Advanced Computer Networking (ACN)

IN2097 – WiSe 2019-2020

Prof. Dr.-Ing. Georg Carle

Sebastian Gallenmüller, Max Helm, Benedikt Jaeger, Patrick Sattler, Johannes Zirngibl

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich
Chapter 6: Software Defined Networking

Introduction

OpenFlow
  Introduction
  Core concepts
  Example

NFV

P4
  Motivation
  P4 targets
  P4 Core
  P4 example: IPv4 router
  Active area of research

Acknowledgements

Bibliography
Chapter 6: Software Defined Networking

Introduction

OpenFlow

NFV

P4

Acknowledgements

Bibliography
Introduction
Management Plane, Control Plane, and Data Plane

Forwarding table updates and lookups are managed by the Control Plane, which in turn is managed by the Management Plane. The Data Plane handles per-packet processing of incoming and outgoing frames.
Management Plane:

- Allows access for administrators to the configuration of the other planes
- Tuning the parameters of the underlying algorithms

Control Plane:

- Has rules about which frames should go where
- Creates lookup tables from those rules
- Provides lookup tables for the data plane

Data Plane (also called Forwarding Plane):

- Uses lookup tables provided by the control plane
- Actually touches / forwards frames
Management Plane:

- Allows access for administrators to the configuration of the other planes
- Tuning the parameters of the underlying algorithms

Control Plane:

- Has rules about which frames should go where
- Creates lookup tables from those rules
- Provides lookup tables for the data plane

Data Plane (also called Forwarding Plane):

- Uses lookup tables provided by the control plane
- Actually touches / forwards frames

Example: Routing

- **Management Plane:** configuring link costs
- **Control Plane:** creating a routing table
- **Data Plane:** forwarding of frames according to routing table
Traditional architectures consist of three planes: management plane, control plane, and data plane.

A plane is a group of algorithms and network protocols.

These algorithms:

- process different kind of traffic,
- have different performance requirements,
- are designed using different methodologies,
- are implemented using different programming languages,
- run on different hardware.
Problems with the standard approach:

- depends heavily on hardware platform and chip vendor,
- depends on vendor implementation,
- access to the source code of the implementation is limited,
- updates to protocols are slow (see adoption of IPv6),
- are often changing from one vendor to another.
What is SDN?

- **Software Defined Networking**
- Provides a layer of abstraction from the physical network

How does it do that?

- Historically, devices include both, the *control plane* and the *data plane*
- SDN has one central *control plane*, which manages all the *data planes* of all the switches
- In your datacenter, you know your traffic flows. It is your datacenter!
- How can you optimize your traffic flows?
  - VM1 to VM3 should flow via W → Z → Y
  - VM2 to VM4 should flow via W → X → Y
VM1 to VM3: W -> Z -> Y
VM2 to VM4: W -> X -> Y

Data plane

Control plane
Introduction
A more formal definition

Two requirements:

• A network in which the control plane is separate from the forwarding plane
• A single control plane controls several forwarding devices

Both have to be met
Introduction
SDN Benefits

Why the term “Software Defined”?

• The control plane is just software

Abstraction:

• No distributed state, one central view of the network (common model: "one big switch")
• No individual configuration, one centrally managed control plane
• Important: View centralized, controller itself may also be distributed

Gain:

• Complex, distributed protocols such as the Spanning Tree Protocol are no longer necessary
• Simpler algorithms utilizing the central view (e.g., Dijkstra)
• Less complexity in the control plane
Chapter 6: Software Defined Networking

Introduction

OpenFlow
  - Introduction
  - Core concepts
  - Example

NFV

P4

Acknowledgements

Bibliography
Introduction
What is OpenFlow?

- OpenFlow is a protocol configuring the forwarding plane
  - runs on top of TCP/SSL
  - Protocol spoken between control plane and forwarding plane
- Standardized by the Open Networking Foundation (ONF)
- Version 1.0 was released in 2009 [1]
- Latest version 1.6 from 2016 [2]
OpenFlow is based on the match+action principle

<table>
<thead>
<tr>
<th>Match</th>
<th>Actions</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Forward (one or more ports)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Encapsulate and send to controller</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Header rewrite</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Push/pop MPLS label / VLAN tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Queues + bitrate limiter (bit/s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Etc..</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>VLAN ID</th>
<th>VLAN pcp</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP ToS</th>
<th>IP Prot</th>
<th>L4 sport</th>
<th>L4 dport</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: (Source: cleanslate.stanford.edu)
## Core concepts

### Tables examples

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC Src</th>
<th>MAC Dst</th>
<th>Eth Type</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP TOS</th>
<th>IP Prot</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>L4 Src</th>
<th>L4 Dst</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>Port 5</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>00:1f:...</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 2: Ethernet switch*

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC Src</th>
<th>MAC Dst</th>
<th>Eth Type</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP TOS</th>
<th>IP Prot</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>L4 Src</th>
<th>L4 Dst</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>Port 5</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>1.2.0.0/16</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 3: Router*

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC Src</th>
<th>MAC Dst</th>
<th>Eth Type</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP TOS</th>
<th>IP Prot</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>L4 Src</th>
<th>L4 Dst</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>22</td>
</tr>
</tbody>
</table>

*Figure 4: Firewall*
Core concepts
Remark about the term switch

Traditional classification

- **Switch:**
  - Works on layer 2
  - Simple forwarding of packets
- **Router:**
  - Works on layer 3
  - Finding out where to route packets (LPM)

In the context of SDN every "box" is considered a switch

- Clear distinction (e.g. switch, router) no longer possible as functionality is determined by software
- These boxes/switches can even be used as firewall, tunnel gateways
This document describes the requirements of an OpenFlow Logical Switch. Additional information describing OpenFlow and Software Defined Networking is available on the Open Networking Foundation website (https://www.opennetworking.org/). This specification covers the components and the basic functions of the switch, and the OpenFlow switch protocol to manage an OpenFlow switch from a remote OpenFlow controller.

