocolo padronizado para interagir com switch
 - ÿ baixar entradas de tabela de fluxo, estatísticas de consulta, etc.
- ÿ OpenFlow = modelo padronizado
 - ÿ abstração de correspondência/ação
- ÿ Conceito de controle *logicamente centralizado* por meio de uma única entidade (“controlador SDN”)
 - ÿ Simplifica o plano de controle

- **Problemas**

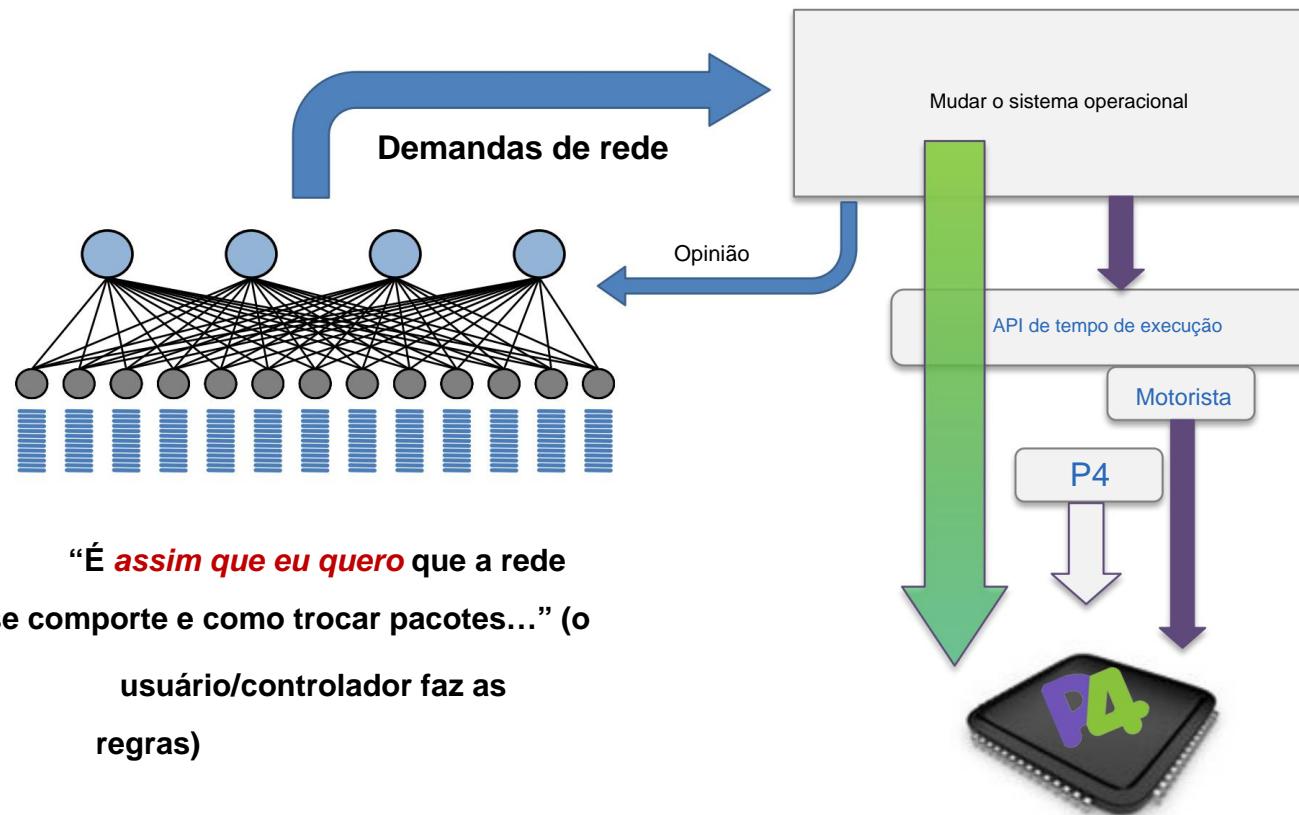
- ÿ A evolução do protocolo do plano de dados requer mudanças nos padrões (12 ÿ 40 campos de correspondência do OpenFlow)
- ÿ Interoperabilidade limitada entre fornecedores (variantes OpenFlow / netconf / JSON / XML)
- ÿ Programabilidade limitada



Status Quo: Design de baixo para cima



Uma abordagem melhor: design de cima para baixo



Benefícios da programabilidade do plano de dados

- **Novos recursos** – Adicionar novos protocolos
- **Reduzir a complexidade** – Remover protocolos não utilizados
- **Uso eficiente de recursos** – uso flexível de tabelas
- **Maior visibilidade** – Novas técnicas de diagnóstico, telemetria, etc.
- **Desenvolvimento de estilo SW** – ciclo de design rápido, inovação rápida, correção de bugs do plano de dados no campo
- **Você mantém suas próprias ideias**

Pense em programação em vez de protocolos...

Dispositivos de rede programáveis

- **PISA: ASICs flexíveis Match+Action**

- ÿ Intel Flexpipe, Cisco Doppler, Cavium (Xpliant), Barefoot Tofino, ... • **NPU**

- ÿ EZchip, Netronome, ... •

CPU

- ÿ Abra Vswitch, eBPF, DPDK, VPP...

- **FPGA**

- ÿ Xilinx, Altera, ...

Esses dispositivos nos permitem dizer a eles como processar pacotes.

O que você pode fazer com P4?

- Balanceador de Carga de Camada 4 – SilkRoad[1]
- Controle de Congestionamento de Baixa Latência – NDP[2]
- Telemetria de Rede em Banda – INT[3]
- Cache e coordenação em Rede – NetCache[4] / NetChain[5]
- Agregação para Aplicações MapReduce [7] ... e muito mais
-

[1] Miao, Rui, et al. "SilkRoad: Tornando o balanceamento de carga de camada 4 com estado rápido e barato usando ASICs de comutação." SIGCOMM, 2017.

[2] Handley, Mark, et al. "Rearquitetura de redes e pilhas de datacenter para baixa latência e alto desempenho." SIGCOMM, 2017.

[3] Kim, Changhoon, et al. "Telemetria de rede em banda via planos de dados programáveis." SIGCOMM. 2015.

[4] Xin Jin et al. "NetCache: balanceamento de armazenamentos de valor-chave com cache rápido na rede". A ser publicado no SOSP 2017. [5] Jin, Xin et al. "NetChain: coordenação de sub-RTT sem escala". NSDI, 2018.

[6] Dang, Huynh Tu, et al. "NetPaxos: Consenso na velocidade da rede." SIGCOMM, 2015.

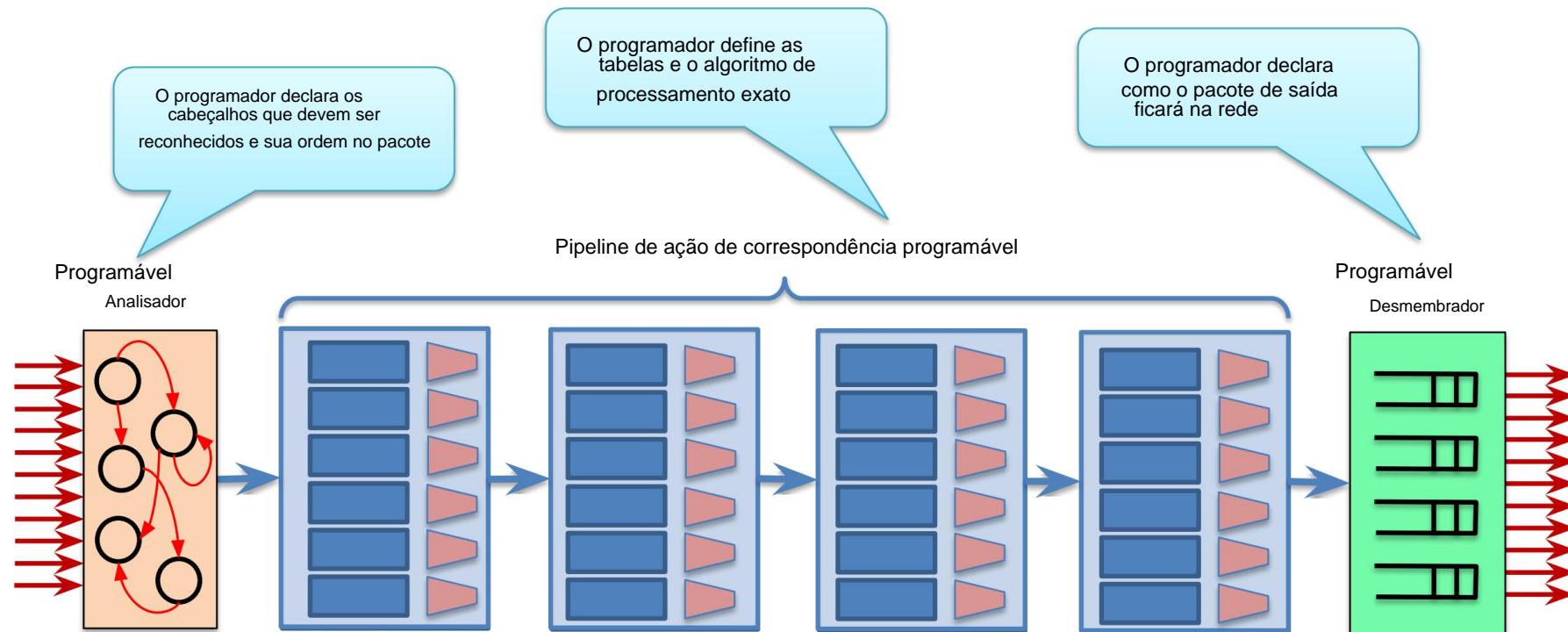
[7] Sapiro, Amedeo, et al. "A computação em rede é uma ideia idiota cuja hora chegou." *Tópicos em alta em redes*. ACM, 2017.

Breve História e Curiosidades

- Maio de 2013: Ideia inicial e o nome “P4” • Julho de 2014: Primeiro artigo (**SIGCOMM CCR**) • Agosto de 2014: Primeiro rascunho da especificação **P414 (v0.9.8)** • Setembro de 2014: Especificação P414 lançada (v1.0.0) • Janeiro de 2015: P414 v1.0.1 • Março de 2015: P414 v1.0.2 • Novembro de 2016: P414 v1.0.3 • Maio de 2017: P414 v1.0.4
- Abr 2016: P416 – primeiros commits
- Dez 2016: Primeiro Rascunho da Especificação P416 • Maio 2017: Especificação P416 lançada

Modelo de plano de dados P4_16

PISA: Arquitetura de Switch Independente de Protocolo

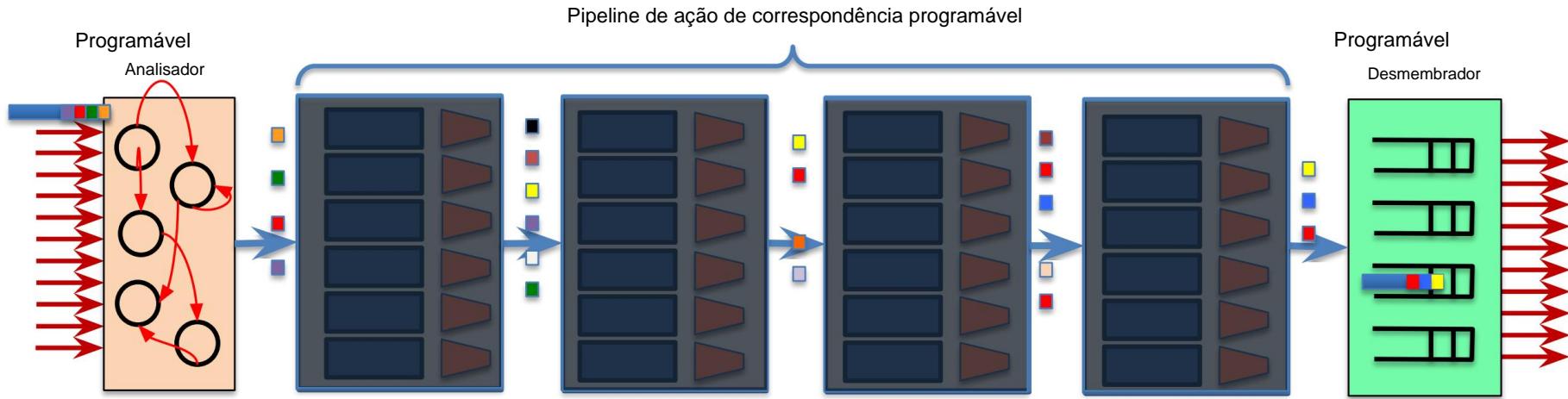


PISA em Ação

- O pacote é analisado em cabeçalhos individuais (representação analisada) •

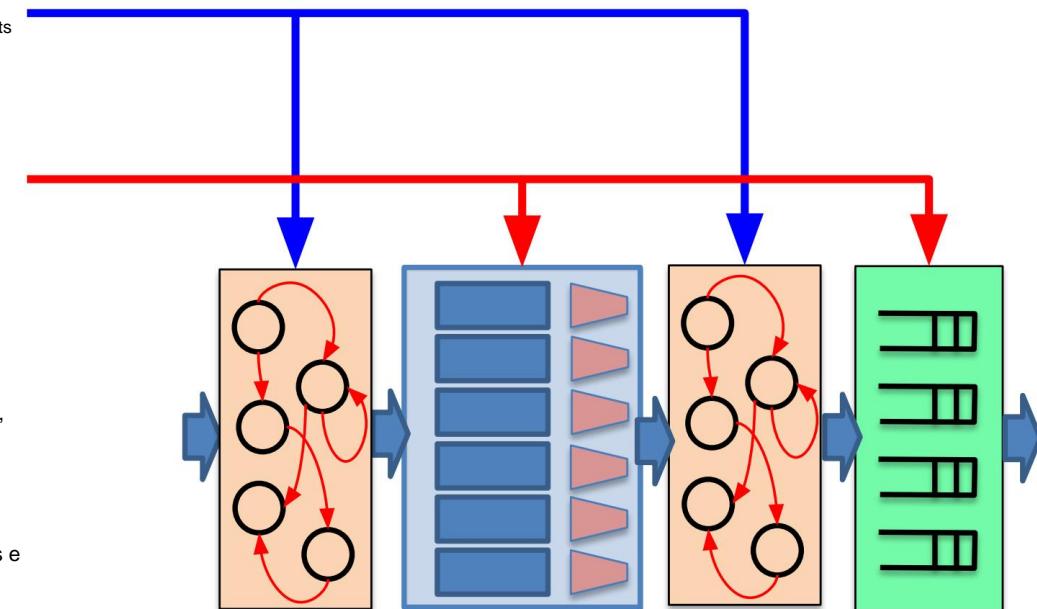
Cabeçalhos e resultados intermediários podem ser usados para correspondência e ações

- Os cabeçalhos podem ser modificados, adicionados ou removidos
- O pacote é desmembrado (serializado)



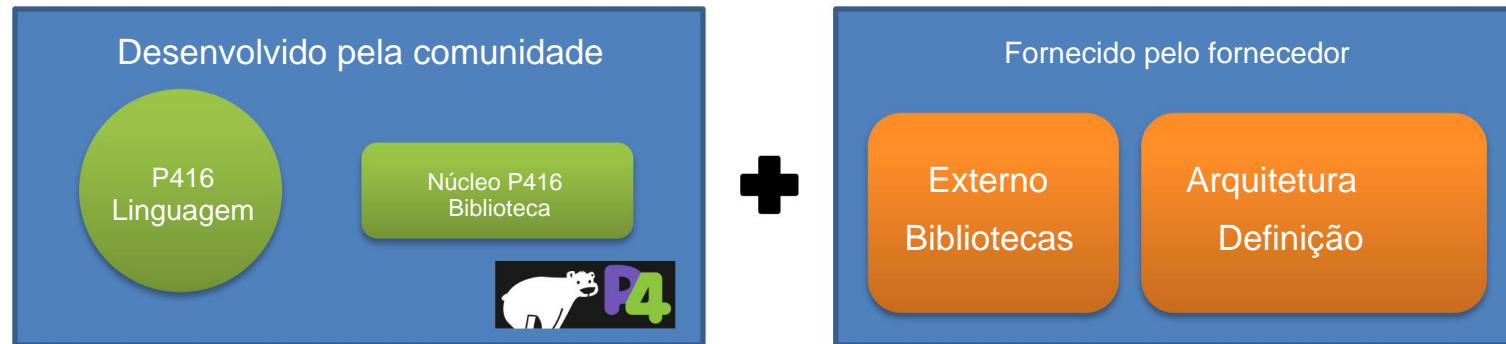
P416 Elementos da Linguagem

Analisadores	Máquina de estados, extração de campo de bits
Controles	Tabelas, ações, instruções de fluxo de controle
Expressões	Operações básicas e operadores
Tipos de dados	Bistings, cabeçalhos, estruturas, matrizes
Arquitetura Descrição	Blocos programáveis e suas interfaces
Bibliotecas externas	Supporte para componentes especializados



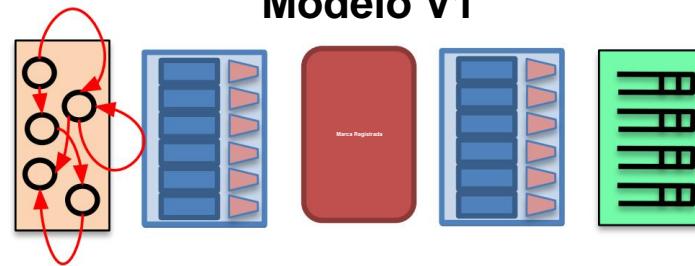
Abordagem P4_16

Prazo	Explicação
Alvo P4	Uma forma de realização de uma implementação de hardware específica
Arquitetura P4	Fornece uma interface para programar um alvo por meio de algum conjunto de componentes programáveis P4, externos, componentes fixos

