Redes Definidas por Software
Aula 4

Sumário

Switch-Level Schematic

Começamos considerando um switch bare-metal como um todo, onde a melhor analogia é imaginar um PC construído a partir de uma coleção de componentes prontos para uso.

Na verdade, uma especificação arquitetônica completa para switches que aproveitam tais componentes está disponível on-line no Open Compute Project (OCP).

Isto é o equivalente em hardware ao software de código aberto e possibilita que qualquer pessoa construa um switch de alto desempenho, análogo a um PC caseiro. Mas assim como o ecossistema do PC inclui fornecedores de servidores comuns como Dell e HP, você pode comprar um switch pré-construído (compatível com OCP) de fornecedores de switches bare-metal, como EdgeCore, Delta e outros.

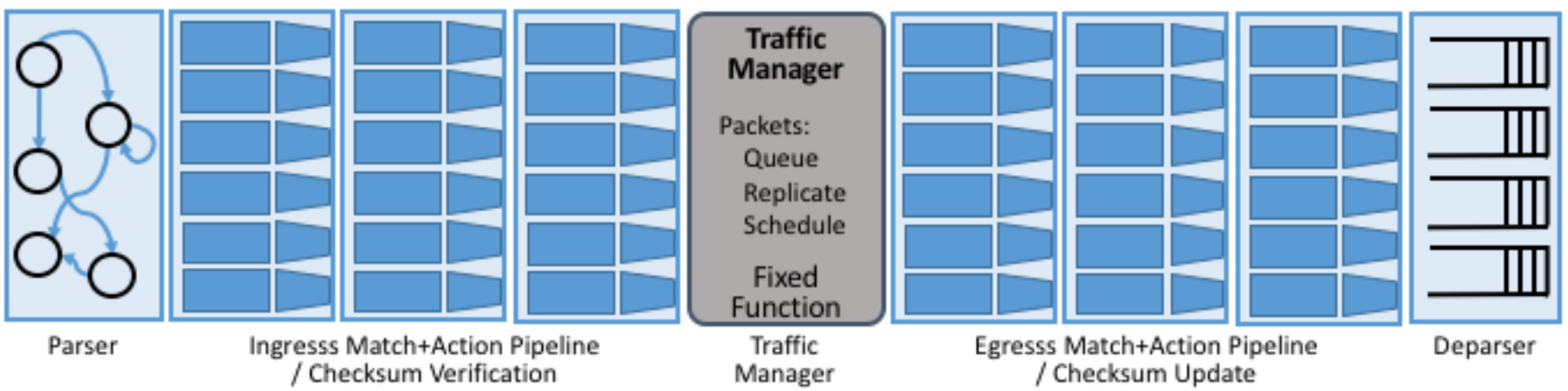
SWITCHES BARE-METAL

Este capítulo descreve os switches bare-metal que fornecem a base de hardware subjacente para SDN. O nosso objetivo não é fornecer um esquema de hardware detalhado, mas sim esboçar o suficiente do projeto para apreciar o pilha de software que é executada sobre ela. Observe que esta pilha ainda está evoluindo, com diferentes aplicações de implementação, abordagens adotadas ao longo do tempo e por diferentes fornecedores.

Portanto, este capítulo discute tanto o P4 como um abordagem para programar o plano de dados do switch e OpenFlow como uma alternativa de primeira geração. Vamos introduzir essas duas abordagens em ordem cronológica inversa, começando com a abordagem mais geral e programável caso de P4.

NPU (Network Processing Unit)

A Figura seguinte apresenta um esquema resumido de um switch bare-metal. A Unidade de Processamento de Rede (NPU) – uma chip de comutação de silício comercial – é otimizado para analisar cabeçalhos de pacotes e tomar decisões de encaminhamento.
NPUs são capazes de processar e expedir pacotes a taxas medidas em Terabits por segundo (Tbps), com facilidade e rapidez para acompanhar as portas de 32x100 Gbps ou as portas de 48x40 Gbps mostradas na figura. No momento em que este livro foi escrito, o o que há de mais moderno para esses chips é 25,6 Tbps com portas de 400 Gbps.



Network Processing Unit (NPU)

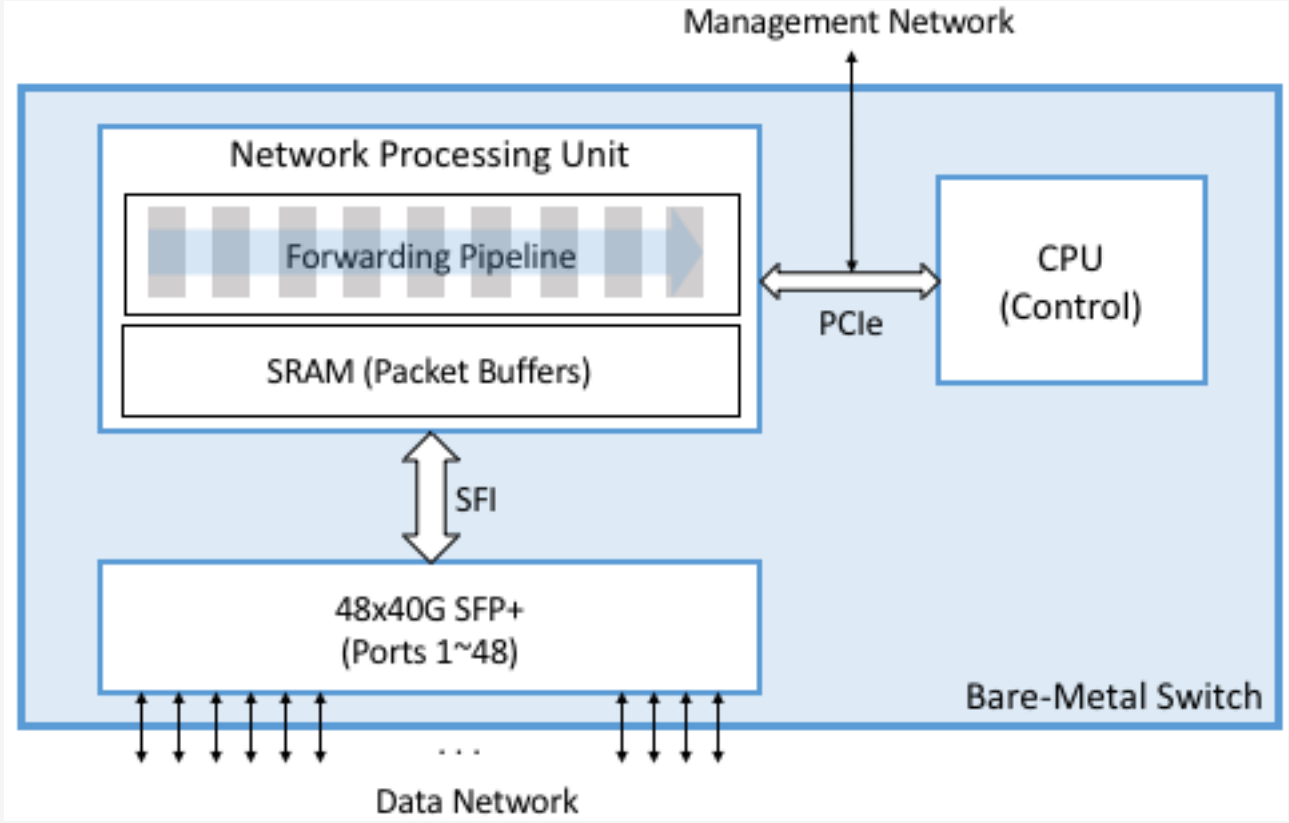
Observe que nosso uso do termo NPU pode ser considerado um pouco fora do padrão.
Historicamente, a NPU foi a nome dado a chips de processamento de rede definidos de forma mais restrita, usados, por exemplo, para implementar firewalls ou inspeção profunda de pacotes.

A tendência de longo prazo, no entanto, tem sido em direção a NPUs que correspondam o desempenho de ASICs (*application specific integrated circuit*) de função fixa, proporcionando ao mesmo tempo um grau muito maior de flexibilidade.

Parece provável que os atuais chips de comutação de silício comercial tornarão a geração anterior de redes especialmente construídas com processadores obsoletos.

A nomenclatura NPU usada aqui é consistente com a tendência geral da indústria de construir processadores programáveis específicos de domínio, incluindo GPUs (Graphic Processing Units) para gráficos e TPUs (unidades de processamento de tensores) para IA.

Esquema de alto nível do bare-metal switch



Pipeline de vários estágios

Os switches de alta velocidade usam um pipeline de vários estágios para processar pacotes.
A relevância de usar um multi-estágio pipeline em vez de um processador de estágio único é que o encaminhamento de um único pacote provavelmente envolve olhar para vários campos de cabeçalho.
Cada estágio pode ser programado para observar uma combinação diferente de campos.
Um estágio multi- pipeline adiciona um pouco de latência ponta a ponta a cada pacote (medida em nanossegundos), mas significa que vários pacotes podem ser processados ao mesmo tempo.

A principal distinção em como uma determinada NPU implementa esse pipeline é se os estágios são de função fixa. (ou seja, cada estágio entende como processar cabeçalhos para algum protocolo fixo) ou programável (ou seja, cada estágio é programado dinamicamente para saber quais campos de cabeçalho processar).
Na discussão a seguir, começa com o caso mais geral – um pipeline programável – e retorna-se ao seu equivalente de função fixa no fim.

PISA (Protocol Independent Switching Architecture)

No nível arquitetônico, o pipeline programável é frequentemente chamado de Protocol Independent Switching. Arquiteture (PISA).

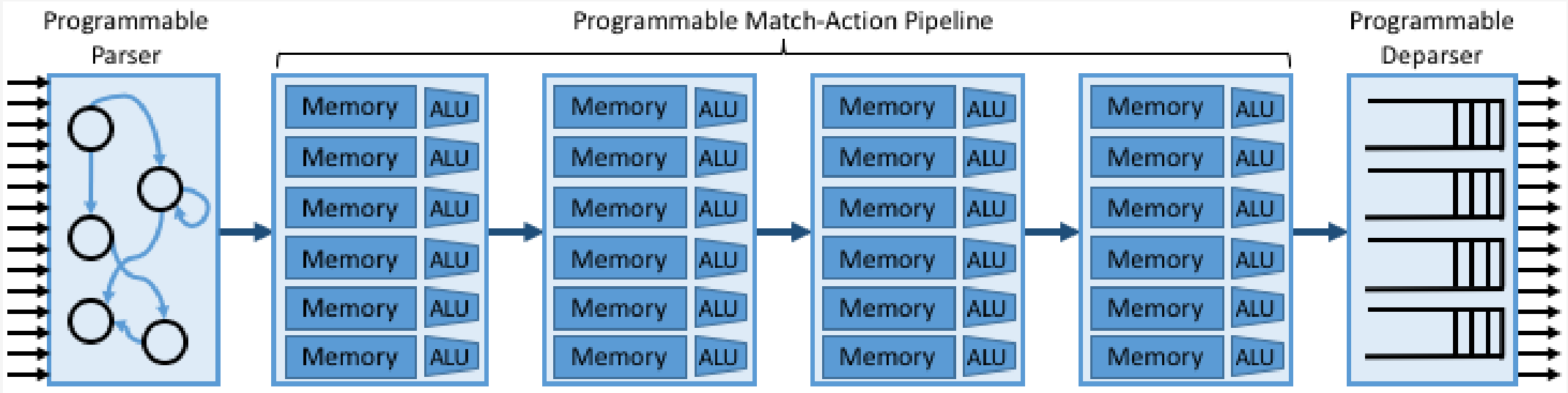
A Figura seguinte apresenta uma visão geral de alto nível do PISA, que inclui três componentes principais.

O primeiro é um parser, que é programado para definir quais campos de cabeçalho (e sua localização no pacote) devem ser ser reconhecido e acompanhado por estágios posteriores.