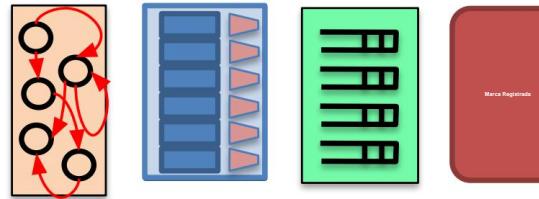


Exemplos de arquiteturas e alvos

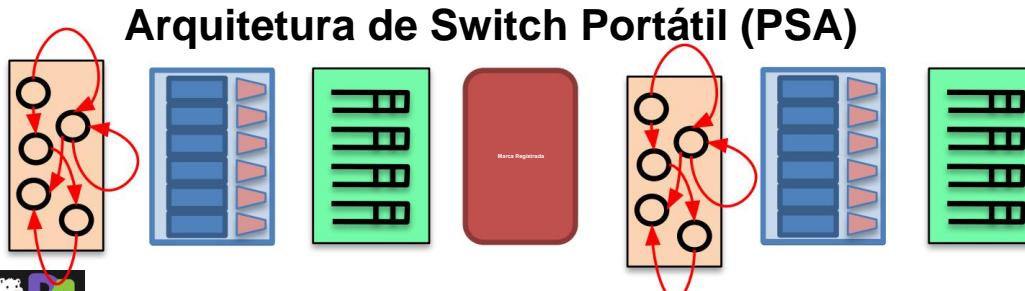
Modelo V1



SimpleSumSwitch

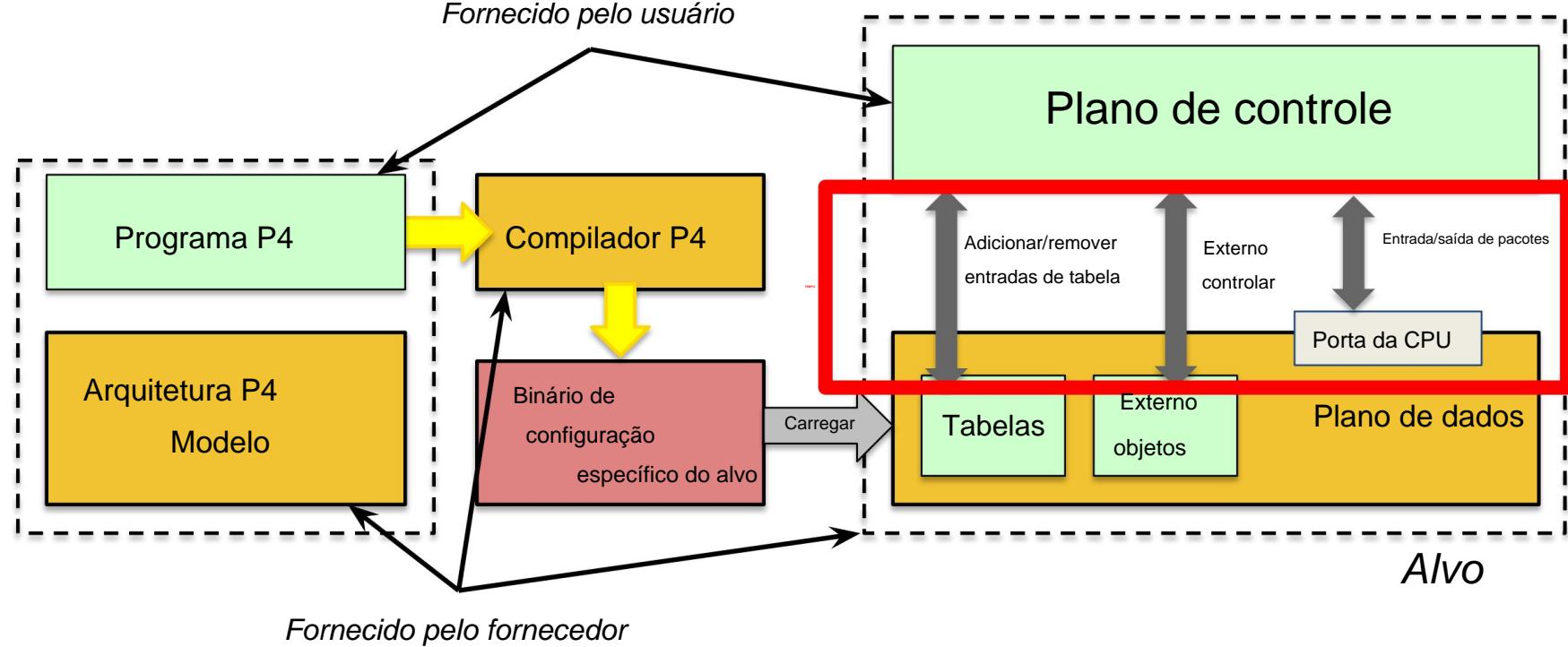


Arquitetura de Switch Portátil (PSA)



Qualquer coisa

Programação de um alvo P4



Laboratório 1: Noções básicas

Antes de começarmos...

- **Instale a imagem da VM (procure o instrutor com pendrives)**
- **Certifique-se de que sua VM esteja atualizada** ↴ \$ cd ~/tutorials && git pull
- **Usaremos**

várias ferramentas de software pré-instaladas na VM

↳ Bmv2: um switch de software P4

↳ p4c: o compilador P4 de referência

↳ Mininet: um ambiente de emulação de rede leve

- **Cada diretório contém alguns scripts**

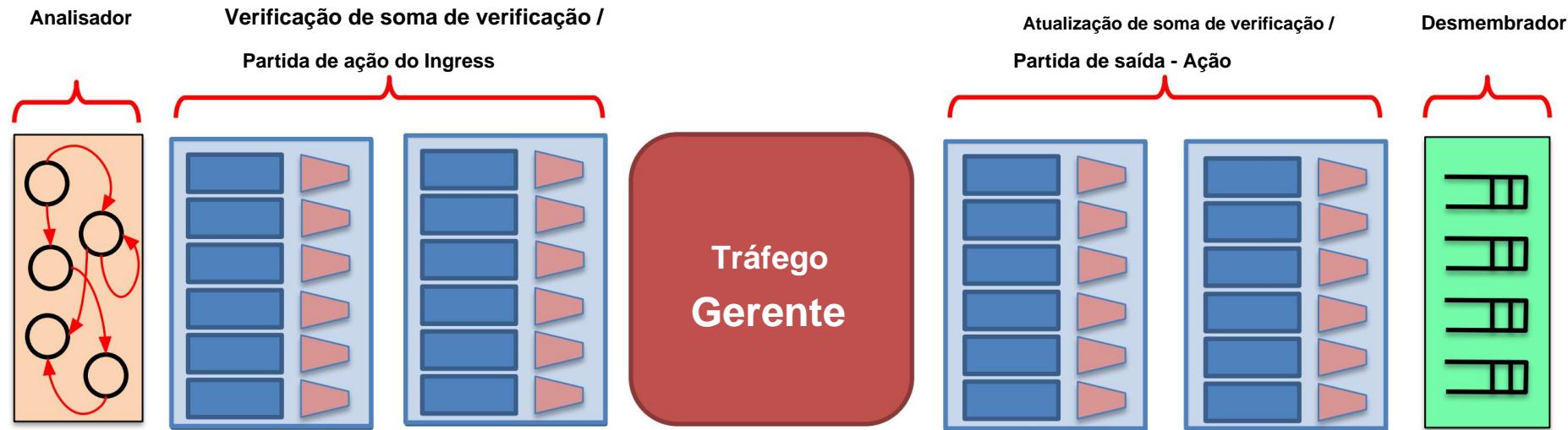
↳ \$ make: compila o programa P4, executa no Bmv2 no Mininet, preenche tabelas ↳ *.py: envia e recebe pacotes de teste no Mininet

- **Exercícios**

↳ Cada exemplo vem com uma implementação incompleta; sua tarefa é finalizá-la! ↳ Procure por "TODOs" (ou dê uma olhada no código P4 em solution/, se necessário)

Arquitetura V1Model

- Implementado sobre o alvo simple_switch do Bmv2



Metadados Padrão do Modelo V1

```
struct standard_metadata_t { bit<9>
    porta_de_entrada; bit<9>
    especificação_de_saída;
    bit<9> porta_de_saída;
    bit<32>
    especificação_de_clone;
    bit<32>
    tipo_de_instância; bit<1> descartar;
    bit<16> porta_de_recirculação;
    bit<32> comprimento_do_pacote;
    bit<32>
    carimbo_de_hora_enq; bit<19>
    profundidade_de_q;
    bit<32> delta_de_tempo_de_q; bit<19>
    profundidade_de_q; bit<48>
    carimbo_de_hora_global_de_entrada; bit<32> lista_de_campos_lf; bit<16> grp_mcast; bit<1> sinalizador_de_reenvio; bit<16> id_de_s
}
```

- **ingress_port** - a porta na qual o pacote chegou
- **egress_spec** - a porta para a qual o pacote deve ser enviado para
- **egress_port** - a porta que o pacote será enviado de (ler somente no pipeline de saída)

Modelo de Programa P416 (Modelo V1)

```
#incluir <core.p4>
#incluir <v1model.p4>
/* CABEÇALHOS */
metadados de estrutura { ... }
cabeçalhos de estrutura {
    ethernet_t ethernet;
    ipv4_t ipv4;
}

```

```
/* ANALISADOR */
analisador MyParser(packet_in pacote,
    cabeçalhos de saída hdr,
    metadados de entrada e saída,
    inout standard_metadata_t smeta) {
    ...
}
```

```
/* VERIFICAÇÃO DE CHECKSUM */
controle MyVerifyChecksum(em cabeçalhos hdr, em
    metadados meta) {
    ...
}
```

```
/* PROCESSAMENTO DE ENTRADA */
controle MyIngress(inout cabeçalhos hdr, inout
    metadados meta, inout
    standard_metadata_t std_meta) {
    ...
}
```

```
/* PROCESSAMENTO DE SAÍDA */
controle MyEgress(inout cabeçalhos hdr, inout
    metadados meta, inout
    standard_metadata_t std_meta) {
    ...
}
```

```
/* ATUALIZAÇÃO DE CHECKSUM */
controle MyComputeChecksum(inout cabeçalhos hdr, inout
    metadados meta) {
    ...
}
```

```
/* DESPARSER */
controle MyDeparser(inout cabeçalhos hdr, inout
    metadados meta) {
    ...
}
```

```
/* TROCAR */
Interruitor V1(
    MeuParser(),
    MyVerifyChecksum(),
    MeuIngresso(),
    MinhaSaída(),
    MyComputeChecksum(),
    MeuDeparser()
) principal;
```

P416 Olá Mundo (Modelo V1)

```
#include <core.p4>
#include <v1model.p4>
metadados de estrutura
{} cabeçalhos de estrutura {}

analisador MyParser(packet_in pacote,
    cabeçalhos de saída
    hdr, metadados de entrada
    meta, metadados padrão de entrada_t metadados_padrão) {
    estado iniciar { transição aceitar; }
}

controle MyVerifyChecksum(inout cabeçalhos hdr, inout metadados meta)
{ aplicar { } }

controle MyIngress(inout cabeçalhos hdr, inout
    metadados meta, inout
    standard_metadata_t standard_metadata) { aplicar { se
        (standard_metadata.ingress_port == 1)
            { standard_metadata.egress_spec = 2; }
        senão se (standard_metadata.ingress_port == 2)
            { standard_metadata.egress_spec = 1;
        }
    }
}
```

```
controle MyEgress(inout cabeçalhos hdr, inout
    metadados meta, inout
    metadados_padrão_t metadados_padrão) { aplicar { } }

controle MyComputeChecksum(inout cabeçalhos hdr, inout metadados meta)
{ aplicar
    { }
}

controle MyDeparser(packet_out pacote, em cabeçalhos hdr) { aplicar { } }

Interruptor V1
    MeuParser(),
    MyVerifyChecksum(),
    MeuIngresso(),
    MinhaSaída(),
    MyComputeChecksum(),
    MyDeparser() )
principal;
```

P416 Olá Mundo (Modelo V1)

```
#incluir <core.p4>
#incluir <v1model.p4>
metadados de estrutura {}
cabeçalhos de estrutura {}

analisador MyParser(pacote_de_entrada_pacote, saída_cabeçalhos_hdr ,
    metadados de entrada e saída,
    inout metadados_padrão_t metadados_padrão) {
    estado iniciar { transição aceitar; }
}

controle MyIngress ( cabeçalhos de entrada hdr, metadados de entrada meta,
    inout metadados_padrão_t metadados_padrão) {
    ação set_egress_spec(bit<9> porta) {
        standard_metadata.egress_spec = porta;
    }
    tabela para frente {
        chave = { standard_metadata.ingress_port: exato; }
        ações = {
            definir_espec_de_saída;
            Sem ação;
        }
        tamanho = 1024;
        default_action = SemAção();
    }
    aplicar { encaminhar.aplicar(); }
}
```

```
controle MyEgress( cabeçalhos de entrada e saída hdr,
    metadados de entrada e saída,
    inout metadados_padrão_t metadados_padrão) {
    aplicar {}
}

controle MyVerifyChecksum(inout cabeçalhos hdr, inout metadados meta)
{ aplicar {} }

controle MyComputeChecksum(inout cabeçalhos hdr, inout metadados meta)
{ aplicar {} }

controle MyDeparser(packet_out pacote, em cabeçalhos hdr) {
    aplicar {}
}
```

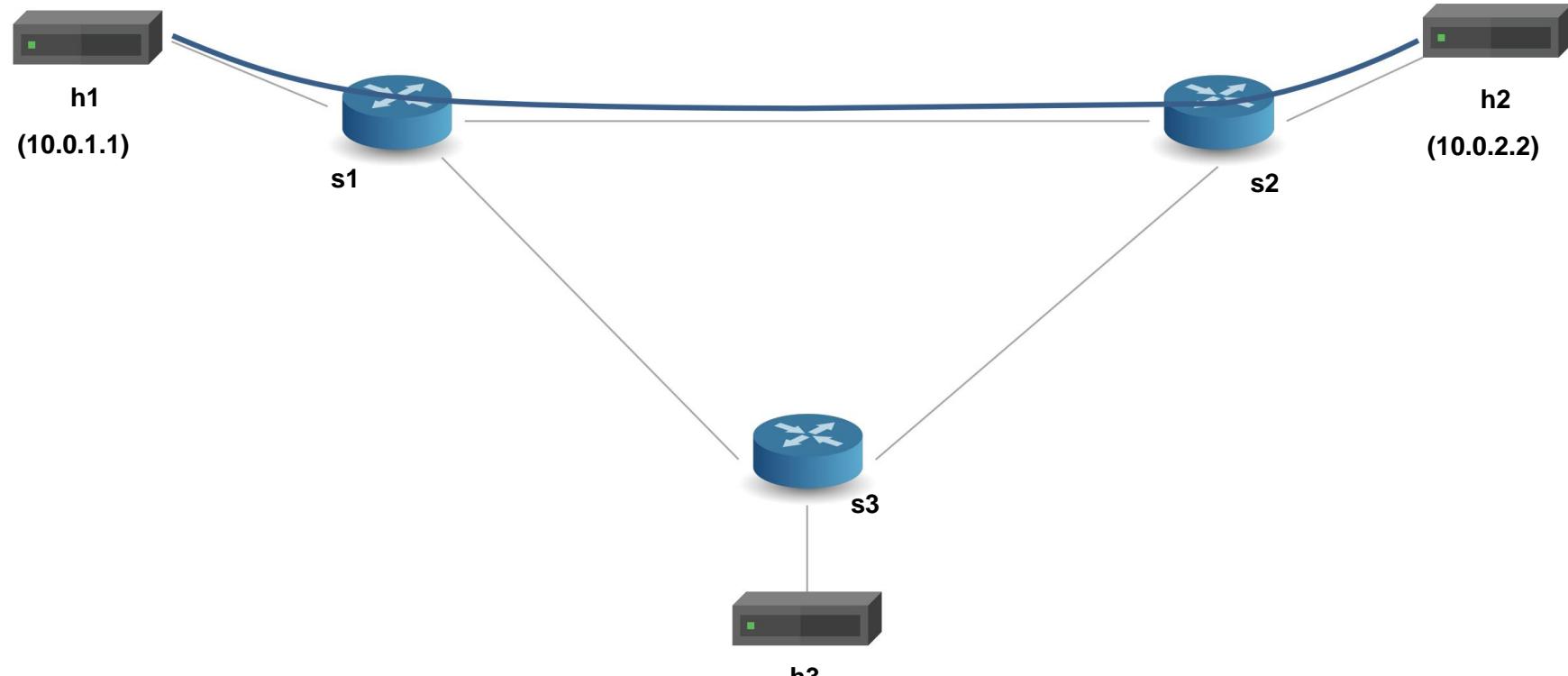
V1Switch(MeuParser(), MeuVerificarChecksum(), MeuIngresso(),
 MyEgress(), MyComputeChecksum(), MyDeparser()) principal;

Chave	ID da ação	Dados de ação
1	ID do conjunto_egress_spec	2
2	ID do conjunto_egress_spec	1

Exemplo de execução: encaminhamento básico

- Usaremos um aplicativo simples como exemplo de execução — um roteador básico — para ilustrar os principais recursos do P416
- Funcionalidade básica do roteador:
 - ÿ Analisar cabeçalhos Ethernet e IPv4 do pacote
 - ÿ Encontrar destino na tabela de roteamento IPv4
 - ÿ Atualizar endereços MAC de origem/destino
 - ÿ Diminuir campo de tempo de vida (TTL)
 - ÿ Definir porta de saída
 - ÿ Desmembrar cabeçalhos de volta para um pacote
- Escrevemos um código inicial para você (basic.p4) e implementamos um plano de controle estático

Encaminhamento básico: topologia



Tipos P416 (Tipos Básico e de Cabeçalho)

```

typedef bit<48> macAddr_t; typedef
bit<32> ip4Addr_t; cabeçalho ethernet_t
{ macAddr_t r; dstAdd
    macAddr_t srcAddr; bit<16>
    etherType;
}

cabeçalho ipv4_t { bit<4>
    versão; ihl;
    um pouco<4>
    pouco<8>      diffserv;
    bit<16> totalLen; bit<16>
    identificação; bit<3> sinalizadores;
    bit<13> fragOffset; bit<8>
    ttl; protocolo; bit<16> hdrChecksum;
    ip4Addr_t srcAddr;
    pouco<8>      ip4Addr_t
    dstAddr;
}

```

Tipos básicos

- **bit<n>**: inteiro sem sinal (bitstring) de tamanho n • **bit** é o mesmo que **bit<1>** • **int<n>**: inteiro com sinal de tamanho n ($>=2$) • **varbit<n>**: bitstring de comprimento variável

Tipos de cabeçalho: Coleção ordenada de membros • Pode conter **bit<n>**, **int<n>** e **varbit<n>** • Alinhado por bytes • Pode ser válido ou inválido • Fornece várias operações para testar e definir a validade do bit: **isValid()**, **setValid()** e **setInvalid()**

Typedef: Nome alternativo para um tipo

Tipos P416 (Outros Tipos)

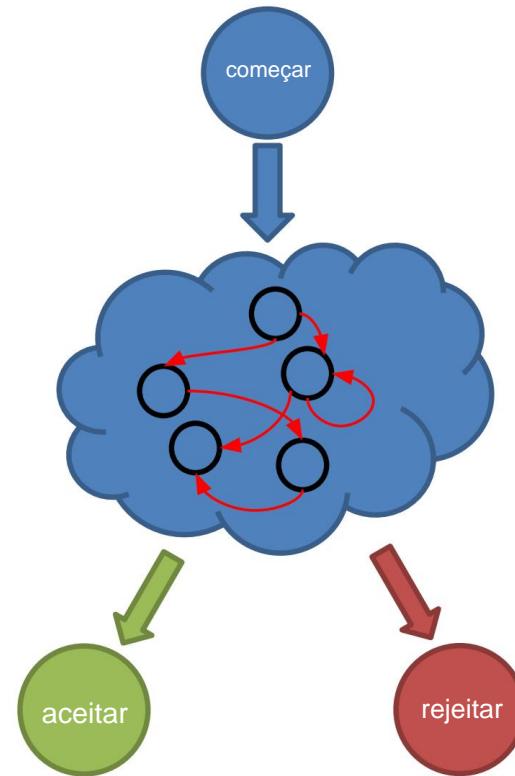
```
/* Arquitetura */ struct  
standard_metadata_t { bit<9>  
    porta_de_entrada; bit<9>  
    especificação_de_saída;  
    bit<9> porta_de_saída;  
    bit<32>  
    especificação_de_clone;  
    bit<32>  
    tipo_de_instância; bit<1> descarte;  
    bit<16> porta_de_recirculação; bit<32> comprimento_do_pacote;  
    ...  
}  
  
/* Programa do usuário  
 */ struct metadata {  
    ...  
}  
  
} struct cabeçalhos  
{ ethernet_t ethernet; ipv4_t  
    ipv4;  
}
```