A segunda é uma sequência de Unidades de Unifica-Ação, cada uma das quais é programado para corresponder (e potencialmente agir de acordo com) um ou mais dos campos de cabeçalho identificados.

O Deparser reconstrói a representação over-the-wire para cada pacote a partir de todos os cabeçalhos na memória que são campos processados pelos estágios anteriores.

Protocol Independent Switching Architecture (PISA)



TCAM versus SRAM

Não é mostrada na figura uma coleção de metadados sobre os pacotes que atravessam o pipeline.

Isso inclui tanto o estado por pacote, como a porta de entrada e o carimbo de data/hora de chegada, quanto o estado do nível de fluxo calculado entre pacotes sucessivos, como contadores de switch e profundidade da fila.

Esses metadados, que possuem uma contraparte ASIC (por exemplo, um registro), estão disponíveis para estágios individuais de leitura e gravação.

Também pode ser usado pelo Unidade Match-Action, por exemplo, correspondente na porta de entrada

TCAM e caracteres curinga

Especificamente, o “CAM” em TCAM significa “Content Addressable Memory”, o que significa que a chave que a procurar numa tabela pode efetivamente ser usado como o endereço na memória que implementa a tabela. O “T” significa “Ternário”, que é uma forma técnica de dizer que a chave que você deseja procurar pode ter caracteres curinga nele (por exemplo, a chave 10*1 corresponde a 1001 e 1011). Do ponto de vista do software, a principal conclusão é que correspondências curinga são mais caras que correspondências exatas e devem ser evitadas sempre que possível

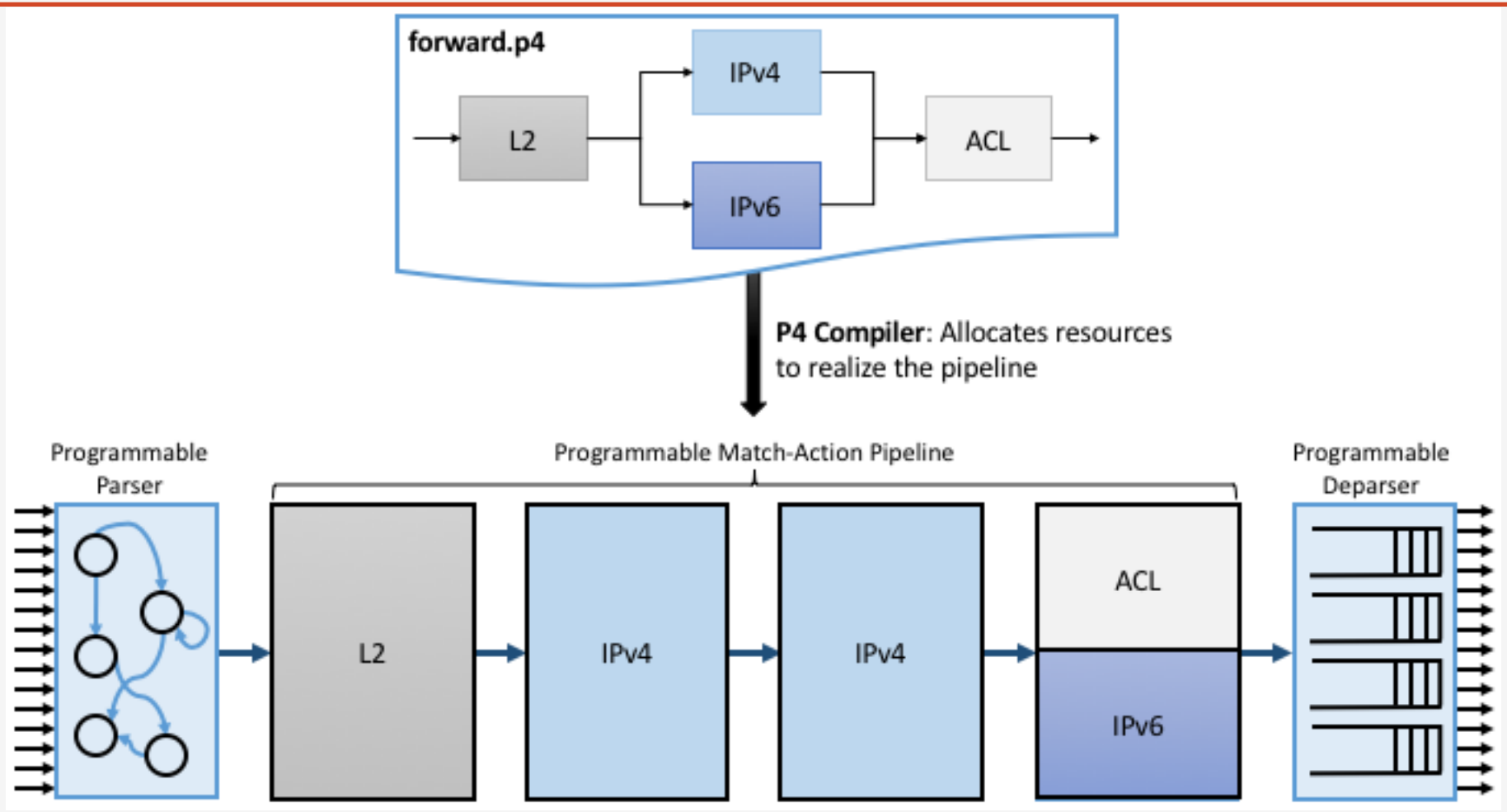
ID VRF (Encaminhamento/Expedição Verificável)

A ALU mostrada na figura implementa então a ação emparelhada com o padrão correspondente.
É possível que as acções incluam modificar campos de cabeçalho específicos (por exemplo, diminuir um TTL), enviar ou exibir tags (por exemplo, VLAN, MPLS), incrementando ou limpando vários contadores internos do switch (por exemplo, pacotes processados),e definir metadados internos/do usuário (por exemplo, o ID VRF a ser usado no tabela expedição)

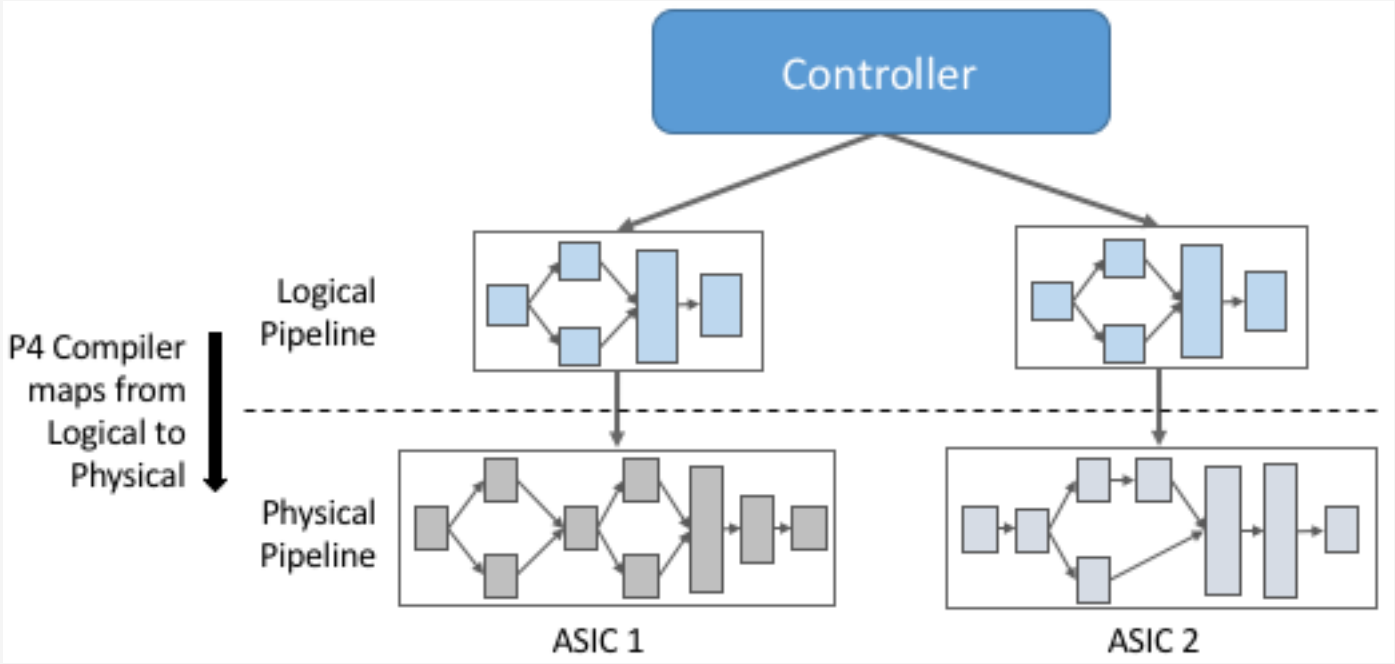
Programar diretamente o parser, as unidades de ação de correspondência e o deparser seria tedioso, semelhante a escrever código de assembly do processador

Então, em vez disso, expressamos o comportamento desejado usando uma linguagem de alto nível como P4, que depende de um compilador para gerar o programa de baixo nível equivalente.

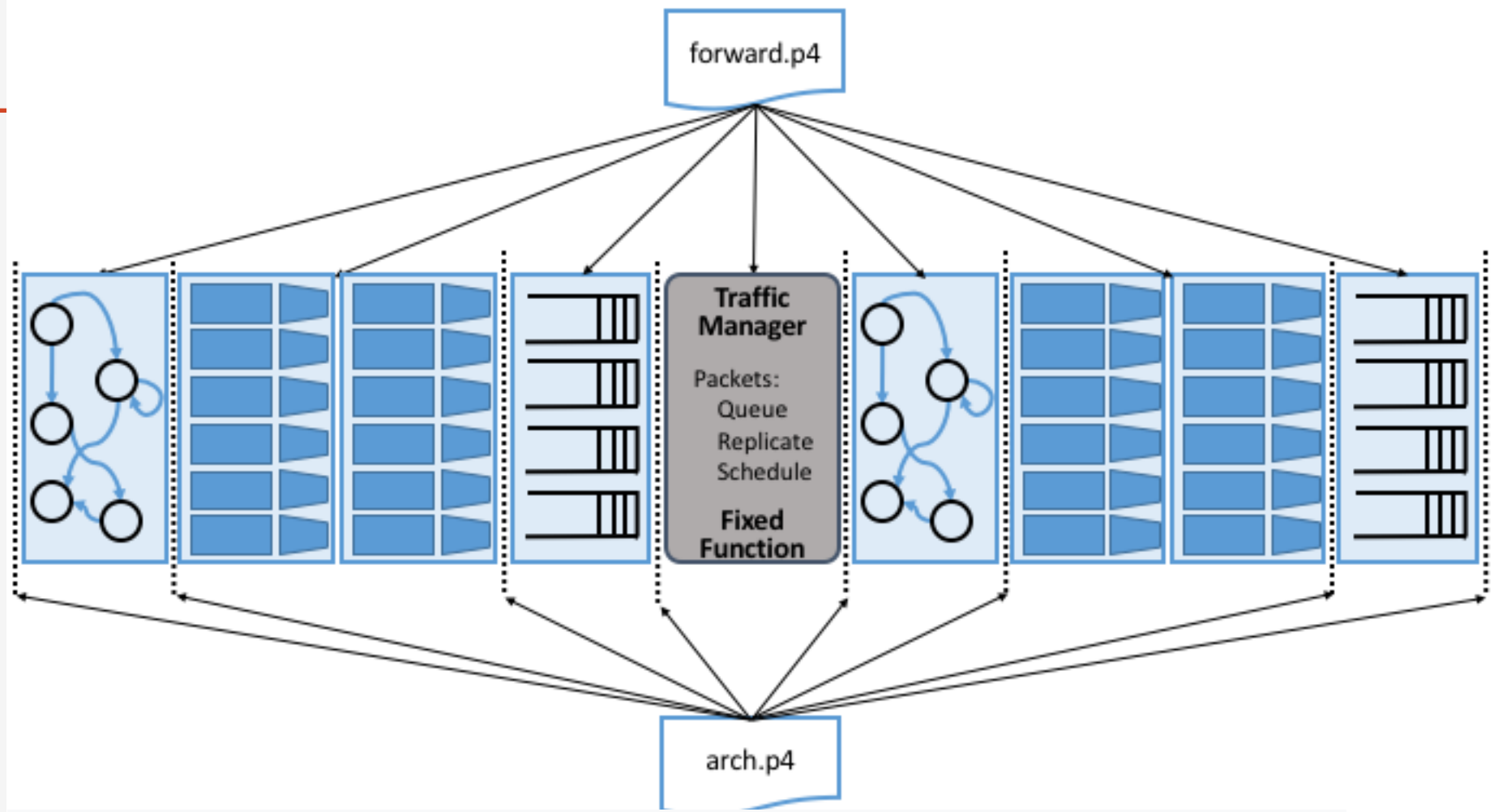
Representação do comportamento de encaminhamento desejado (conforme especificado por uma representação pictórica de um programa P4) mapeado no PISA.