Outros tipos úteis

- **Estrutura:** Coleção não ordenada de membros (sem restrições de alinhamento)
- **Pilha de cabeçalhos:** matriz de cabeçalhos
- **União de cabeçalhos:** um dos vários cabeçalhos

Analisadores P416

- Os analisadores são funções que mapeiam pacotes em cabeçalhos e metadados, escritos em um estilo de máquina de estado
- Cada analisador tem três estados predefinidos
 - ÿ começar
 - ÿ aceitar
 - ÿ rejeitar
- Outros estados podem ser definidos pelo programador
- Em cada estado, execute zero ou mais instruções e, em seguida, faça a transição para outro estado (loops são aceitáveis)



Analisadores (Modelo V1)

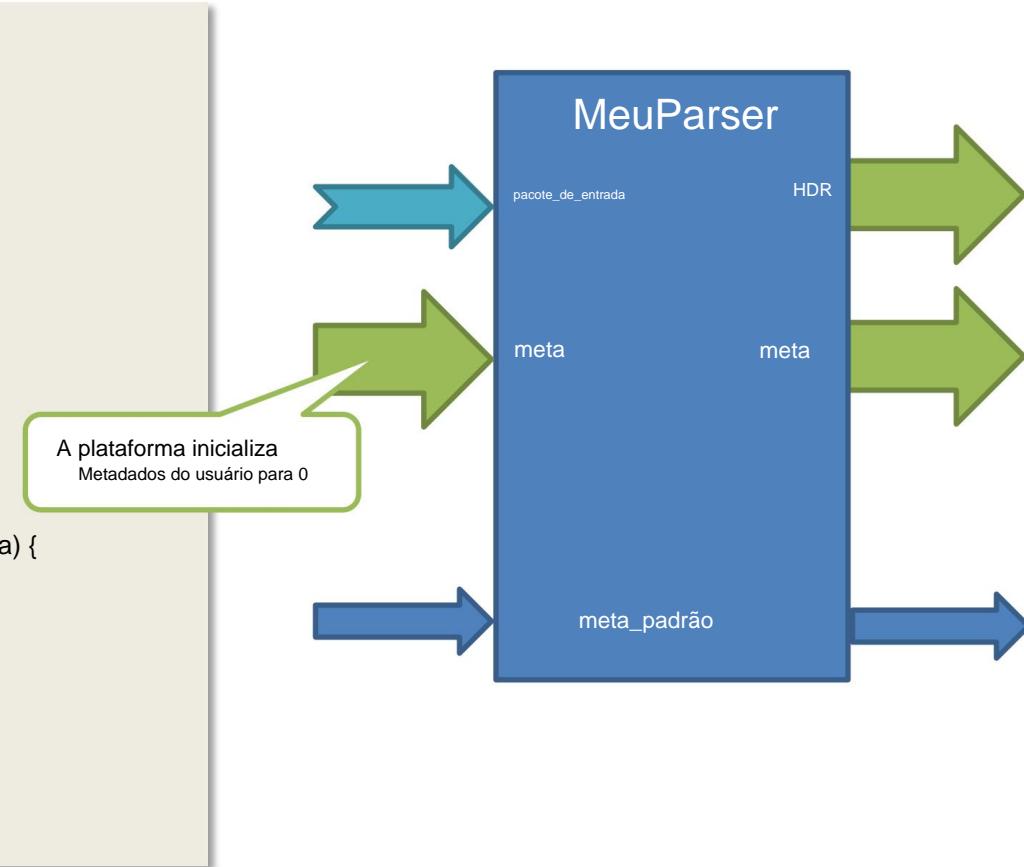
```

/* De core.p4 */ extern
packet_in { void
    extract<T>(out T hdr); void
    extract<T>(out T variableSizeHeader, in bit<32>
        variableFieldSizeInBits);
    T lookahead<T>();
    void avançar(em bit<32> tamanholnBits);
    bit<32> comprimento();
}
/* Programa do usuário
*/
parser MyParser(packet_in packet,
    cabeçalhos de saída
    hdr, metadados de entrada
    meta, metadados padrão de entrada_t std_meta) {

estado iniciar
{ pacote.extract(hdr.ethernet);
transição aceitar;
}

}

```



Selecione a declaração

```
início do estado {  
    transição parse_ethernet;  
}  
  
estado parse_ethernet {  
    pacote.extraír(hdr.ethernet);  
    transição selecionar(hdr.ethernet.etherType) {  
        0x800: analisar_ipv4;  
        padrão: aceitar;  
    }  
}
```

P416 tem uma instrução select que pode ser usada para ramificar em um analisador

Semelhante às instruções de caso em C ou Java, mas sem “comportamento de queda” — ou seja, as instruções break não são necessárias

Em analisadores, muitas vezes é necessário ramificar com base em alguns dos bits recém-analisados

Por exemplo, etherType determina o formato do restante do pacote

Os padrões de correspondência podem ser literais ou cálculos simples, como máscaras

Intervalo de codificação

Controles P416

- Semelhante às funções C (sem loops)
- Pode declarar variáveis, criar tabelas, instanciar externos, etc.
- Funcionalidade especificada pelo código na instrução apply
- Representa todos os tipos de processamento que podem ser expressos como DAG:
 - ÿ Pipelines de ação de correspondência
 - ÿ Desanalisadores
 - ÿ Formas adicionais de processamento de pacotes (atualização de somas de verificação)
- As interfaces com outros blocos são governadas por tipos especificados pelo usuário e pela arquitetura (normalmente cabeçalhos e metadados)

Exemplo: Refletor (Modelo V1)

```
controle MyIngress( cabeçalhos de entrada e saída hdr,
                    metadados de entrada e saída,
                    entrada/saída metadados_padrão_t meta_padrão) {
    /* Região de declarações */
    bit<48> tmp;

    aplicar {
        /* Fluxo de controle */
        tmp = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
        hdr.ethernet.srcAddr = tmp;
        std_meta.egress_spec = std_meta.ingress_port;
    }
}
```

Comportamento desejado:

- Trocar endereços MAC de origem e destino
- Devolver o pacote na porta física em que ele chegou ao switch

Exemplo: Ações Simples

```

controle MyIngress(inout cabeçalhos hdr, inout
                    metadados meta, inout
                    standard_metadata_t std_meta) {

    ação swap_mac(inout bit<48> src, inout
                  bit<48> dst) { bit<48> tmp
        = src; src = dst; dst =
        tmp;

    }

    aplicar
    { swap_mac(hdr.ethernet.srcAddr,
               hdr.ethernet.dstAddr);
      std_meta.egress_spec = std_meta.ingress_port;
    }
}

```

- Muito semelhante às funções C •
- Podem ser declaradas dentro de um controle ou globalmente • Os parâmetros têm tipo e direção
- As variáveis podem ser instanciadas dentro
- Muitas operações aritméticas e Operações lógicas são suportadas \ddot{y} +, -, \ddot{y} ~, $\&$, $|$, \wedge , $*$
 $>>$, $<<$ \ddot{y} ==, !=, >, \geq , <, \leq \ddot{y} Sem divisão/ módulo
- Operações não padronizadas:
 \ddot{y} Bit-slicing: [m:l] (funciona também como l-value) \ddot{y} Concatenação de bits: ++

Tabelas P416

- A unidade fundamental de um pipeline de ação de correspondência

Especifica quais dados corresponder e o tipo de correspondência

ÿ Especifica uma lista de ações possíveis

Especifica opcionalmente uma série de **propriedades** de tabela

ÿ

Tamanho ÿ Ação

padrão ÿ Entradas

estáticas ÿ etc.

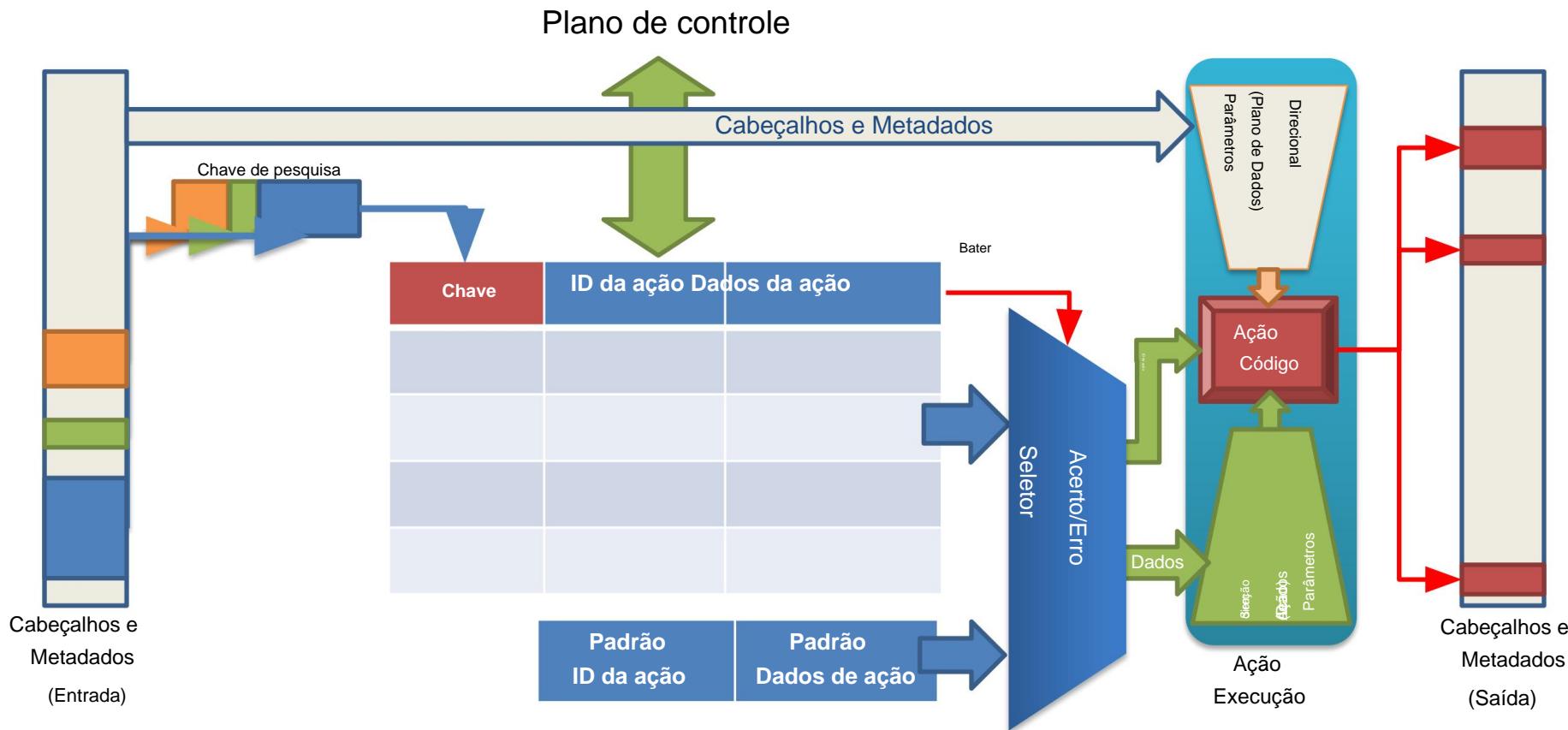
- Cada tabela contém uma ou mais entradas

- (regras) • Uma entrada contém:

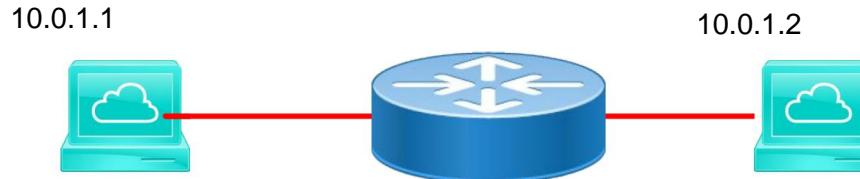
ÿ Uma chave específica para

corresponder ÿ Uma **única** ação que é executada quando um pacote corresponde à
entrada ÿ Dados de ação (possivelmente vazios)

Tabelas: Processamento de Ação de Correspondência



Exemplo: Tabela IPv4_LPM



Chave	Ação	Dados de ação
10.0.1.1/32 ipv4_forward dstAddr=00:00:00:00:01:01		porta=1
10.0.1.2/32 queda *	Sem ação	

- **Programa Data Plane (P4)**

 - ÿ Define o formato da tabela

 - ÿ Campos-chave

 - ÿ Ações

 - ÿ Dados de ação

 - ÿ Executa a pesquisa

 - ÿ Executa a ação escolhida

- **Plano de controle (pilha IP, Protocolos de roteamento)**

 - ÿ Preenche entradas de tabela com informações específicas

 - ÿ Com base na configuração

 - ÿ Baseado na descoberta automática

 - ÿ Com base em cálculos de protocolo

Tabla IPv4_LPM

```
tabla ipv4_lpm
{ chave
  = { hdr.ipv4.dstAddr: lpm;

} ações =
{ ipv4_forward;
descartar;
Sem ação;

} tamanho =
1024; ação_padrão = NoAction();
}
```

Tipos de correspondência

```
/* core.p4 */
match_kind
{ exato,
ternário, lpm
}

/* v1model.p4 */
tipo_de_correspondência {
intervalo,
seletor
}

/* Alguma outra arquitetura */ match_kind {
expressão
regular, difusa
}
```

- O tipo **match_kind** é especial em P4

- A biblioteca padrão (core.p4) **define três tipos de correspondência padrão** :
 - Correspondência exata
 - Correspondência ternária
 - Correspondência LPM

- A arquitetura (v1model.p4) **define dois tipos de correspondência adicionais:**

- alcance
 - seletor

- Outras arquiteturas podem definir (e fornecer implementação para) tipos de correspondência adicionais

Definindo ações para encaminhamento L3

```
/* núcleo.p4 */
ação NoAction() { }
```

```
/* básico.p4 */
ação drop() {
    marcar_para_soltar();
}
```

```
/* básico.p4 */
ação ipv4_forward(macAddr_t dstAddr,
                  bit<9> porta) {
    ...
}
```

- As ações podem ter dois tipos diferentes de parâmetros

ÿ Direcional (do Plano de Dados)
 ÿ Sem direção (do controle Avião)

- Ações que são chamadas diretamente:
- ÿ Use apenas parâmetros direcionais
- Ações utilizadas em tabelas:

ÿ Normalmente use parâmetros sem direção
 ÿ Às vezes pode usar parâmetros direcionais também



Aplicando tabelas em controles

```
controle MyIngress(inout cabeçalhos hdr, inout
                    metadados meta, inout
                    standard_metadata_t standard_metadata) { tabela ipv4_lpm {
    ...
}

} aplicar {
    ...
    ipv4_lpm.aplicar();
    ...
}
}
```

P416 Desmembramento

```

/* De core.p4 */ extern
packet_out { void
    emit<T>(in T hdr); }

/* Programa do usuário
*/ control DeparserImpl(packet_out packet, in
                           headers hdr) {
    aplicar {
        ...
        pacote.emitir(hdr.ethernet);
        ...
    }
}

```

- Reúne os cabeçalhos em um pacote bem formado
- Expresso como uma função de controle
 - ÿNão há necessidade de outra construção!
- **packet_out** extern é definido em **core.p4**:
 - emit(hdr): serializa o cabeçalho se for válido
- **Vantagens:**
 - Torna a análise sintática explícita...
 - ...mas se desvincula da análise sintática

Intervalo de codificação

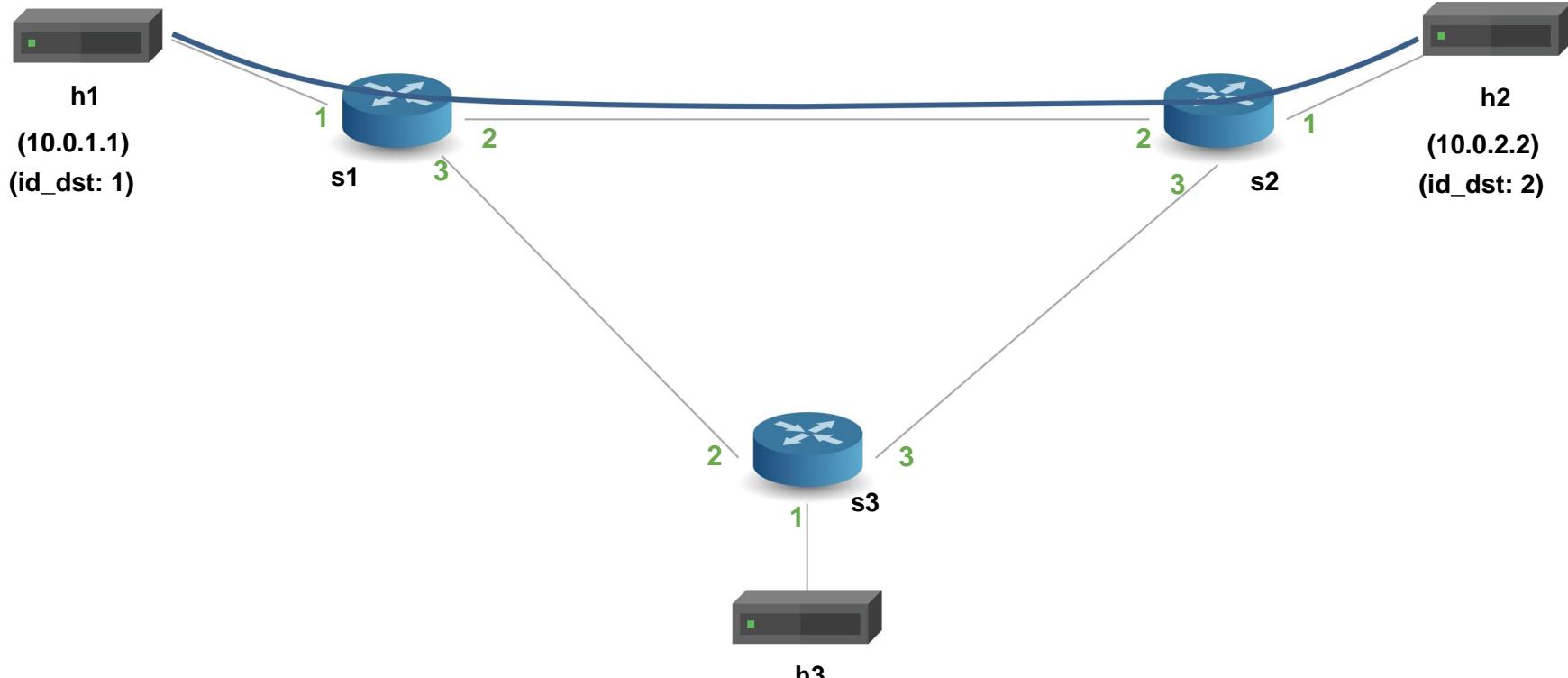
Tunelamento básico

- Adicione suporte para tunelamento básico ao roteador IP básico
- Defina um novo tipo de cabeçalho (myTunnel) para encapsular o pacote IP
- o cabeçalho myTunnel inclui:
 - proto_id: tipo de pacote que está sendo encapsulado
 - dst_id: ID do host de destino
- Modifique o switch para executar o roteamento usando o cabeçalho myTunnel

Lista de tarefas básicas de tunelamento

- Defina o tipo de cabeçalho `myTunnel_t` e adicione à estrutura de cabeçalhos
- Atualizar analisador
- Definir a ação `myTunnel_forward`
- Definir a tabela `myTunnel_exact`
- Atualizar a lógica do aplicativo de tabela na instrução `apply` do MyIngress
- Atualizar desparser
- Adicionando regras de encaminhamento

Encaminhamento básico: topologia



Intervalo de codificação

Perguntas frequentes

- **Posso aplicar uma tabela várias vezes no meu Programa P4?**

ÿ Não (exceto via reenvio/recirculação)

- **Posso modificar entradas de tabela do meu programa P4?**

ÿ Não (exceto para contadores diretos)

- **O que acontece ao atingir o estado de rejeição do analisador?**

ÿ Dependente da arquitetura

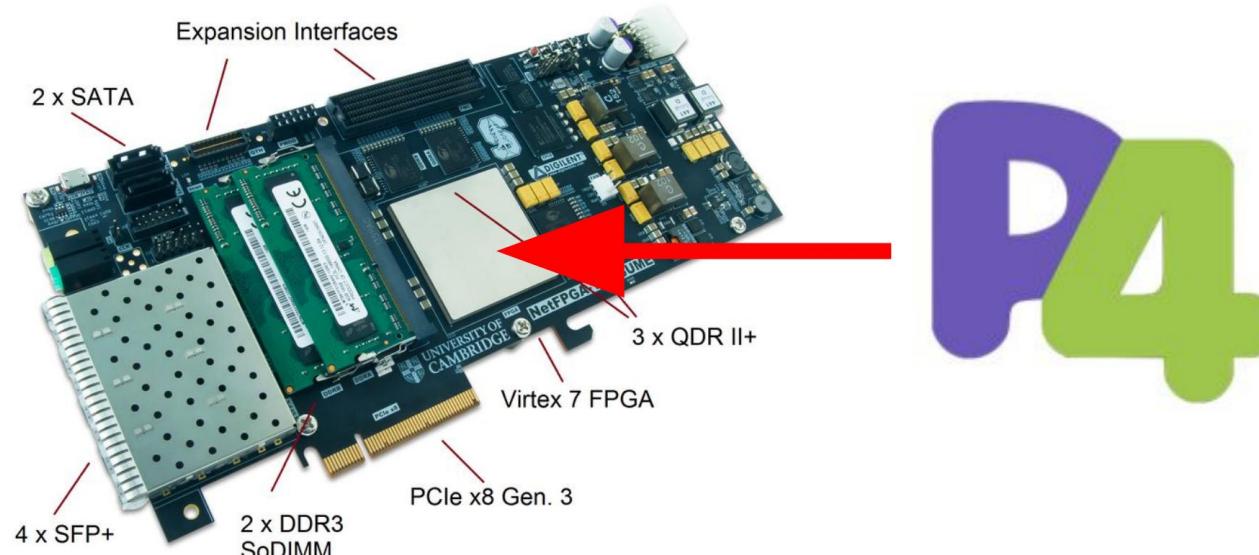
- **Quanto do pacote posso analisar?**

ÿ Dependente da arquitetura

Fim!

P4yNetFPGA

- Prototipar e avaliar programas P4 em hardware real!
- Interfaces de rede 4x10G
- Preço especial para usuários acadêmicos :)
- <https://github.com/NetFPGA/P4-NetFPGA-public/wiki>



Depuração

```
controlar MyIngress(...) {  
    tabela de  
  
    depuração { chave = { std_meta.egress_spec : exato;  
    }  
    ações = { }  
}  
    aplicar {  
        ...  
        depurar.aplicar();  
    }  
}
```

- O Bmv2 mantém registros que rastreiam como os pacotes são processados em detalhes
 - /tmp/p4s.s1.log
 - /tmp/p4s.s2.log
 - /tmp/p4s.s3.log
- Pode adicionar manualmente informações ao logs usando uma tabela de depuração fictícia que lê cabeçalhos e metadados de interesse
 - [15:16:48.145] [bmv2] [D] [tópico 4090]
[96.0] [cxt 0]
Procurando a chave:
* std_meta.egress_spec : 2

Perguntas e Respostas



<https://bit.ly/join-p4-lang-slack>

Junte-se ao canal #d2-2018-spring

Perguntas para TAs }



Pigeonhole®

<https://pigeonhole.at>

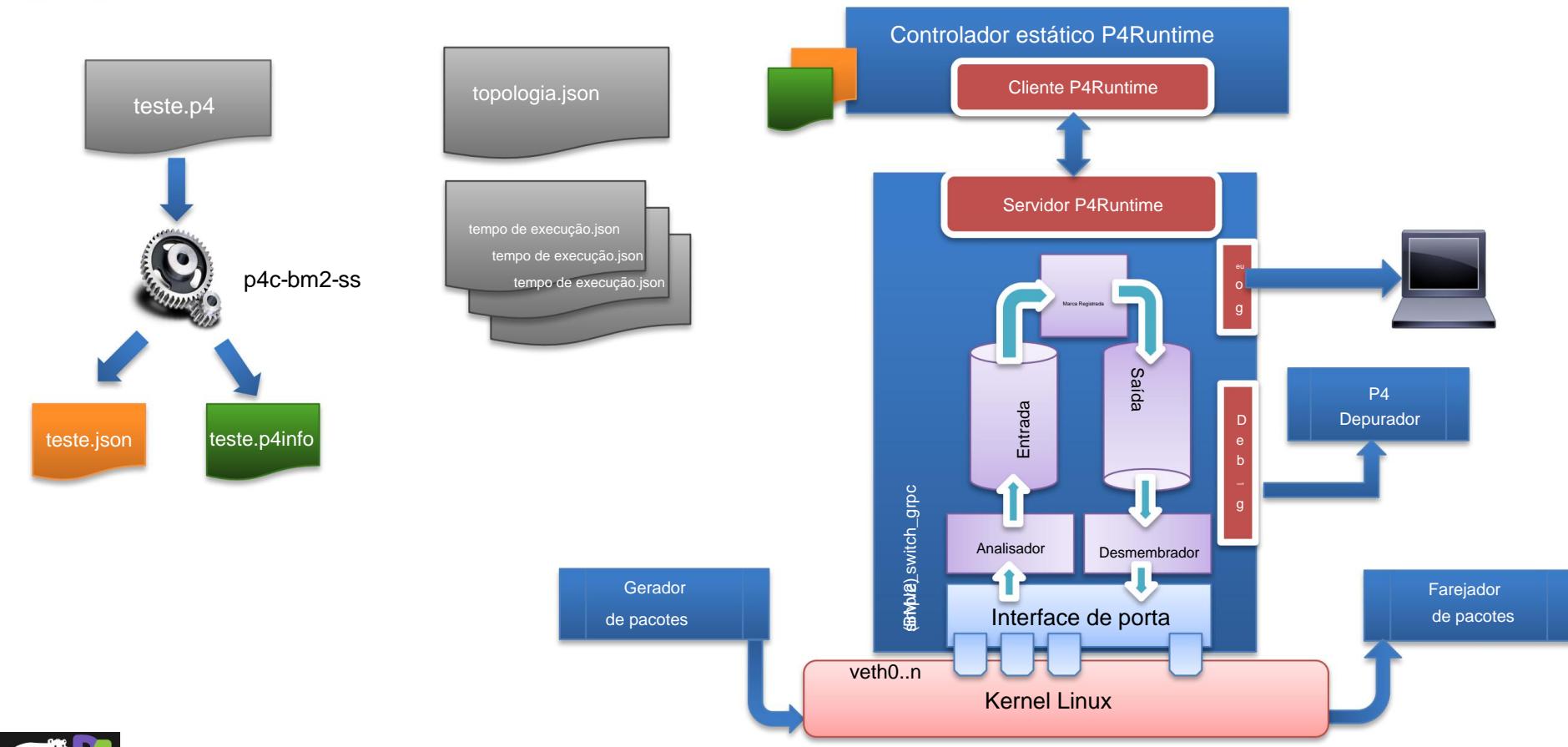
Código do evento: P4D2

Perguntas para o Painel }

Laboratório 2: P4Runtime

Ferramentas de software P4

Makefile: nos bastidores



Makefile: nos bastidores (em pseudocódigo)

```
P4C_ARGS = --p4runtime-file $(nome base $@).p4info --p4runtime-format texto  
RUN_SCRIPT = ../../utils/run_exercise.py
```

TOPO = topology.json

dirs:

mkdir -p build pcaps logs **build: para cada**

programa P4, gerar arquivo BMv2 json

p4c-bm2-ss --p4v 16 \$(P4C_ARGS) -o \$@ \$< *[alvo padrão]*

depois sudo python **executar: construir,**

\$(RUN_SCRIPT) -t \$(TOPO) **parar: sudo mn -c limpar:**

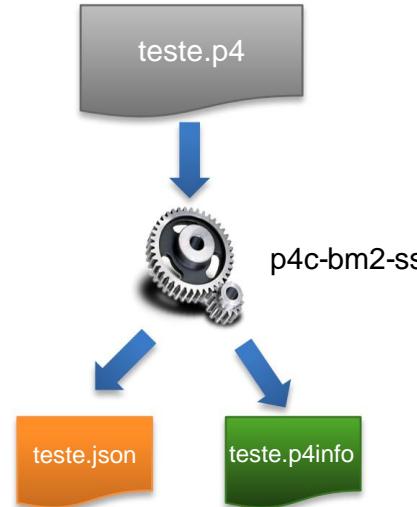
parar, depois rm -f

*.pcap rm -rf construir

pcaps logs



Etapa 1: Compilação do programa P4 [fase de construção]



```
$ p4c-bm2-ss --p4v 16 \ -o test.json  
  \ --p4runtime-file  
  test.p4info \ --p4runtime-format text \ test.p4
```

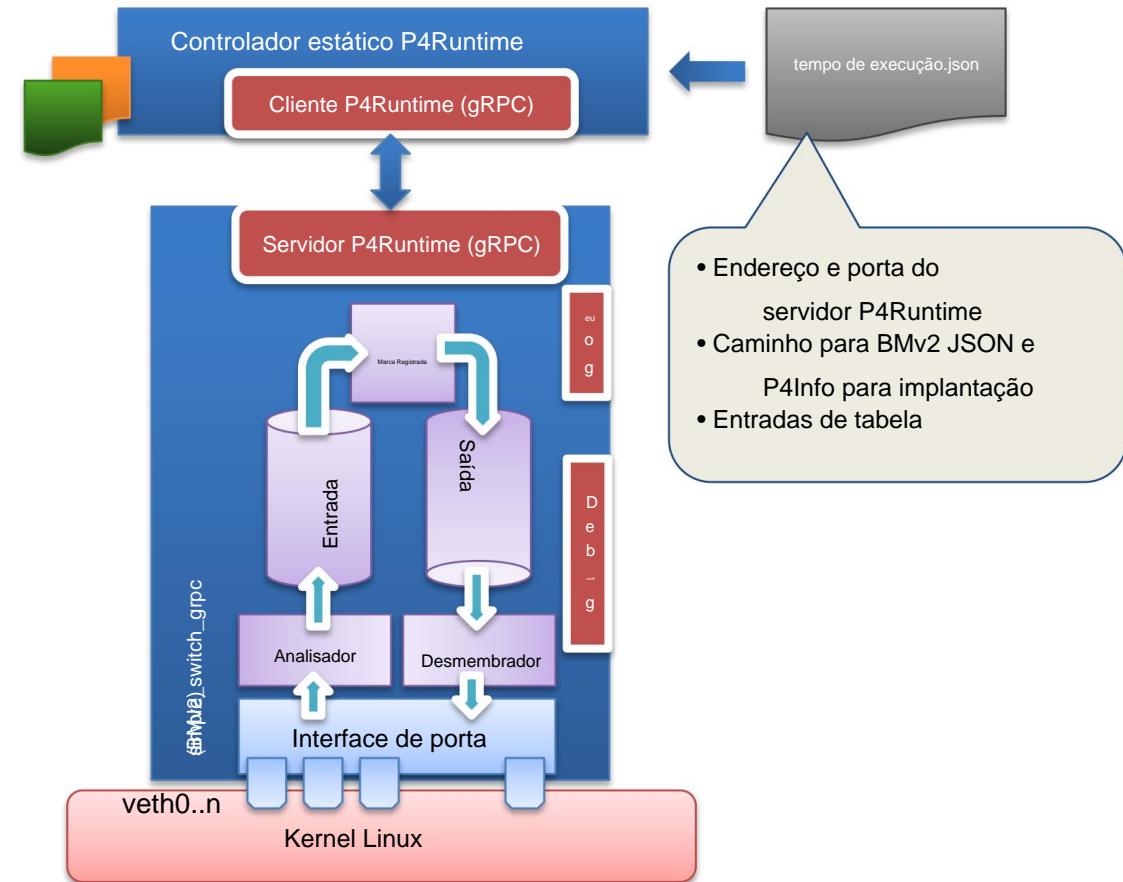
Etapa 2: run_exercise.py [fase de execução]

a. Criar rede baseada
em topology.json

b. Iniciar
simple_switch_grpc
instância para cada
switch

c. Use P4Runtime para
envie o programa P4
(P4Info e BMv2 JSON)

d. Adicione as regras estáticas
definido em runtime.json

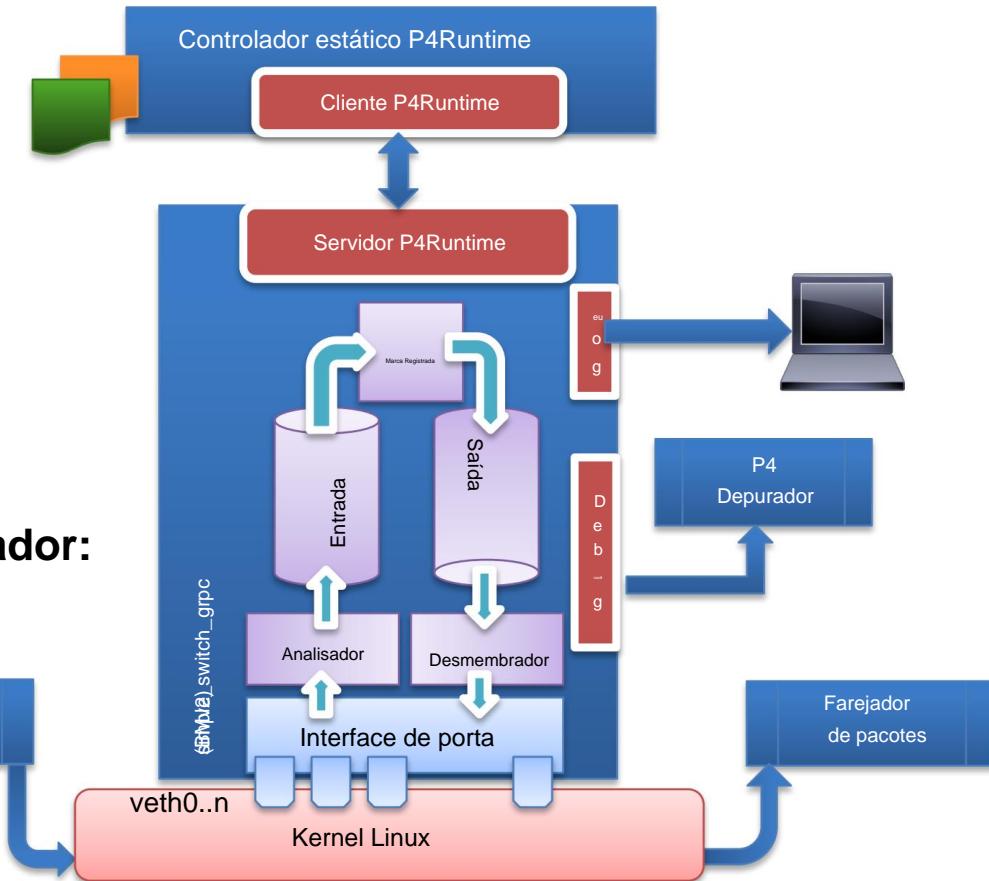


Etapa 3: execute o gerador de tráfego e o farejador

Em alguns exercícios, isso é send.py e receive.py

Em outros, usamos o padrão Programas Linux, como ping

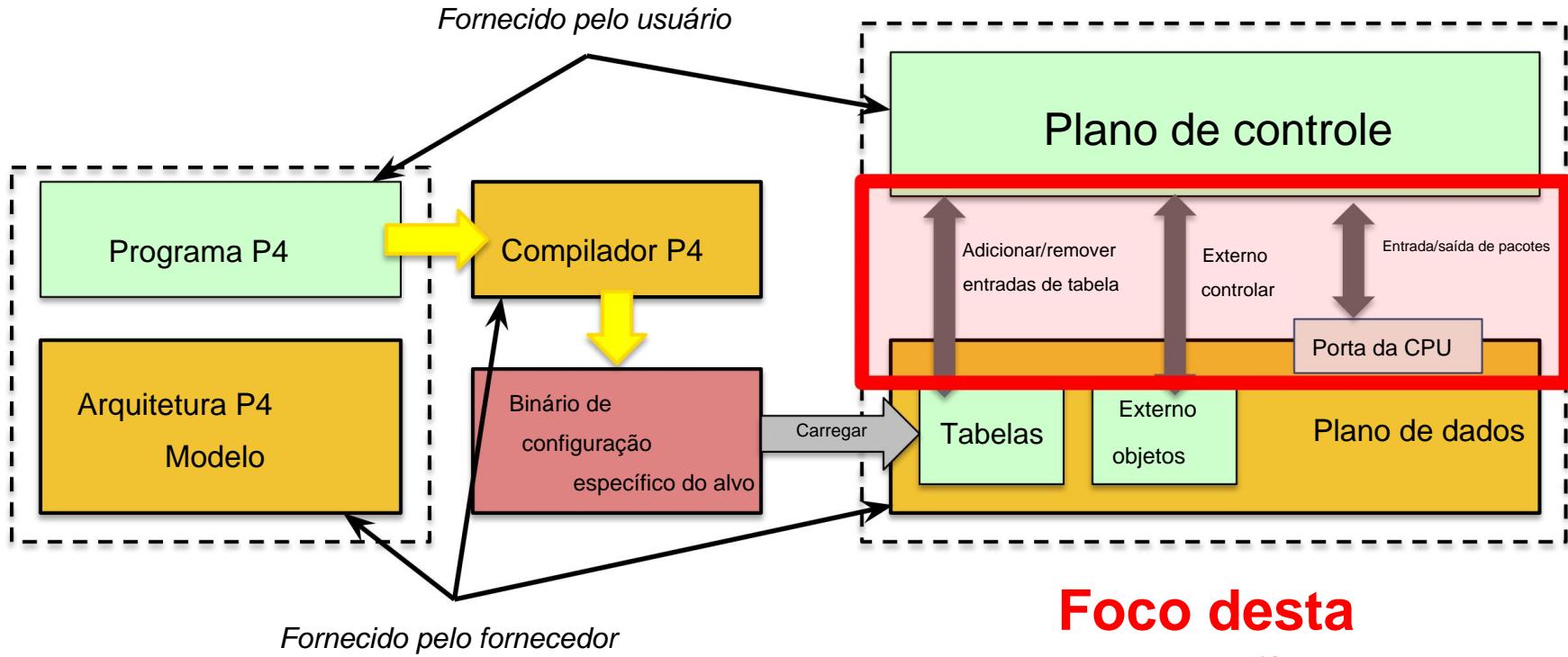
No exercício p4runtime, também executamos nosso próprio controlador: mycontroller.py em vez do estático



P4Tempo de execução

- Visão geral da API
- Fluxo de trabalho • Exercício - Tunelamento • Por que P4Runtime?