Abstract Pipeline



A próxima peça do puzzle é considerar diferentes chips de comutação que implementam diferentes capacidades físicas dos pipelines. Para fazer isso, precisamos de um pipeline abstrato (canônico) que seja geral o suficiente para representar de forma justa o hardware disponível, além de uma definição de como o pipeline abstrato é mapeado no pipeline físico. Com um modelo lógico para o pipeline, seremos capazes de oferecer suporte a controladores independentes de pipeline, conforme ilustrado na Figura acima



Arquitetura P4 conhecida como Portable Switch Architecture (PSA). Inclui o arch.p4 genérico como especificação do modelo de arquitetura, mas para PSA isso seria substituído por psa.p4.Quando comparado com o PISA mais simples temos o Traffic Manager

Forward.p4

Como especificamos o pipeline lógico? Isto também é feito com um programa P4, resultando na situação mostrada na Figura 6.5.

Observe que estamos revisitando os dois programas P4 apresentados na Figura 5.1. O primeiro programa (forward.p4) define a funcionalidade que desejamos do chip de comutação disponível.

Este programa está escrito pelos programadores que desejam estabelecer o comportamento do plano de dados.

Portable Switch Architecture

O exemplo mostrado na Figura 6.5 é chamado de Portable Switch Architecture (PSA). Pretende-se fornecer aos programadores P4 a possibilidade de implementar programas de expedição tipo forward.p4 com uma máquina de destino abstrata, análoga a uma Java Virtual Machine. O objetivo é o mesmo do Java: suportar uma operação de gravação única, execução em qualquer lugar paradigma de programação. (Observe que a Figura 6.5 inclui o arch.p4 genérico como especificação do modelo de arquitetura, mas na prática o modelo de arquitetura seria específico do PSA, como psa.p4.)

Arch.p4

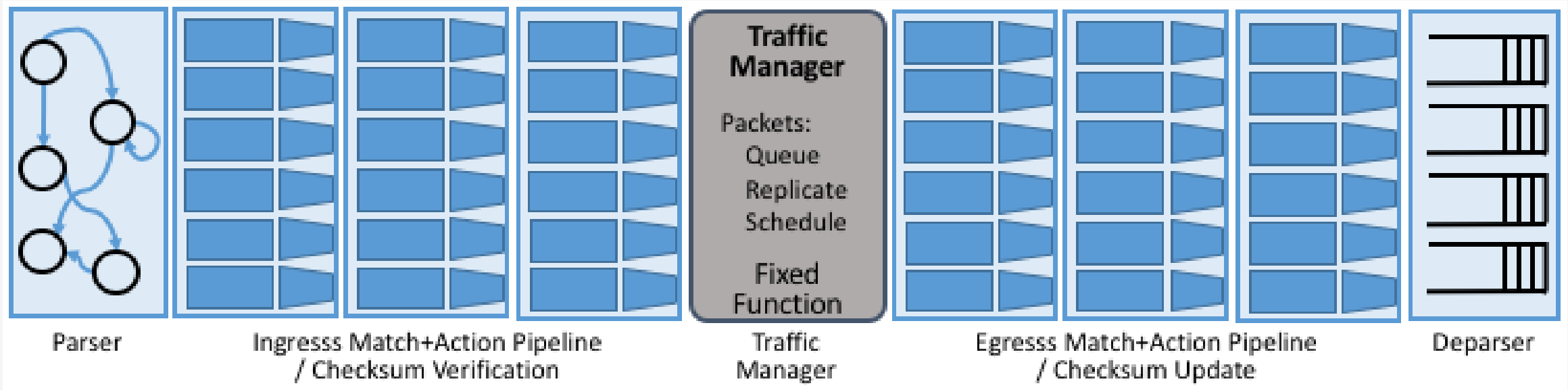
O que é que o arch.p4 realmente define?

1. Conforme implícito na Figura 6.5, ela define as assinaturas da interface entre blocos em termos de sinais de E/S (pense em "parâmetros de função e tipo de retorno"). O objetivo de um programador P4 é fornecer um implementação para cada bloco programável P4 que recebe os sinais de entrada fornecidos, como porta de entrada onde um pacote foi recebido e grava nos sinais de saída para influenciar o comportamento dos blocos seguintes (por exemplo, a fila/porta de saída para onde um pacote deve ser direcionado).

Arch.p4

2. Declarações de tipo para externos, que podem ser vistas como serviços adicionais de função fixa exposta pelo alvo e que pode ser invocado por um programador P4. Exemplos de tais externos são soma de seleção e unidades de computação hash, contadores de pacotes ou bytes, cifras para criptografar/decifrar a carga útil do pacote, e assim por diante. A implementação de tais externos não é especificada em P4 pela arquitetura, mas a sua nterface é.