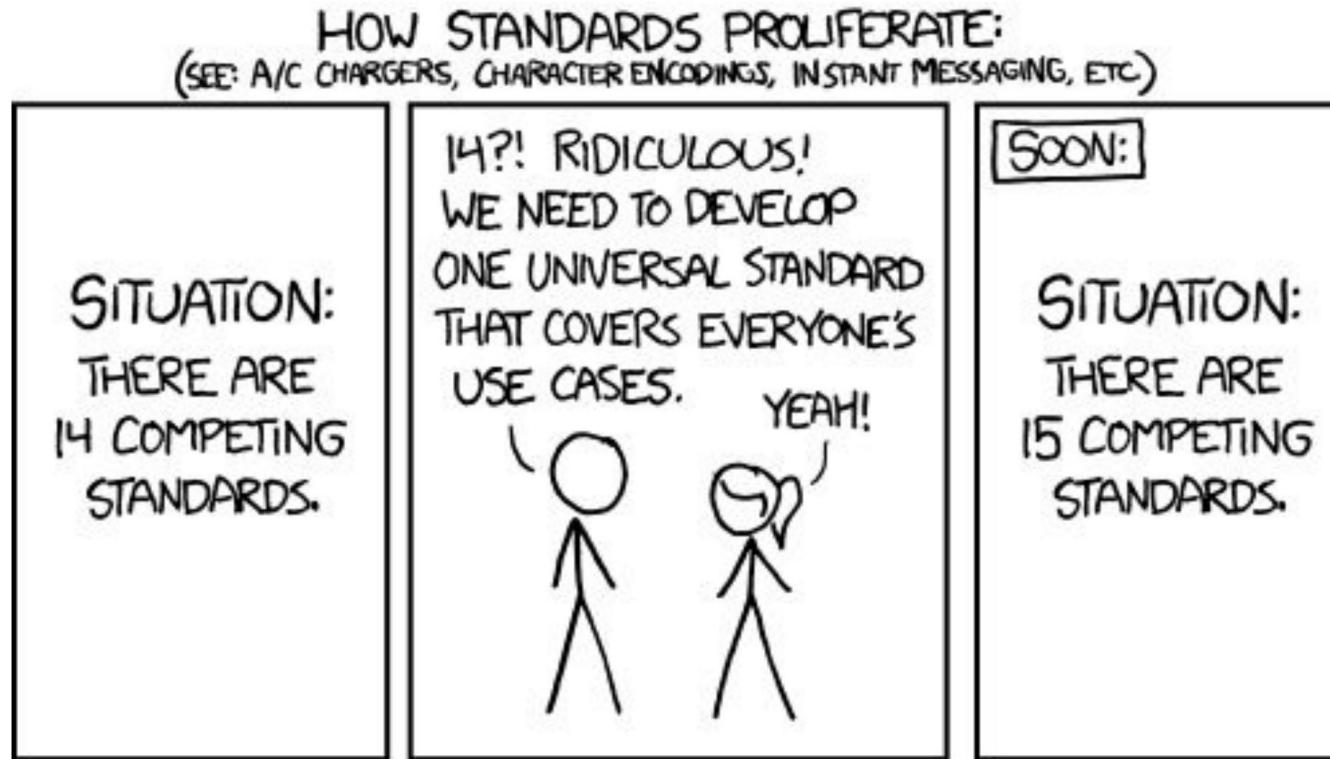
Controle de tempo de execução de planos de dados P4



Abordagens existentes para controle de tempo de execução

- APIs de tempo de execução geradas automaticamente pelo compilador P4
 - ÿ Dependente do programa — difícil provisionar um novo programa P4 sem reiniciar o plano de controle!
- CLI do BMv2
 - ÿ Independente do programa, mas específico do alvo -- plano de controle não portátil!
- OpenFlow
 - ÿ Independente do alvo, mas dependente do protocolo — cabeçalhos de protocolo e ações incorporados na especificação!
- Interface de abstração de switch OCP (SAI)
 - ÿ Independente do alvo, mas dependente do protocolo

Por que precisamos de outra API de controle de plano de dados?



Propriedades de uma API de controle de tempo de execução

API	Protocolo independente de alvo	
Compilador P4 gerado automaticamente	ÿ	ÿ
CLI do BMv2	ÿ	ÿ
Fluxo aberto	ÿ	ÿ
SAI	ÿ	ÿ
P4Tempo de execução	ÿ	ÿ

O que é P4Runtime?

- Estrutura para controle de tempo de execução de alvos P4 → API de código aberto + implementação de servidor → <https://github.com/p4lang/PI>

Contribuição inicial do Google e da Barefoot

→

- Trabalho em andamento pelo grupo de trabalho da API p4.org

→ Rascunho da versão 1.0 disponível

- Definição de API baseada em Protobuf → p4runtime.proto →

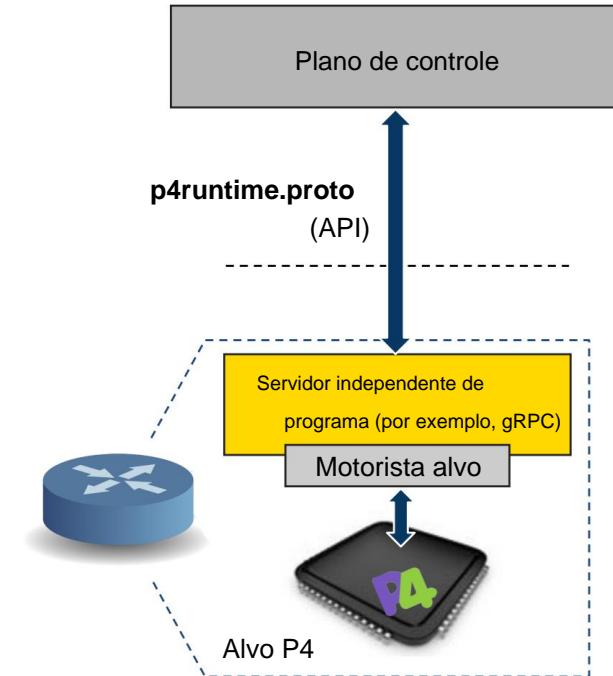
transporte gRPC

- Independente do programa P4 → A

API não muda com o programa P4

- Permite reconfiguração de campo →

Capacidade de enviar um novo programa P4 sem recompilar a pilha de software dos switches de destino



Noções básicas sobre buffers de protocolo

- Linguagem para descrever dados para serialização de forma estruturada
- Formato de fio binário comum
- Neutro em termos de linguagem → Geradores de código para:
Action Script, C, C++, C#, Clojure, Lisp, D, Dart, Erlang, Go, Haskell, Java, Javascript, Lua, Objective C, OCaml, Perl, PHP, Python, Ruby, Rust, Scala, Swift, Visual Basic
- Neutro em relação à plataforma
- Extensível e compatível com versões anteriores
- Fortemente tipado

```
sintaxe = "proto3";  
  
mensagem Pessoa {  
    string nome = 1; int32 id  
    = 2; string email = 3;  
  
    enumeração PhoneType {  
        MÓVEL = 0;  
        CASA = 1;  
        TRABALHO = 2;  
    }  
  
    mensagem PhoneNumber { string  
        número = 1;  
        Tipo de telefone = 2;  
    }  
  
    PhoneNumber repetido telefone = 4;  
}
```

Noções básicas de gRPC

- Use Buffers de Protocolo para definir API de serviço e mensagens

- Gerar automaticamente stubs nativos em:

- C# • Dart •

- Go •

- Java •

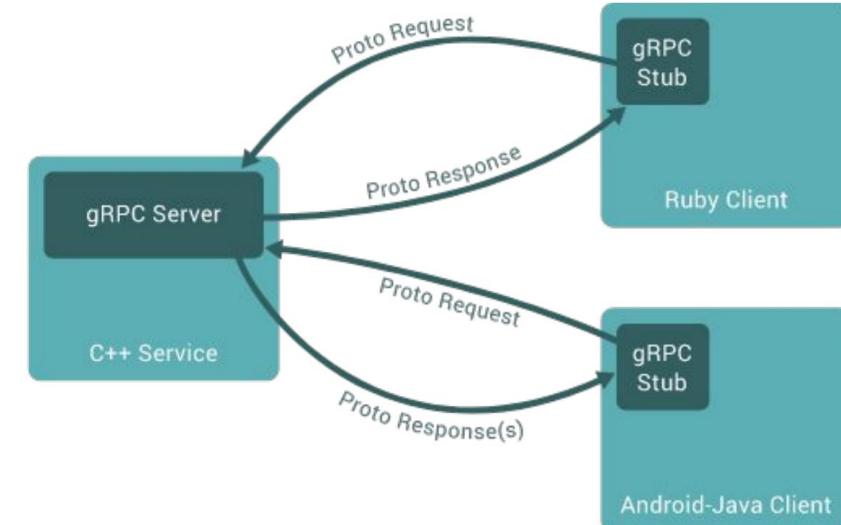
- Node.js

- PHP •

- Python • Ruby

- Transporte sobre HTTP/2.0 e TLS •

Implementação eficiente de conexão TCP única que suporta streaming bidirecional



Exemplo de serviço gRPC

```
// A definição do serviço de saudação.  
service Greeter { //  
    Envia uma saudação rpc  
    SayHello (HelloRequest) retorna (HelloReply) {}  
}  
  
// A mensagem de solicitação contendo o nome do usuário.  
message HelloRequest  
{ string name = 1;  
}  
  
// A mensagem de resposta contendo a mensagem de  
saudação HelloReply  
{ string message = 1;  
}
```

Mais detalhes aqui: <https://grpc.io/docs/guides/>

Serviço P4Runtime

Permite que uma entidade local ou remota carregue o pipeline/programa, envie/receba pacotes e leia e grave entradas da tabela de encaminhamento, contadores e outros recursos do chip.

serviço P4Runtime

```
{ rpc Write(WriteRequest) retorna (WriteResponse) {} rpc  
Read(ReadRequest) retorna (stream ReadResponse) {} rpc  
SetForwardingPipelineConfig(SetForwardingPipelineConfigRequest) retorna  
    (SetForwardingPipelineConfigResponse) {} rpc  
GetForwardingPipelineConfig(GetForwardingPipelineConfigRequest) retorna  
    (GetForwardingPipelineConfigResponse) {}  
rpc StreamChannel(stream StreamMessageRequest)  
    retorna (stream StreamMessageResponse) {}  
}
```

Serviço P4Runtime

Definição de Protobuf:

<https://github.com/p4lang/PI/blob/master/proto/p4/v1/p4runtime.proto>

Especificação de serviço:

o rascunho de trabalho da versão 1.0 já está disponível <https://p4.org/p4-spec/docs/P4Runtime-v1.0.0.pdf>

Solicitação de gravação P4Runtime

```
mensagem WriteRequest { uint64
    device_id = 1; uint64 role_id =
    2;
    Uint128 election_id = 3; atualizações
    repetidas = 4; }
```

```
mensagem Atualizar
{ enum Tipo {
    NÃO ESPECIFICADO = 0;
    INSERIR = 1;
    MODIFICAR = 2;
    EXCLUIR = 3;
}
Tipo tipo = 1;
Entidade entidade = 2; }
```

```
mensagem Entidade
{ uma das entidades {
    Entrada externa entrada_externa = 1;
    TableEntry entrada_de_tabela = 2;
    ActionProfileMember
        ação_perfil_membro = 3;
    Grupo de Perfil de Ação
        grupo_de_perfil_de_ação = 4;
    MeterEntry entrada_do_medidor = 5;
    Entrada_do_Medidor_direto entrada_do_medidor_direto = 6;
    CounterEntry contador_entrada = 7;
    Entrada_do_Contador_direta entrada_do_contador_direta = 8;
    PacketReplicationEngineEntry
        entrada_do_mecanismo_de_replicação_de_pacote = 9;
    ValueSetEntry valor_conjunto_entrada = 10;
    RegisterEntry registro_entrada = 11;
} }
```

Entrada de tabela P4Runtime

Trechos simplificados de p4runtime.proto:

mensagem TableEntry

```
{ uint32 table_id;
correspondência FieldMatch repetida;
Ação ação ; int32
prioridade;
...
}
```

mensagem Ação {

```
uint32 action_id;
mensagem Parâmetro {
    uint32 param_id; bytes
    valor;
}
Param params repetidos ;
```

mensagem FieldMatch

```
{ uint32 field_id;
```

mensagem Exato

```
{ bytes valor;
```

mensagem Ternária

```
{ bytes valor;
```

```
bytes máscara;
```

```
}
```

```
...
```

um dos campos do tipo de correspondência {

Exatamente exato;

Ternário ternário ;

...

}

Definição completa do

protobuf: <https://github.com/p4lang/P1/blob/master/proto/p4/p4runtime.proto>

Para adicionar uma entrada de tabela, o plano de controle precisa saber:

- IDs de entidades P4

Tabelas, correspondências de campos, ações, parâmetros, etc.

- Correspondências de campo para

- a tabela específica

Tipo de correspondência, largura de bits, etc.

- Parâmetros para a ação específica

- Outros atributos do programa P4

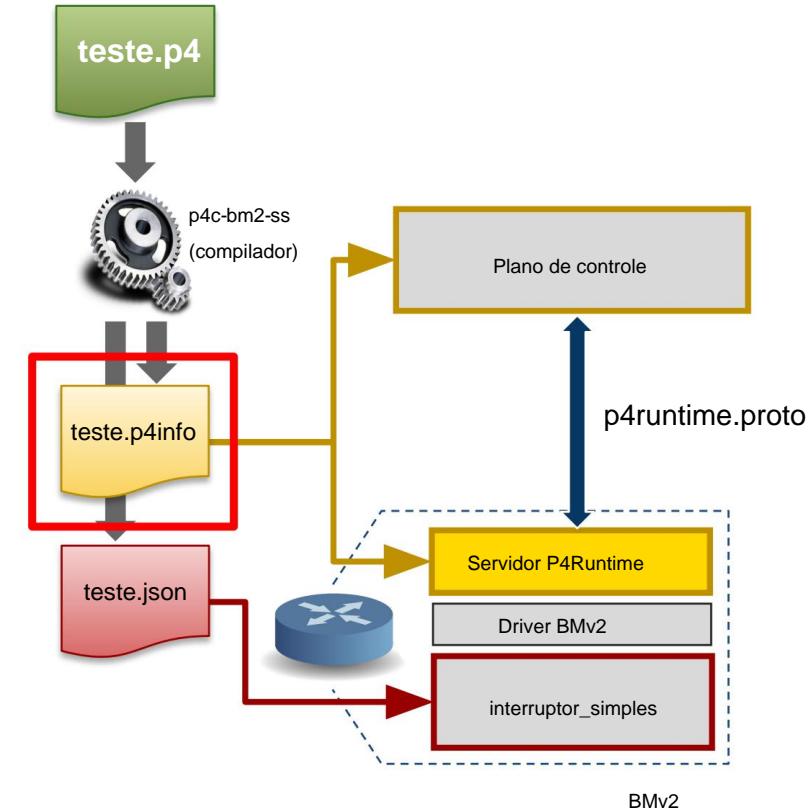
Fluxo de trabalho P4Runtime

P4Info

- Captura atributos do programa P4 necessários em tempo de execução
 - ÿ IDs para tabelas, ações, parâmetros, etc.
 - ÿ Estrutura da tabela, parâmetros de ação, etc.
- Formato baseado em Protobuf
- Saída do compilador independente do alvo
 - ÿ Mesmo P4Info para BMv2, ASIC, etc.

Especificação completa do protobuf

P4Info: <https://github.com/p4lang/PI/blob/master/proto/p4/config/v1/p4info.proto>



Exemplo de P4Info

roteador_básico.p4

```

...
ação ipv4_forward (bit<48> dstAddr bit<9> porta) { ,
    /* Implementação de ação */
}

...
tabela ipv4_lpm
{ chave
    = { hdr.ipv4.dstAddr: lpm ;
} ações =
{ ipv4_forward;
...
}
...
}
```



Compilador P4

roteador_básico.p4info

```

ações { id:
16786453 nome:
"ipv4_forward" parâmetros { id: 1
nome:
"dstAddr"
largura de bits: 48
...
id: 2
nome: "porta"
largura de bits: 9
}
...
tabelas { id:
33581985 nome:
"ipv4_lpm"
campos_de_correspondência { id: 1 nome:
"hdr.ipv4.dstAddr"
largura de bits: 32 tipo_de_correspondência: LPM
} ação_ref_id: 16786453
}
```

Exemplo de entrada de tabela P4Runtime

roteador_básico.p4

```
ação ipv4_forward(bit<48> dstAddr, bit<9> porta) {
    /* Implementação de ação */

} tabela ipv4_lpm
{ chave =
    { hdr.ipv4.dstAddr: lpm;

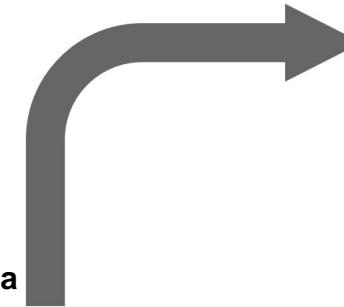
} ações =
{ ipv4_forward;
...
}
...
}
```



Visão lógica da entrada da tabela

hdr.ipv4.dstAddr=10.0.1.1/32 ->
ipv4_forward(00:00:00:00:00:10, 7)

O plano de
controle gera



Mensagem do Protobuf

```
table_entry
{ table_id: 33581985 match
{ field_id:
    1 lpm { value:
        "\n\000\001\001" prefix_len: 32
    }
} ação
{ action_id: 16786453
params
{ param_id: 1
valor: "\000\000\000\000\000\n"
} params
{ param_id: 2
valor: "\000\007"
}
}
```

P4Runtime SetPipelineConfig

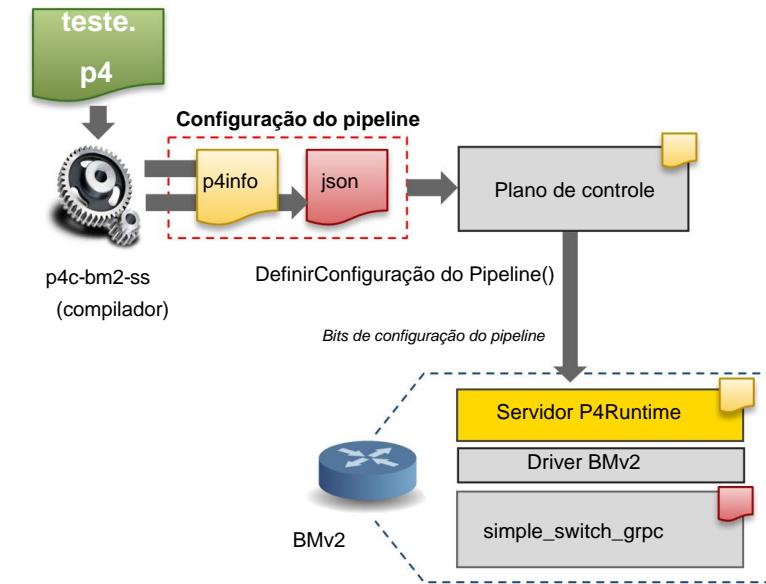
```

mensagem SetForwardingPipelineConfigRequest { enum Ação {

    NÃO ESPECIFICADO = 0;
    VERIFICAR = 1;
    VERIFICAR_E_SALVAR = 2;
    VERIFICAR_E_CONFIRMAR = 3;
    COMPROMISSO = 4;
    RECONCILIAR_E_COMMITIR = 5;

} uint64 device_id = 1; uint64
role_id = 2;
Uint128 eleição_id = 3;
Ação ação = 4;
Configuração do ForwardingPipelineConfig = 5; }

```



```

mensagem ForwardingPipelineConfig { config.P4Info
p4info = 1; // Configuração P4
específica do alvo. bytes p4_device_config = 2; }

```

Canal de transmissão P4Runtime

```

mensagem StreamMessageRequest { uma
de atualização {
    MasterArbitrationUpdate arbitragem
        = 1;
    Pacote PacketOut = 2;
}
}

```

// Pacote enviado do controlador para o switch. **message PacketOut**
{ bytes payload = 1; // Isso
será baseado no cabeçalho
P4 anotado como // @controller_header("packet_out").