3. Extensões para os principais tipos de linguagem P4, incluindo tipos de correspondências alternativas (por exemplo, intervalo e lpm descritos mais adiante)



O Model V1 usado na prática para abstrair os detalhes de diferentes pipelines de encaminhamento físico. Os programadores escrevem P4 para este modelo arquitetônico abstrato.

Utilização do V1MODEL

```
include <core.p4>

#include <v1model.p4>

/* Headers */

struct metadata { ... }

struct headers {

ethernet_t ethernet;

ipv4_t ipv4;

}
```

(continua da próxima página)

Parser

```
/* Parser */

parser MyParser(

packet_in packet,

out headers hdr,

inout metadata meta,

inout standard_metadata_t smeta) {

...}
```

Checksum Verification and Ingress Processing

```
/* Checksum Verification */
control MyVerifyChecksum(
in headers, hdr,
inout metadata meta) {
...}
/* Ingress Processing */
control MyIngress(
inout headers hdr,
inout metadata meta,
inout standard_metadata_t smeta) {
...}
```

Egress Processing and Checksum Update

```
/* Egress Processing */
control MyEgress(
inout headers hdr,
inout metadata meta,
inout standard_metadata_t smeta) {
...
}
/* Checksum Update */
control MyComputeChecksum(
inout headers, hdr,
inout metadata meta) {
...
}
```

```
/*************************************************
 C H E C K S U M C O M P U TATION
 *************************************************/
control MyComputeChecksum(inout headers hdr, inout metadata meta)
{apply {update_checksum( hdr.ipv4.isValid(),
{ hdr.ipv4.version,
hdr.ipv4.ihl,
hdr.ipv4.diffserv,
hdr.ipv4.totalLen,
hdr.ipv4.identification,
hdr.ipv4.flags,
hdr.ipv4.fragOffset,
hdr.ipv4.ttl,
hdr.ipv4.protocol,
hdr.ipv4.srcAddr,
hdr.ipv4.dstAddr },
hdr.ipv4.hdrChecksum,
HashAlgorithm.csum16); }  }
```

Deparser

O Deparser é normalmente é muito simples. Tendo potencialmente alterado vários campos do cabeçalho durante o processamento do pacote, temos agora que colocar o cabeçalho atualizado.

Somente cabeçalhos comercializados como válidos serão re-serializado nesse pacote. Não há necessidade dizer nada sobre a a carga útil que é incluída na mensagem de saída.

Os detalhes de como os pacotes são emitidos são especificados pela arquitetura.

Por exemplo, a TNA suporta truncamento da carga útil com base na configuração de um valor de metadados especial consumido pelo deparser.

Deparser

```
/*************************************************************
************* D E P A R S E R ***********************
*************************************************************/
control MyDeparser(
packet_out packet,
in headers hdr) {
apply {
packet.emit(hdr.ethernet);
packet.emit(hdr.ipv4);
   }
}
```

V1 SWITCH MODEL

O programa P4 deve definir o comportamento do switch como um todo, que é dado pelo pacote V1Switch mostrado abaixo. O conjunto de elementos neste pacote é definido por v1model.p4 e consiste em referências a todas as outras rotinas definidas anteriormente.

O exemplo é minimalista mas serve para ilustrar as ideias essenciais de um programa P4.

O que está oculto neste exemplo é a interface usada pelo plano de controle para injetar dados na tabela de roteamento; tabela ipv4_lpm define a tabela, mas não a preenche com valores.

A forma como plano de controle coloca valores na tabela será apresentado com o P4Runtime aula 5.

Switch Definition

```
/*************************************************************

S W I T C H

*************************************************************/

V1Switch(

MyParser(),

MyVerifyChecksum(),

MyIngress(),

MyEgress(),

MyComputeChecksum(),

MyDeparser()

) main;
```

Pipelines de Função Fixa

Voltamos agora aos pipelines de encaminhamento de função fixa, com o objetivo de colocá-los nos ecossistemas maiores.

Tendo em mente que os chips de comutação de função fixa ainda dominam o mercado, não pretendemos subestimar o seu valor ou o papel que sem dúvida continuarão a desempenhar.

Mas removem um grau de liberdade – a capacidade de reprogramar o plano de dados – o que ajuda a destacar a relação entre todas as partes móveis apresentadas neste capítulo

OF-DPA

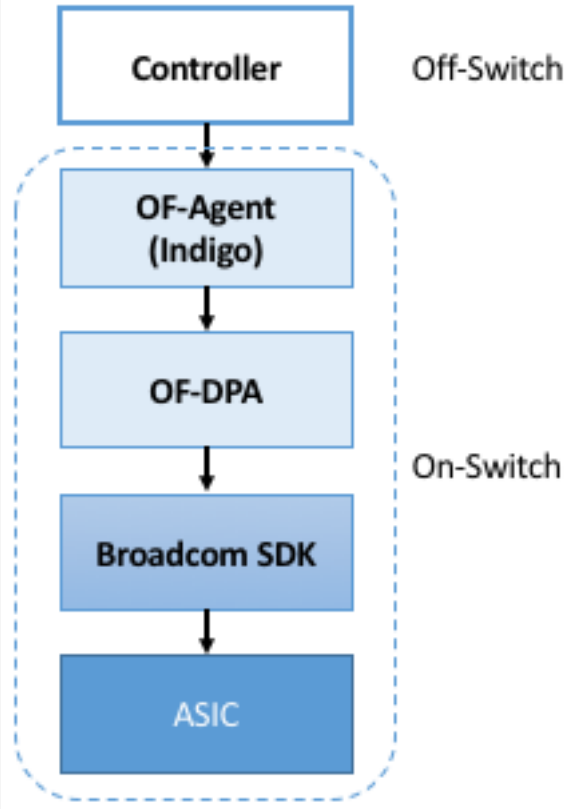
Começamos com um exemplo concreto: a abstração de hardware OpenFlow — Data Plane Abstraction (OF-DPA) que é uma camada que a Broadcom fornece para seus chips de comutação.