// No máximo um cabeçalho P4 pode ter esta anotação. **repetido**
PacketMetadata metadata = 2; }

```

mensagem StreamMessageResponse { uma
de atualização {
    MasterArbitrationUpdate arbitragem
        = 1;
    PacoteEm pacote = 2;
}
}

```

// Pacote enviado do switch para o controlador. **message PacketIn**
{ bytes payload = 1; // Isso
será baseado no cabeçalho
P4 anotado como // @controller_header("packet_in").

// No máximo um cabeçalho P4 pode ter esta anotação. **repetido**
PacketMetadata metadata = 2; }

Parâmetros comuns do P4Runtime

- **device_id** ↴

Especifica o chip de encaminhamento específico ou ponte de software ↴ **Definido como 0 para plataformas de chip único**

- **role_id**

↳ Corresponde a uma função com capacidades específicas (ou seja, quais operações, P4 entidades, comportamentos, etc. estão no escopo de uma determinada função)
↳ A definição de função é atualmente acordada entre os planos de controle e de dados
desconectado

↳ **Role_id padrão (0) tem acesso total ao pipeline**

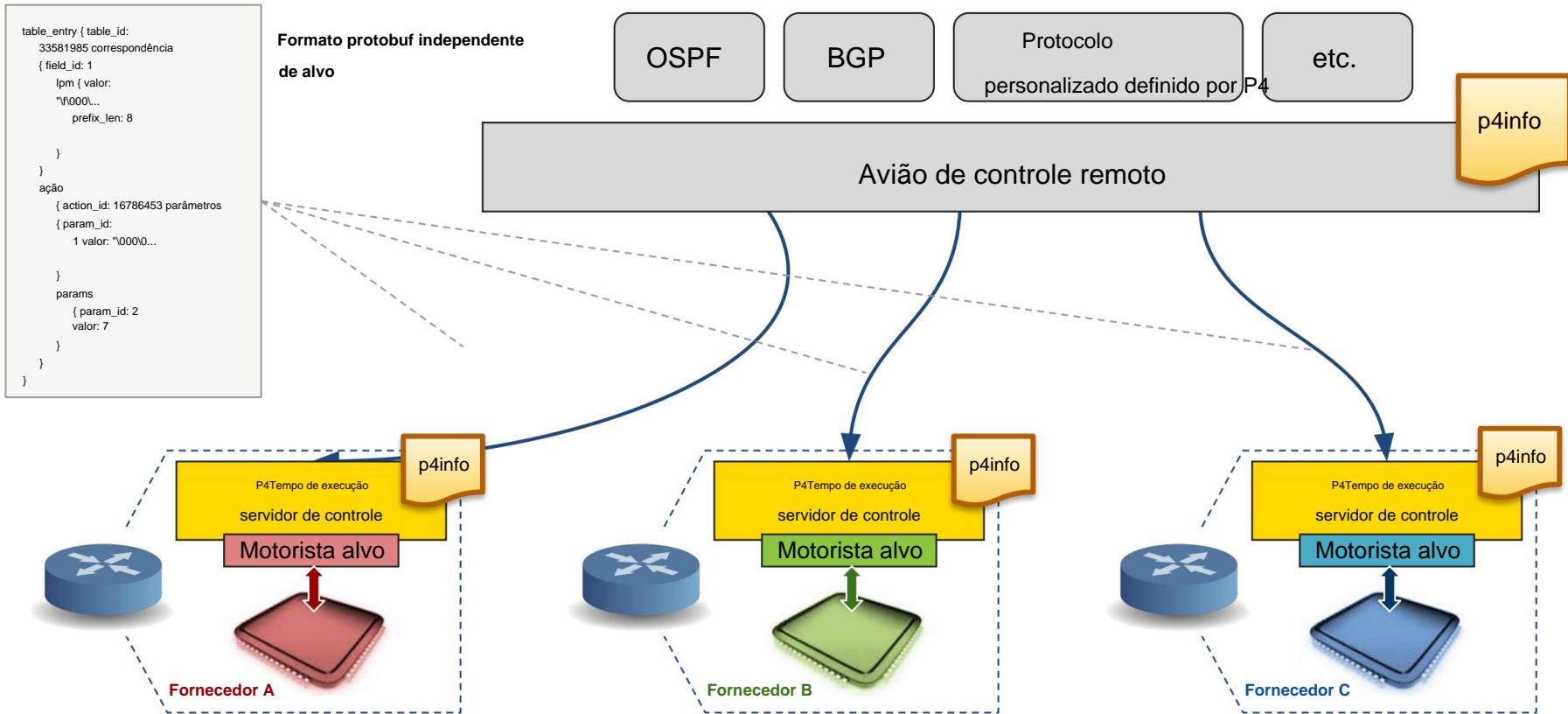
- **election_id** ↴
P4Runtime oferece suporte

ao mestrado por função ↴ O cliente com o ID de eleição mais alto é chamado de "mestre", enquanto todos outros clientes são chamados de "escravos" ↴ **Defina como 0 para controladores de instância única**

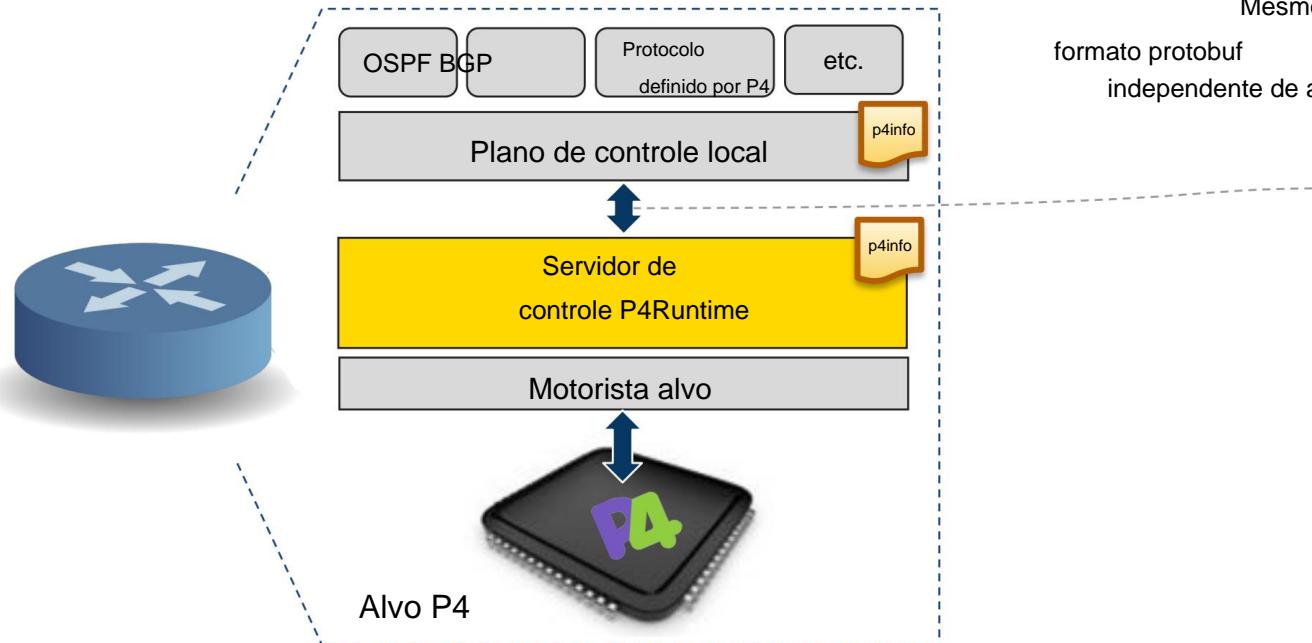
Arbitragem de Mestrado

- Ao conectar-se ao dispositivo, o cliente (por exemplo, controlador) precisa abrir um StreamChannel
- O cliente deve anunciar seu role_id e election_id usando um Mensagem MasterArbitrationUpdate
 - ÿ Se role_id não for definido, isso implicará na função padrão e será concedido acesso total ao pipeline.
 - ÿ O election_id é opaco para o servidor e determinado pelo controle plano (pode ser omitido para plano de controle de instância única)
- O switch marca o cliente para cada função com o election_id mais alto como mestre
- O Mestre pode:
 - ÿ Executar solicitações de gravação
 - ÿ Receber mensagens PacketIn
 - ÿ Enviar mensagens PacketOut

Controle remoto



Controle local



Mesmo formato protobuf independente de alvo

```
table_entry { table_id: 33581985 correspondência { field_id: 1 lpm { valor: "\0000..." prefix_len: 8 } } }
ação { action_id: 16786453 parâmetros { param_id: 1 valor: "\0000..." } }
params { param_id: 2 valor: 7 }
```

O P4Runtime pode ser usado igualmente bem por um plano de controle remoto ou local

Recapitação da API P4Runtime

Coisas que abordamos:

- Definição de P4Runtime
- P4Info
- Entradas de tabela
- Definir configuração de pipeline
- Suporte para entrada/saída de pacotes
- Replicação de controlador
 - ÿ Via arbitragem mestre-escravo

O que não abordamos:

- Como controlar outras entidades P4
 - ÿ Externos, contadores, medidores
- Leituras/gravações em lote
- Configuração do switch
 - ÿ Fora do escopo do P4Runtime
 - ÿ Alcançado com outros mecanismos
 - ÿ por exemplo, OpenConfig e gNMI

Exercício P4Runtime

Visão geral do exercício

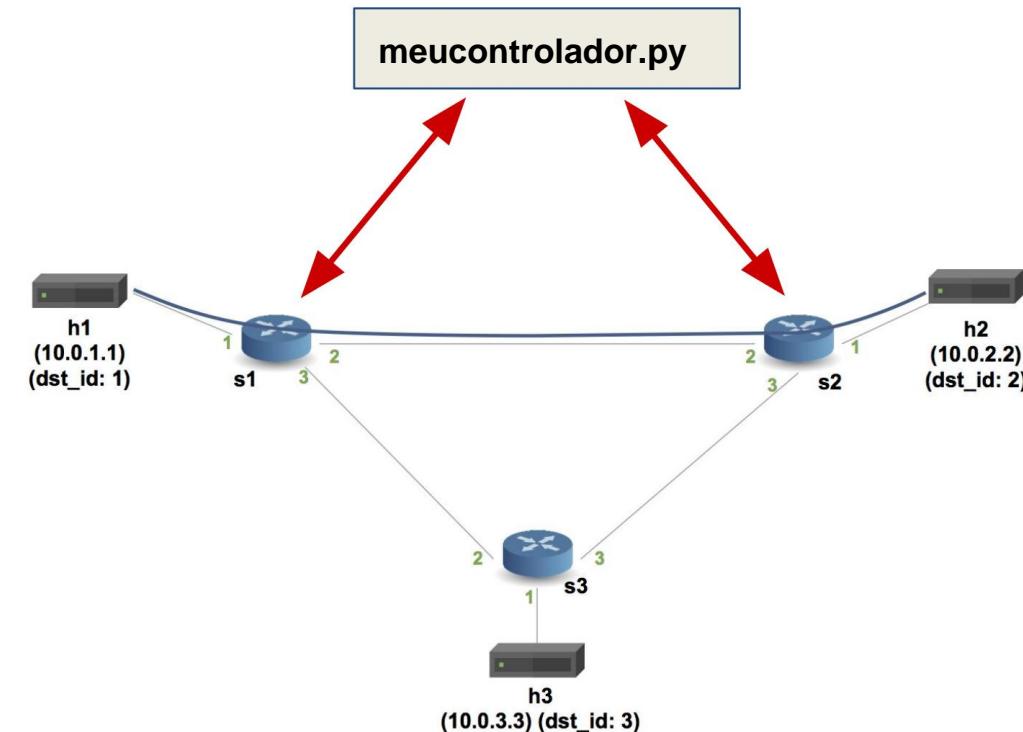
Responsabilidades do controlador:

1. Estabeleça uma conexão gRPC com os switches para o serviço P4Runtime
2. Empurre o programa P4 para cada switch
3. Escreva as regras de encaminhamento do túnel:
 - a. regra **myTunnel_ingress** para encapsular pacotes no switch de entrada

- b. regra **myTunnel_forward** para encaminhar pacotes no switch de entrada

- c. regra **myTunnel_egress** para desencapsule e encaminhe pacotes no switch de saída 4. Leia

os contadores de entrada e saída do túnel a cada 2 segundos



Começando

O código-fonte já foi baixado na sua VM: `~/tutorials/exercises/p4runtime`

Você deve começar lendo o [README.md](#)

Neste exercício, você precisará concluir a implementação de `writeTunnelRules` em `mycontroller.py`

Você precisará de duas janelas de Terminal: uma para sua rede de plano de dados (Mininet) que você começará a usar o [make](#), e a outra para seu programa controlador.

Para encontrar o código-fonte:

<https://github.com/p4lang/tutorials/>

README.md

Implementing a Control Plane using P4 Runtime

Introduction

In this exercise, we will be using P4 Runtime to send flow entries to the switch instead of using the switch's CLI. We will be building on the same P4 program that you used in the [basic_tunnel](#) exercise. The P4 program has been renamed to `advanced_tunnel.py` and has been augmented with two counters (`ingressTunnelCounter`, `egressTunnelCounter`) and two new actions (`myTunnel_ingress`, `myTunnel_egress`).

You will use the starter program, `mycontroller.py`, and a few helper libraries in the `p4runtime_lib` directory to create the table entries necessary to tunnel traffic between host 1 and 2.

Spoiler alert: There is a reference solution in the `solution` sub-directory. Feel free to compare your implementation to the reference.

Step 1: Run the (incomplete) starter code

The starter code for this assignment is in a file called `mycontroller.py`, and it will install only some of the rules that you need to tunnel traffic between two hosts.

Let's first compile the new P4 program, start the network, use `mycontroller.py` to install a few rules, and look at the `ingressTunnelCounter` to see that things are working as expected.