OF-DPA define uma API que pode ser usada para instalar regras de fluxo no Broadcom ASIC subjacente.

Tecnicamente, um agente OpenFlow fica acima do OF-DPA (ele implementa os aspectos over-the-wire do protocolo OpenFlow) e o SDK Broadcom fica abaixo do OF-DPA (implementa a interface proprietária que conhece os detalhes do chip de baixo nível)

O OF-DPA é a camada que fornece uma representação abstrata do pipeline de encaminhamento fixo do Tomahawk ASIC (ver a figura seguinte)

OpenFlow—Data Plane Abstraction (OF-DPA)



Pilha de de Software do pipeline de expedição de função fixa Tomahawk (OF-Agent e OF-DPA são software aberto e o Broadcom SDK é proprietário)

OF-PDA e SAI

Mesmo sem os detalhes da Figura, pode-se reconhecer tabelas para vários protocolos.

Para os nossos propósitos, o que é instrutivo é ver como o OF-DPA mapeia seu pipeline nos seus homólogos programáveis.

No caso programável, somente quando se adiciona um programa como switch.p4 é que se obtém algo aproximadamente equivalente ao OF-DPA.

O v1model.p4 define os estágios disponíveis (blocos de controle).

Mas só depois de adicionar switch.p4 é que você terá a funcionalidade executada nesses estágios.

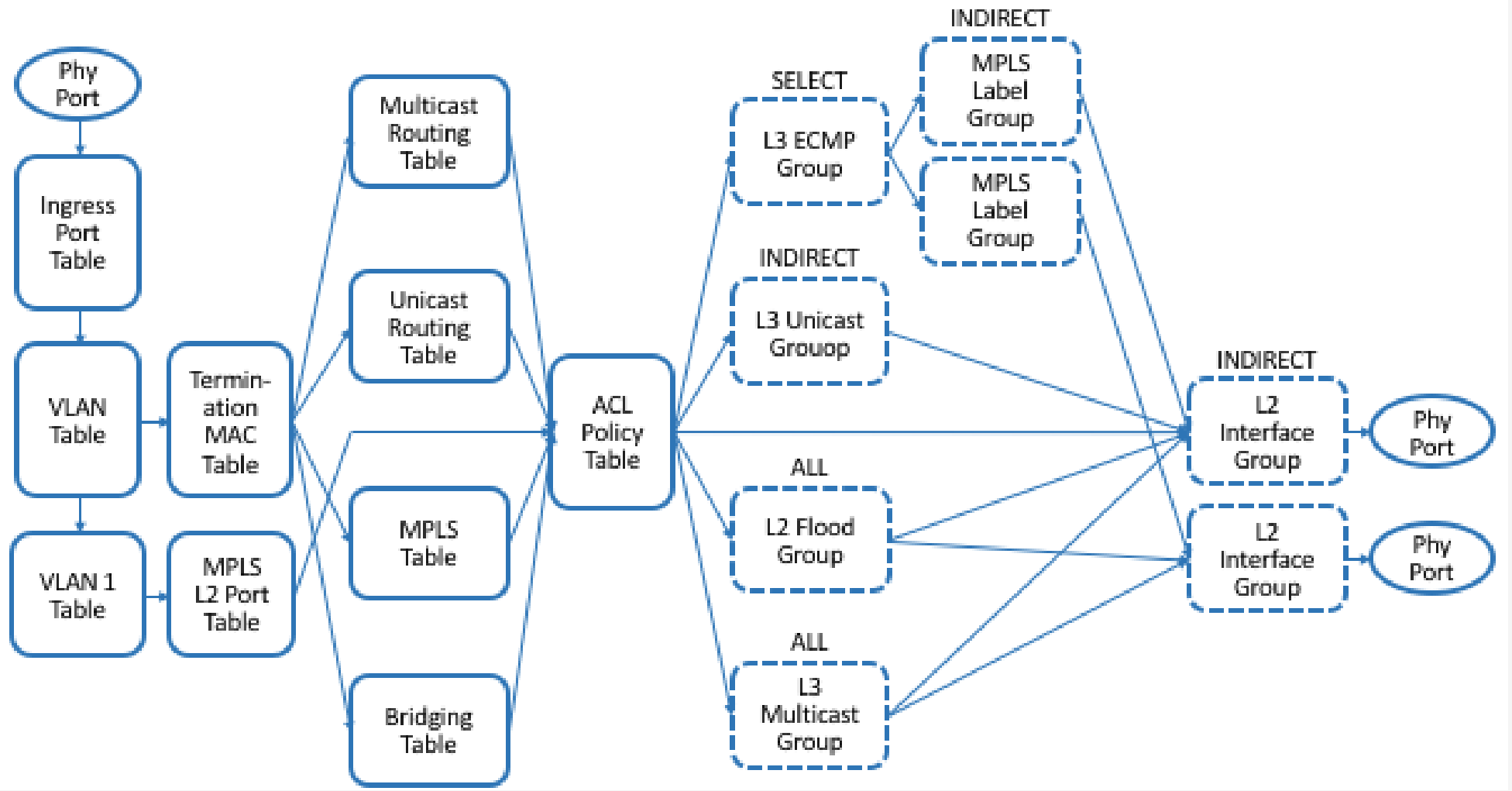
OF-PDA e SAI

Com esta relação em mente, podemos querer incorporar comutadores programáveis e de função fixa em uma única rede e executando uma pilha de software SDN comum.

Isso pode ser feito ocultando ambos tipos de chips por trás do modelo de arquitetura v1model.p4 (ou similar) e permitindo que o compilador P4 produza o código de back-end compreendido por seus respectivos SDKs.

Este cenário não funciona para um programa P4 arbitrário que deseja fazer algo que o chip Tomahawk não suporte, mas funcionará para padrões de comportamento padrão L2/L3.

Logical fixed-function pipeline defined by OF-DPA



Compactação

Esta discussão sobre pipelines lógicos e sua relação com os programas P4 é subtil e vale a pena reafirmá-la.