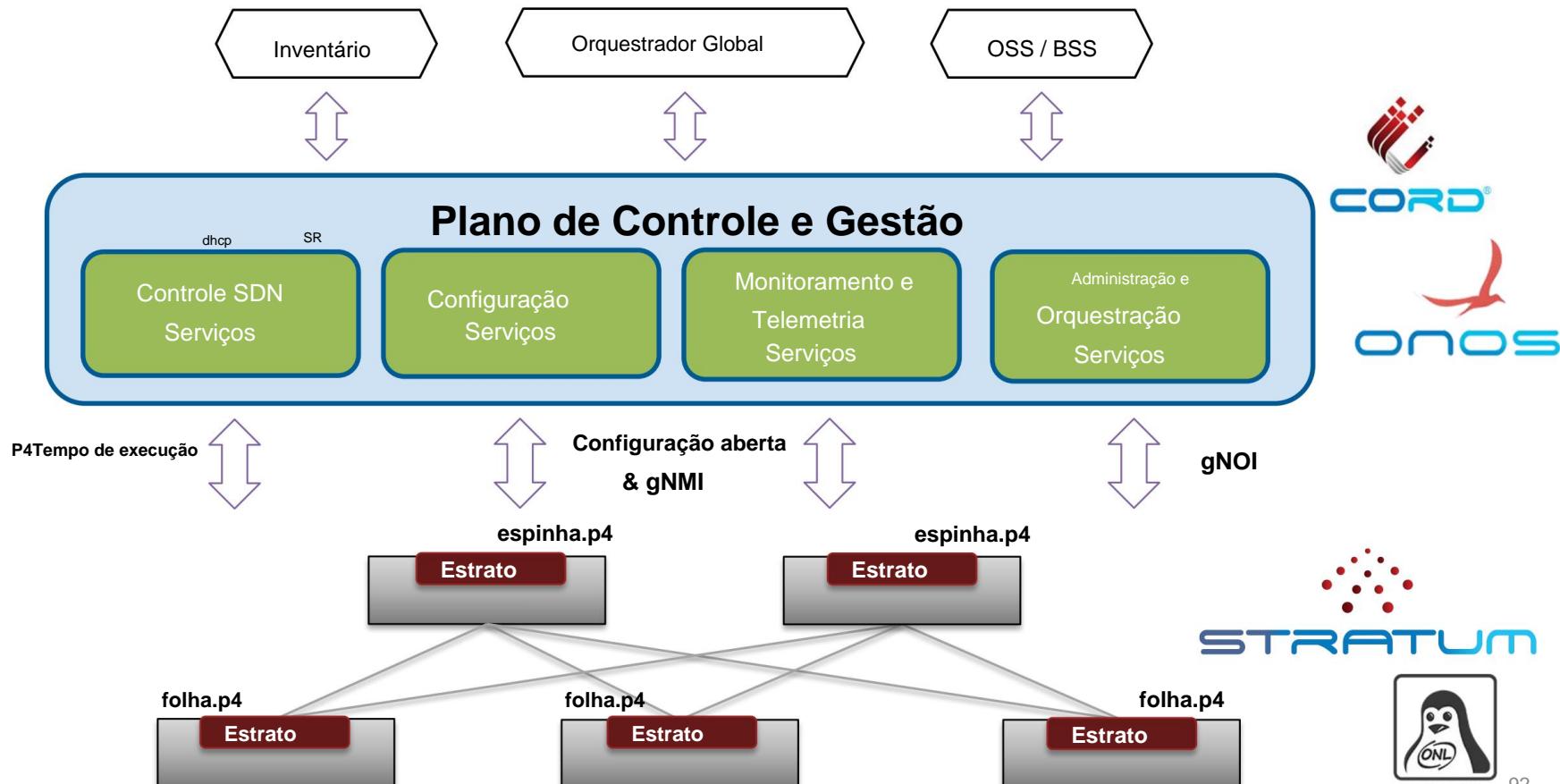
1. In your shell, run:

```
make
```



P4Runtime na natureza

Pilha SDN de código aberto habilitada para P4Runtime



Algumas dicas

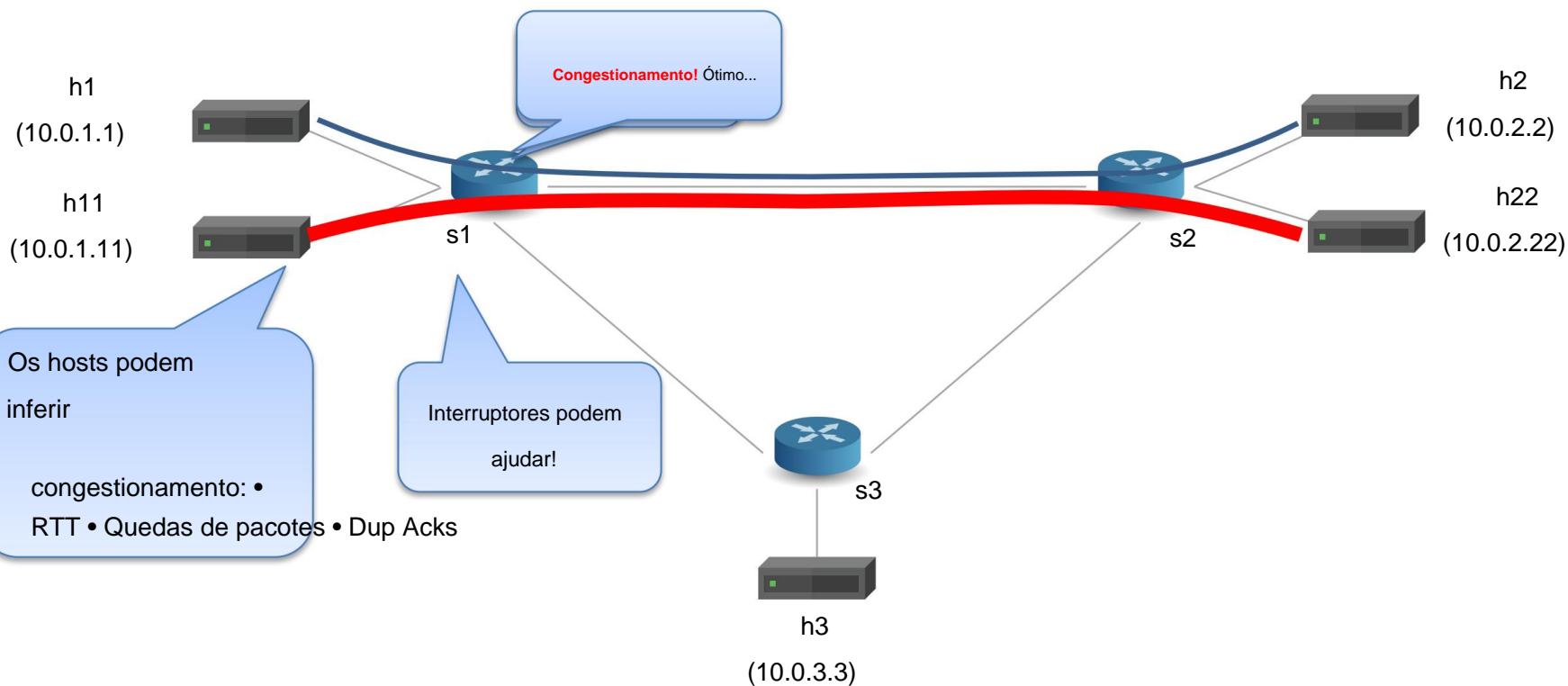
<https://stratumproject.org/>

<https://onosproject.org/>

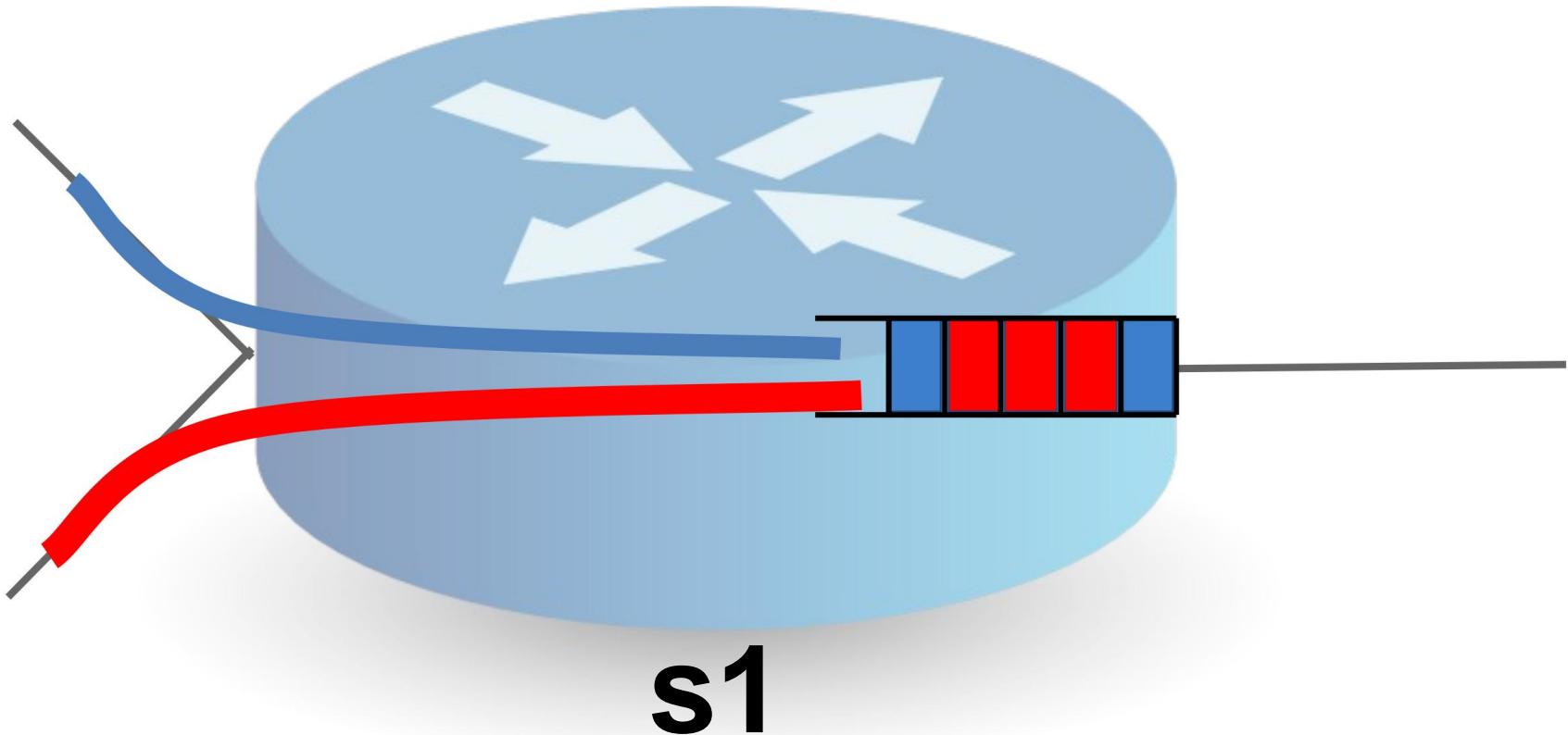
<https://wiki.onosproject.org/display/ONOS/P4+brigade>

Laboratório 3: Monitoramento e Depuração

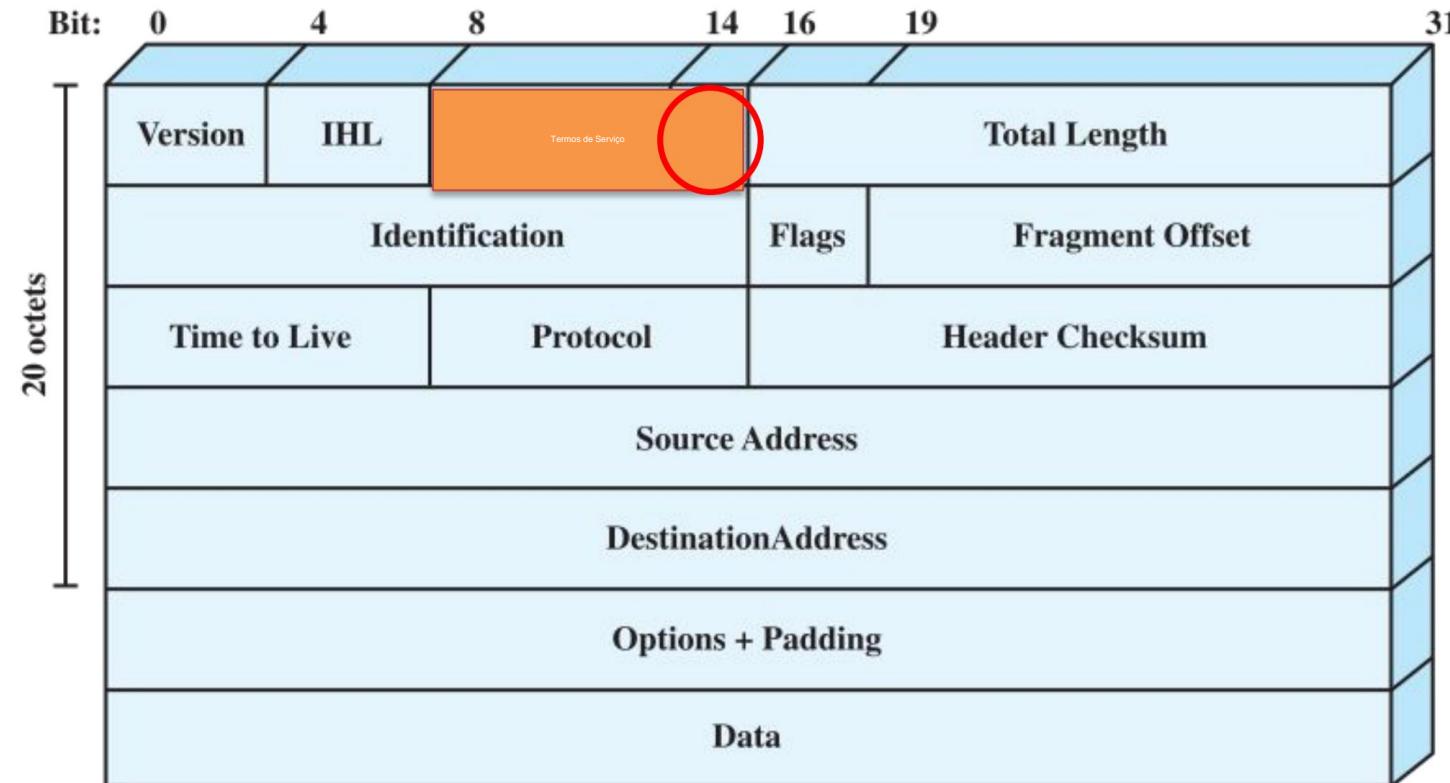
Monitoramento e Depuração



Monitoramento e Depuração



Notificação explícita de congestionamento



Notificação explícita de congestionamento

- **Notificação explícita de congestionamento**

- ÿ00: Transporte não compatível com ECN, não ECT

- ÿ10: Transporte compatível com ECN, ECT(0)

- ÿ01: Transporte compatível com ECN, ECT(1)

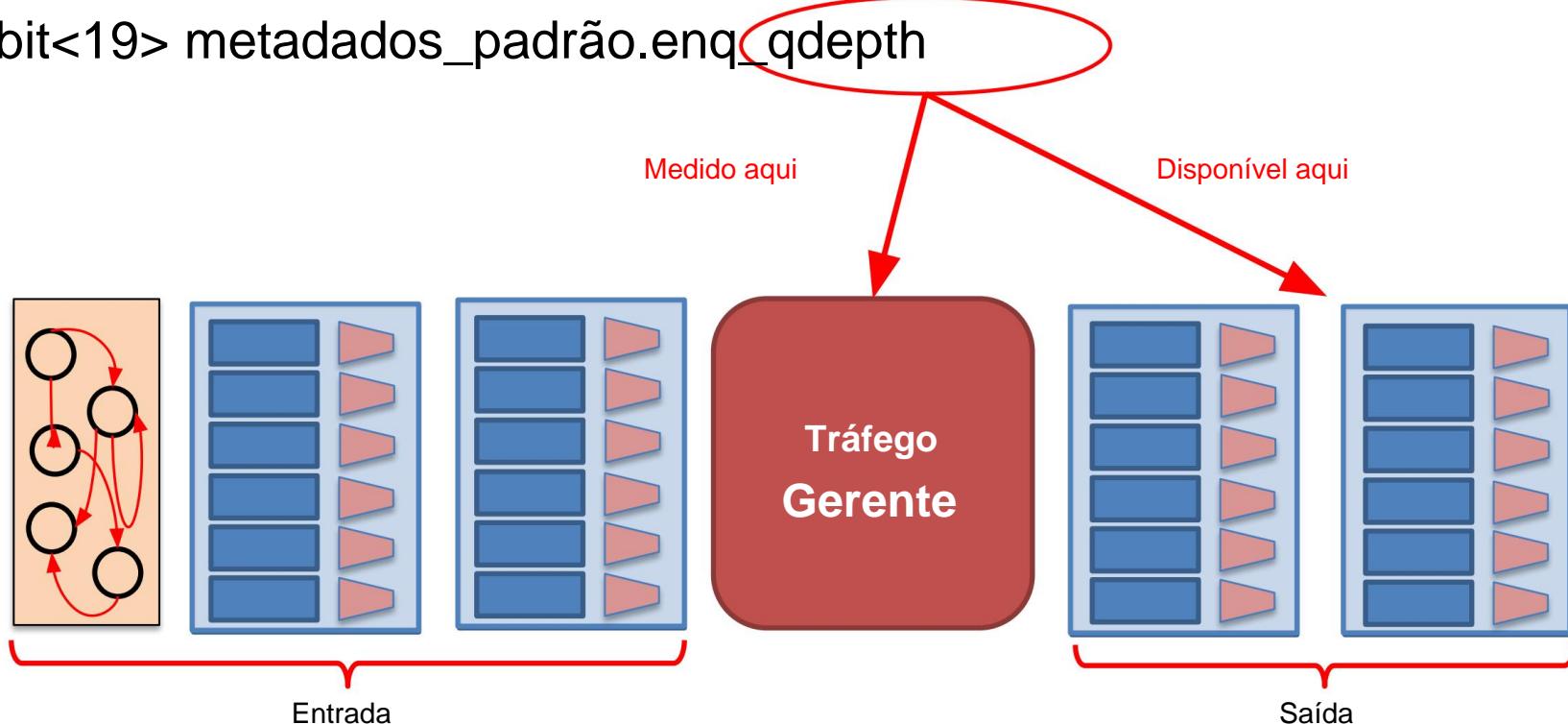
- ÿ11: Congestionamento encontrado, CE

- **Para pacotes originados de ECT, os switches com capacidade ECN definem o bit CE em caso de congestionamento**

- ÿPor exemplo, profundidade de fila observada > limite

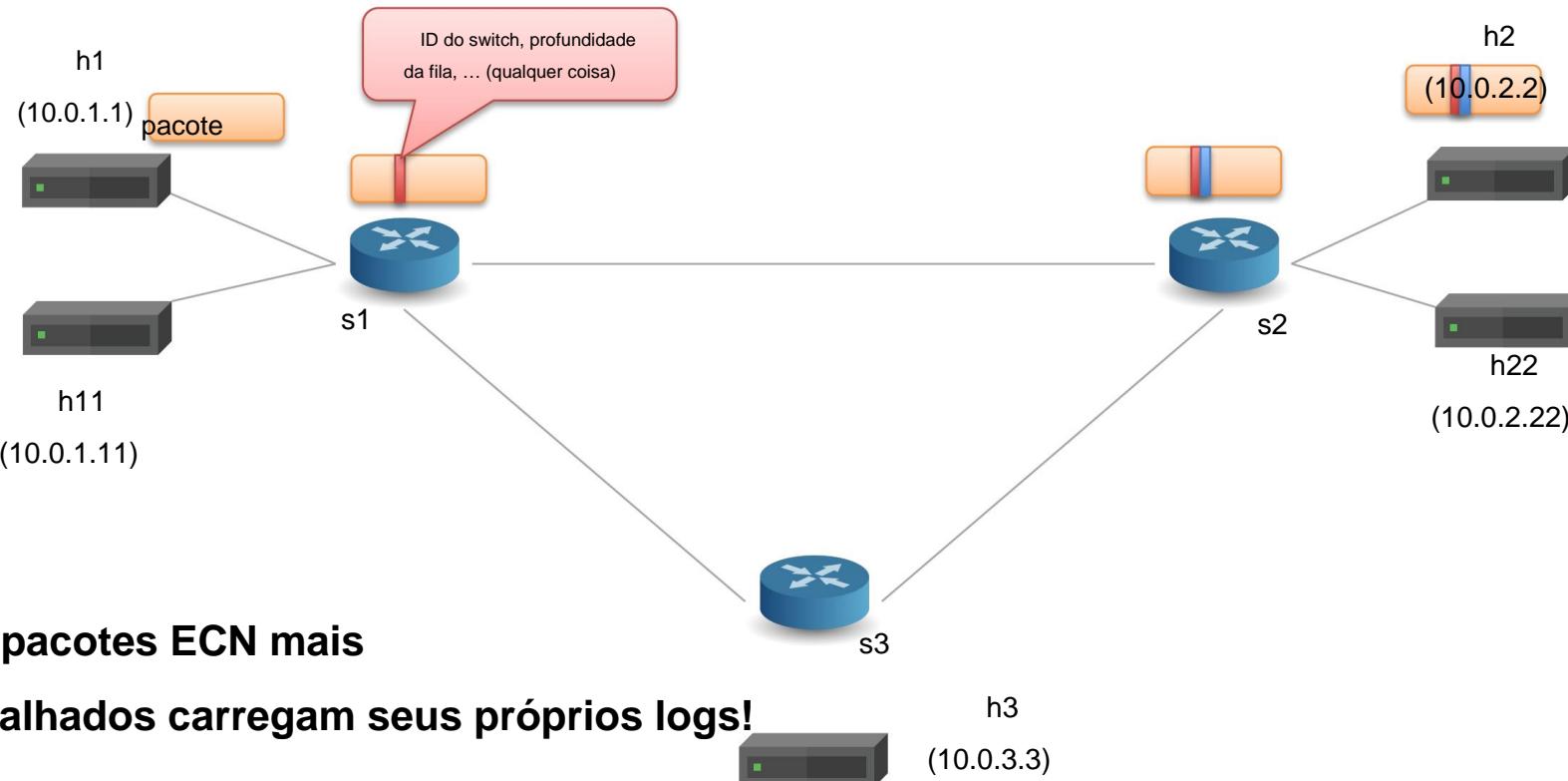
Notificação de congestionamento explícita no P4