Por um lado, há um valor óbvio em ter uma representação abstrata de um pipeline físico, como apresentado como um conceito geral na Figura 6.4.

Quando usado desta forma, um pipeline lógico é um exemplo duma ideia testada e comprovada de introduzir uma camada de abstração de hardware.

No nosso caso, ajuda no plano de controle da portabilidade. OF-DPA é um exemplo específico de camada de abstração de hardware para funções fixas da Broadcom, trocando chips.

SAI

Assim como vimos modelos de arquitetura definidos pelo fornecedor e pela comunidade (TNA e V1Model, respectivamente), também existem pipelines lógicos de função fixa definidos pelo fornecedor e pela comunidade.

OF-DPA é o primeiro, e a Switch Abstraction Interface (SAI) é um exemplo do último.

Porque o SAI tem que funcionar em uma variedade de switches – e pipelines de encaminhamento – ele necessariamente se concentra no subconjunto de funcionalidadestodos os fornecedores podem concordar com o mínimo denominador comum, por assim dizer.

SAI inclui uma interface de configuração e uma interface de controle, sendo esta última a mais relevantepara esta seção porque ela abstrai o pipeline de encaminhamento.

Por outro lado, há pouco valor em olharem mais um pipeline de encaminhamento, portanto, encaminhamos o leitor interessado à especificação SAI para obter mais detalhes

Comparação

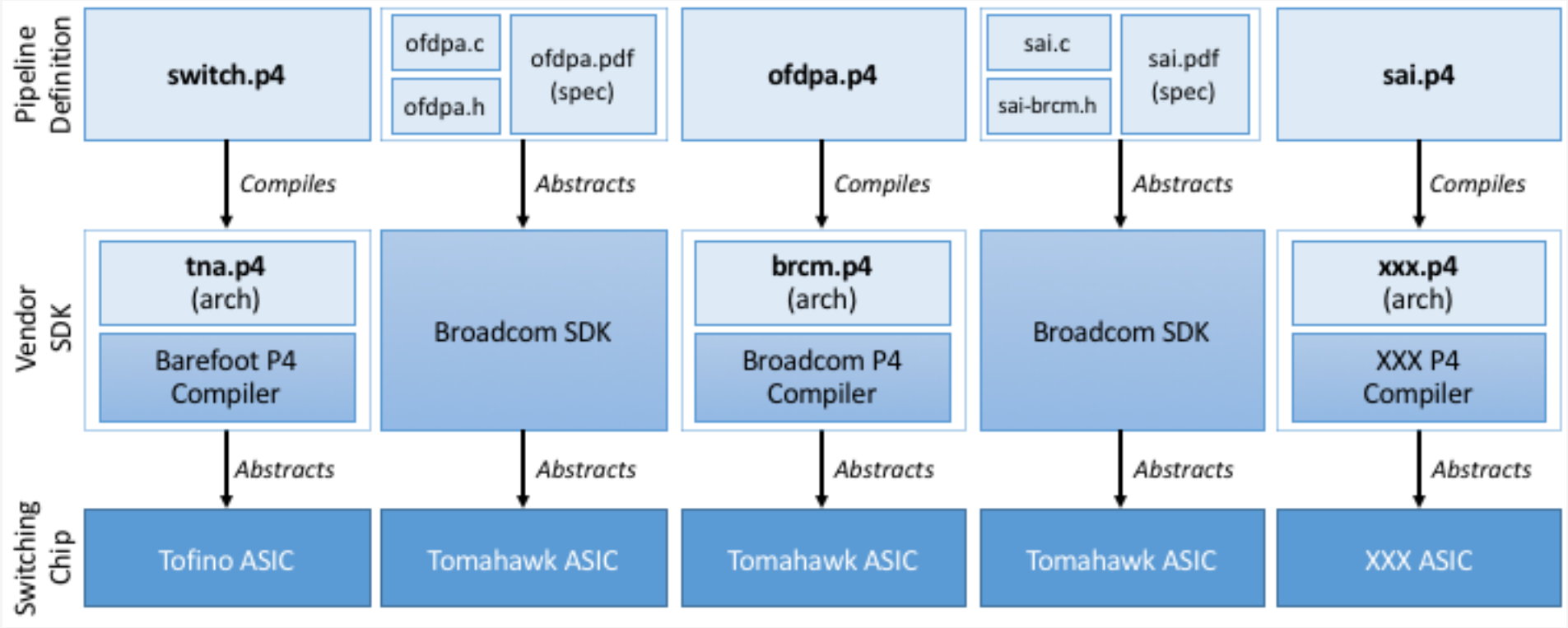
Cada exemplo na Figura seguinte consiste em três camadas: um chip de comutação ASIC, um SDK específico do fornecedor para programação do ASIC e uma definição do pipeline de expedição.

Ao fornecer uma interface programática, os SDKs na camada intermediária abstraem efetivamente o hardware subjacente. Eles são convencionais (por exemplo, o Broadcom SDK mostrado no segundo e quarto exemplos) ou, como acabamos de salientar, correspondem logicamente para um modelo de arquitetura P4 emparelhado com um compilador P4 específico para ASIC.

A camada superior em todos os cinco exemplos define um pipeline lógico que pode posteriormente ser controlado usando uma interface de controle como OpenFlow ou P4Runtime (não mostrado). Os cinco exemplos diferem dependendo se o pipeline é definido por um programa P4 ou através de algum outro meio (por exemplo, a especificação OF-DPA).

Observe que apenas as configurações com um pipeline lógico definido por P4 no topo da pilha (ou seja, primeiro, terceiro, quinto exemplo) pode ser controlado usando P4Runtime.
Isto ocorre pela razão pragmática de que o P4Runtime interface é gerada automaticamente a partir deste programa P4 usando as ferramentas descritas no próximo capítulo

Comparação



Cinco exemplos de pilhas Pipeline/SDK/ASIC. As duas pilhas mais à esquerda, mais a quarta pilha, existem hoje; a pilha intermediária é hipotética; e a pilha mais à direita é um trabalho em andamento