- Os dados padrão para o V1Model incluem a profundidade da fila:
bit<19> metadados_padrão.enq_qdepth



Intervalo de codificação

Inspeção Multi-Rota



Inspeção de múltiplas rotas: formato de pacote

```
cabeçalho mri_t
```

```
{ bit<16> contagem;
```

```
}
```

```
cabeçalho switch_t {
```

```
switchID_t swid;
```

```
qprofundidade_t qprofundidade;
```

```
}
```

cabeçalhos de estrutura

```
{ ethernet_t ethernet; ipv4_t ipv4; opção_ipv4_t
```

```
opção_ipv4; mri_t mri;
```

```
switch_t[MAX_HOPS] swtraces;
```

```
}
```

- Operações de validade de

 - cabeçalho: o `hdr.setValid(): add_header`

 - `hdr.setInvalid(): remove_header` o `hdr.isValid(): test validity`

- Pilhas de cabeçalho

 - o `hdr[Cnt] stk;` • Pilhas de cabeçalho em analisadores

 - o `stk.next` o

 - `stk.last` o

 - `stk.lastIndex` • Pilhas de

- Pilhas de cabeçalho em controles

 - o `stk[i]` o

 - `stk.size` o

 - `stk.push_front(int count)` o `stk.pop_front(int count)`

Verificação de cabeçalho

```
/* Erros padrão, definidos em core.p4 */

erro
{
    { NoError,                      // sem erro //
      PacketTooShort,              bits insuficientes no pacote para extração // a expressão
      NoMatch,                     de correspondência não possui correspondências
      StackOutOfBounds,           // referência a elemento inválido de uma pilha de cabeçalhos OverwritingHeader, // um
      cabeçalho é extraído duas vezes HeaderTooShort, ParserTimeout
                                  // extraíndo muitos bits em um campo varbit // tempo limite
                                  de execução do analisador excedido
    }
}
```

```
/* Erro adicional adicionado pelo programador */
```

```
erro { IPv4BadHeader }
```

```
...
estado parse_ipv4 {
    pacote.extraer(hdr.ipv4);
    verificar(hdr.ipv4.version == 4, erro.IPV4BadHeader);
    transição aceitar;
}
```

Intervalo de codificação

Perguntas e Respostas



<https://bit.ly/join-p4-lang-slack>

Junte-se ao canal #d2-2018-spring

Perguntas para TAs }



Pigeonhole®

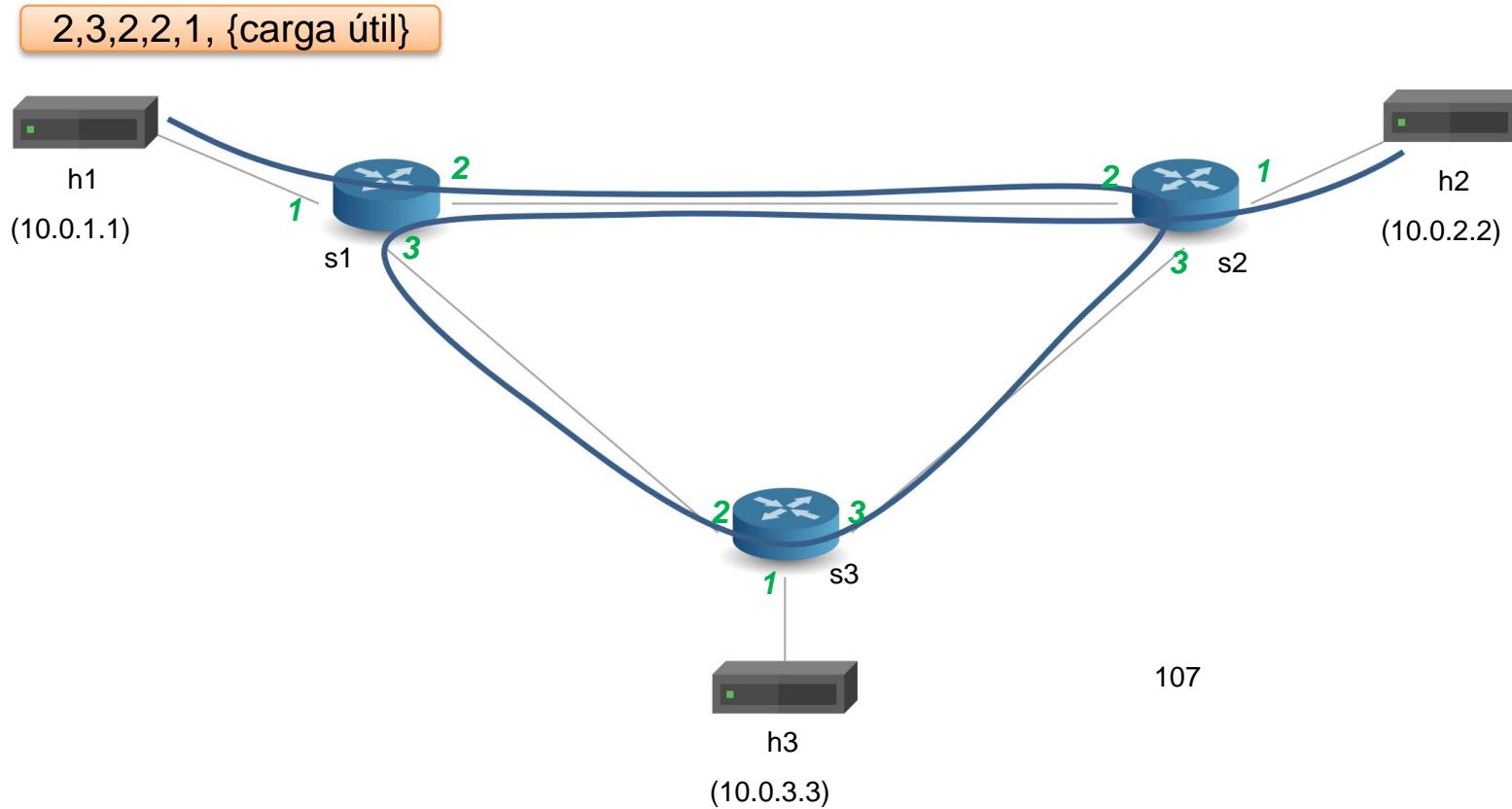
<https://pigeonhole.at>

Código do evento: P4D2

Perguntas para o Painel }

Laboratório 4: Comportamento Avançado

Roteamento de origem



Roteamento de origem: formato de pacote

```
#define MAX_HOPS 9

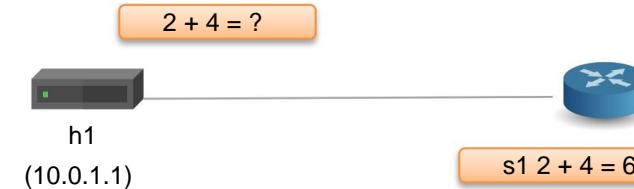
const bit<16> TIPO_IPV4 = 0x800;
const bit<16> TIPO_SRCROUTING = 0x1234;
cabeçalho srcRoute_t {
    bit<1> bos;
    bit<15> porta;
}

cabeçalhos de estrutura {
    ethernet_t ethernet;
    srcRoute_t[SALTOS_MÁXIMOS] srcRoutes;
    ipv4_t ipv4;
}
```

- Analisar rotas de origem somente se etherType for 0x1234
- O valor especial bos == 1 indica o “fundo da pilha”
- Encaminhar pacotes usando rotas de origem e também diminuir o TTL do IPv4
- Descarte o pacote se as rotas de origem não estiverem válidas
- Dica: Use as próximas primitivas pop_front pacote.extrair(hdr.srcRoutes.next) hdr.srcRoutes.pop_front(1)

Intervalo de codificação

Calculadora



Calculadora: Formato de Pacote



Inicializadores de tabela

```
tbl
{ chave = { hdr.hf : exato }
ações = { a1; a2; a3 }
entradas = {
    { 0x01 } : a1(1);
    { 0x02 } : a1(2);
    { _ } : SemAção();
}
}
```

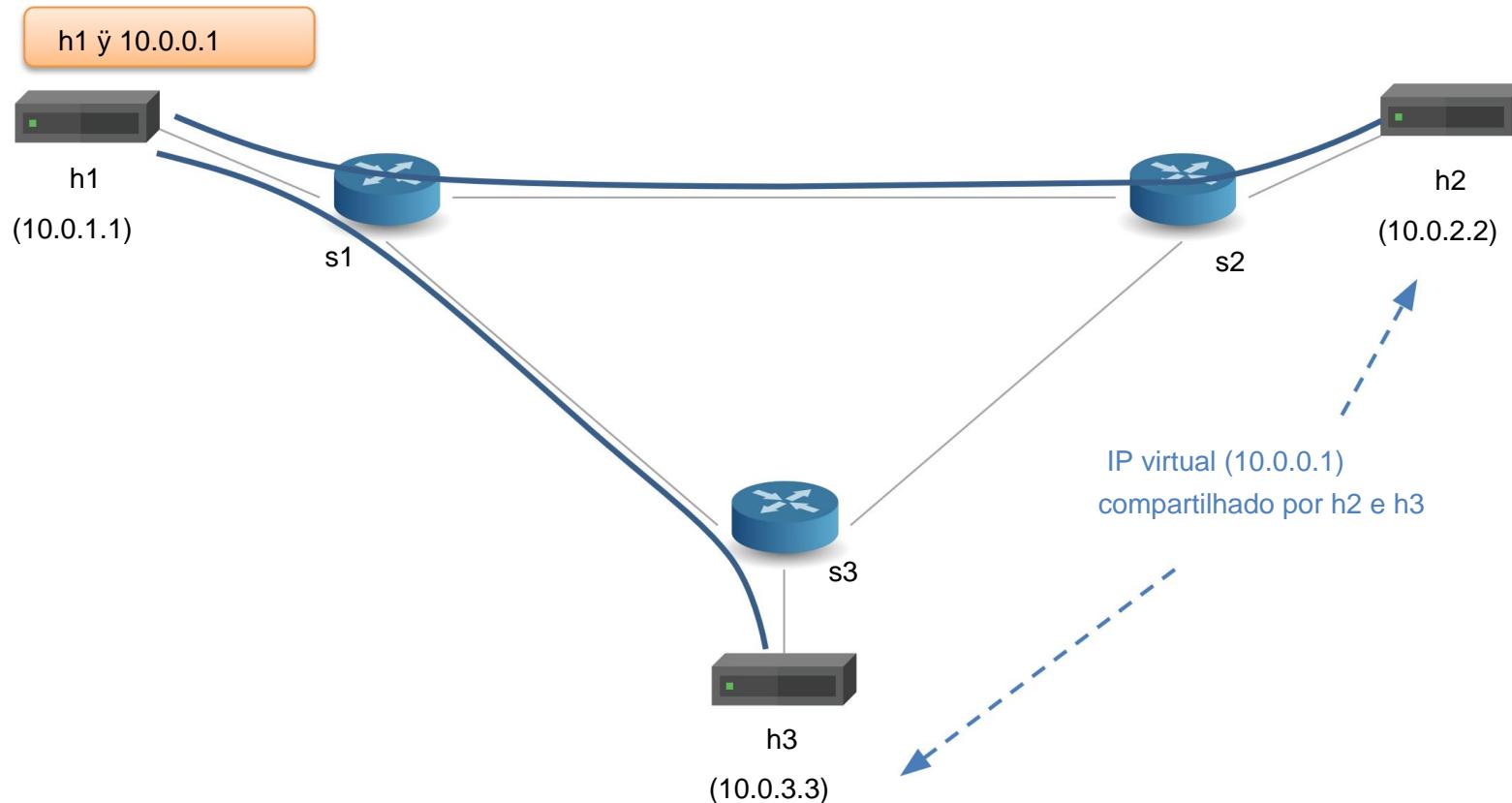
Pode inicializar tabelas com entradas constantes

Deve especificar completamente o valor de todos os dados de ação, incluindo valores que normalmente são fornecidos pelo plano de controle

Dica: para a calculadora, use uma tabela que corresponda ao código operacional

Intervalo de codificação

Balanceamento de carga simples



Hashing (Modelo V1)

```
enum HashAlgoritmo {  
    csum16,  
    xor16,  
    crc32,  
    crc32_custom,  
    crc16,  
    crc16_customizado,  
    aleatório,  
    identidade  
}  
externo vazio hash<O, T, D, M>(  
    fora O resultado,  
    no algoritmo HashAlgorithm,  
    na base T,  
    nos dados D,  
    em M max);
```

**Calcula o hash dos dados
(usando algoritmo) módulo máximo
e adiciona à base**

**Usa variáveis de tipo
(como modelos C++ /
Java Generics) para permitir
que o primitivo de hash
seja usado com muitos tipos dife**

Conclusão e próximos passos

Por que P4?

- **Semântica claramente definida** ↗

Você pode descrever o que seu programa de plano de dados está fazendo • **Expressivo**

- **Alto nível, independente de destino** ↗ Usa construções convencionais ↗ O compilador gerencia

- os recursos e lida com o hardware • **Tipo seguro** ↗ Aplica boas práticas de design de software e

- elimina bugs “estúpidos” • **Agilidade** ↗ Dispositivos de rede de alta velocidade se tornam tão flexíveis

- quanto qualquer software • **Insight** ↗ Mistura livremente cabeçalhos de pacotes e resultados intermediários

Coisas que abordamos

- A “visão de mundo” P4

- ÿ Processamento de pacotes independente de protocolo
 - ÿ Separação de linguagem/arquitetura
 - ÿ Se você puder interagir com ele, ele pode ser usado

- Tipos de dados

- principais**
 - Construções para análise de pacotes
 - ÿ Programação em estilo de máquina de estados
 - Construções para processamento de pacotes
 - ÿ Ações, tabelas e controles

- Desanálise de pacotes
- Arquiteturas e programas

Coisas que não abordamos

- **Mecanismos de modularidade** ↗

Instanciação e invocação de analisadores ou
controles • **Detalhes do processamento de campos de
comprimento variável** ↗ Análise e desanálise de opções e TLVs

- **Construções de definição de arquitetura**

↗ Como essas definições “modeladas” são criadas

- **Recursos avançados**

↗ Como fazer aprendizado, multicast, clonagem, reenvio ↗ Uniões
de cabeçalho

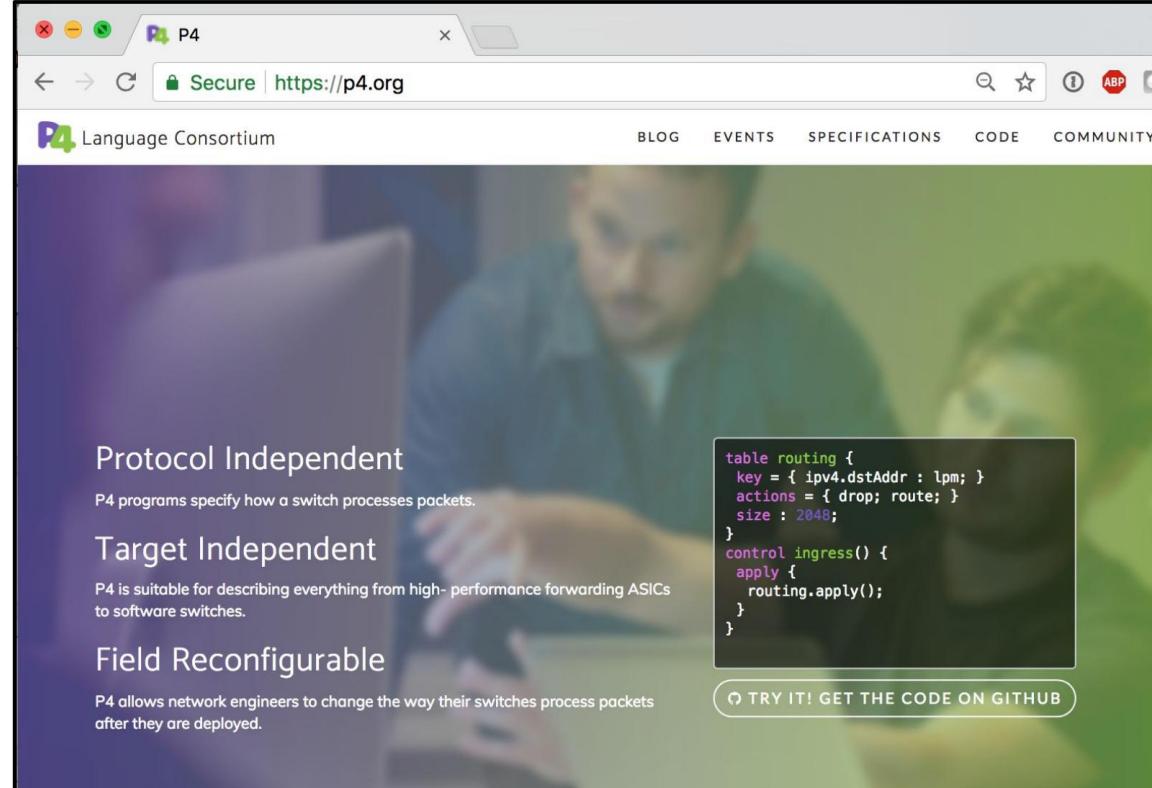
- **Outras arquiteturas** •

Interface do plano de controle



O Consórcio de Línguas P4

- Consórcio de membros
acadêmicos e industriais
- Código aberto, em evolução,
linguagem específica de domínio
- Licença Apache permissiva, código
no GitHub hoje
- A adesão é gratuita:
contribuições são bem-vindas
- Independente, constituída como
Organização sem fins lucrativos da Califórnia



The screenshot shows the P4.org website homepage. At the top, there's a navigation bar with links for BLOG, EVENTS, SPECIFICATIONS, CODE, and COMMUNITY. Below the navigation, there's a large image of a person working at a computer. Overlaid on the image are three sections of text:

- Protocol Independent**: P4 programs specify how a switch processes packets.
- Target Independent**: P4 is suitable for describing everything from high- performance forwarding ASICs to software switches.
- Field Reconfigurable**: P4 allows network engineers to change the way their switches process packets after they are deployed.

To the right of the image, there's a code snippet in the P4 language:

```
table routing {
    key = { ipv4.dstAddr : lpm; }
    actions = { drop; route; }
    size : 2048;
}
control.ingress() {
    apply {
        routing.apply();
    }
}
```

At the bottom right, there's a button labeled "TRY IT! GET THE CODE ON GITHUB".