Figure 1: Main components of an OpenFlow switch.

2 Switch Components

An OpenFlow Logical Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and one or more OpenFlow channels to an external controller (Figure 1). The switch communicates with the controller and the controller manages the switch via the OpenFlow switch protocol.

Using the OpenFlow switch protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and proactively. Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets (see 5.2).

Matching starts at the first flow table and may continue to additional flow tables of the pipeline (see 5.1). Flow entries match packets in priority order, with the first matching entry in each table being used (see 5.3). If a matching entry is found, the instructions associated with the specific flow entry are executed (see 5.5). If no match is found in a flow table, the outcome depends on configuration of the

Figure 5: OpenFlow switch (source: OpenFlow Switch Specification, ONF)
**Example**

Rules installed on the switch

```bash
ovs-ofctl add-flow ctl-if priority=0,actions=controller
```
Example

Rules installed on the switch

```
ovs-ofctl add-flow ctl-if priority=0,actions=controller
```

- `add-flow`: "OpenFlow rule" (Not a regular network flow!)
- `ctl-if`: Destination for this OpenFlow flow
- `actions=controller`: Send packets matching this rule to the controller
- `priority=0`: 0 is lowest priority
Example

Rules installed on the switch

`ovs-ofctl add-flow ctl-if priority=0,actions=controller`

- Packet sent from Client 1 to Client 2
**Example**

Rules installed on the switch

```
ovs-ofctl add-flow ctl-if priority=0,actions=controller
```

- Packet sent from Client 1 to Client 2
- Packet matches against rule → Controller
Rules installed on the switch

```bash
ovs-ofctl add-flow ctl-if priority=0,actions=controller
```

- Packet sent from Client 1 to Client 2
- Packet matches against rule → Controller
- Controller instructs switch to send packet to destination
**Example**

**Rules installed on the switch**

```shell
ovs-ofctl add-flow ctl-if priority=0,actions=controller

ovs-ofctl add-flow ctl-if dl_type=0x0800,nw_dst=10.0.0.2, priority=10000,actions=output:2
```
Rules installed on the switch

```bash
ovs-ofctl add-flow ctl-if priority=0,actions=controller
ovs-ofctl add-flow ctl-if dl_type=0x0800,nw_dst=10.0.0.2, priority=10000,actions=output:2
```

- Controller can also install rule on switch to make forwarding more efficient
- IPv4 packets (matching ethertype 0x0800 destination address 10.0.0.2) from Client 1 get directly forwarded to Client 2
OpenFlow in the wild

- OpenFlow is not SDN
- OpenFlow with its standardized interface enables SDN deployment
- Very successful in software switches (Open vSwitch)
  - OvS is used for switching between VMs
  - Can also be used to connect physical machines
- There are hardware switches with OpenFlow support
  - Did not make traditional switches obsolescent as initially expected
  - Still many proprietary switches today

OpenFlow

- Allows programming the control plane
- Allows modifications in the data plane
- Standard supports only a limited number of protocols
  - To introduce new protocols the standard must be updated
  - Switches must be upgraded to handle the new standard
Chapter 6: Software Defined Networking

Introduction

OpenFlow

NFV

P4

Acknowledgements

Bibliography
NFV
Network Function Virtualization (NFV)

• Defined by ETSI (European Telecommunications Standards Institute)
• Telco-driven approach for networks initiated in 2012
• Definition of NFV according to ETSI: NFV is a concept "leveraging standard IT virtualisation technology to consolidate many network equipment types onto industry standard high volume servers, switches and storage, which could be located in Datacentres, Network Nodes and in the end user premises."
Figure 6: from https://www.slideshare.net/nearyd/nfv-for-beginners
Some Terminology

- **(V)NF**: *(Virtualized)* **Network Function**, *(virtualized)* building block performing a network task
- **NFC**: **Network Function Chaining**, putting together several network functions to create more complex packet processing chains

*Figure 7: Example of a chain of Virtual Network Functions*
NFV vs. SDN

• "SDN and NFV are complementary but increasingly co-dependent" [3]
• SDN: dynamically control the network
• NFV: manage and orchestrate the virtualization of resources for the provisioning of network functions and their composition into higher-layer network services
NFV architectures I

Traditional approach
- One VM per NF
- Communication between NFs via virtual switch
  + Strong isolation between NFs
  + Uses traditional OS sockets
  - High load on virtual switch

Non-virtualized NFC
- Entire NFC running directly on host system
- Communication between NFs via NF framework (e.g. DPDK), initial entry and last exit via virtual switch
  + No costs for virtual switch
  - NFs need to be rewritten to use NF framework

Figure 8: Traditional VM-based NFC

Figure 9: Non-virtualized framework-based NFs
Hybrid solution: virtualized NFC

- One VM for entire NFC
- Communication between NFs via NF framework, initial entry and last exit via virtual switch
  + Lower load on virtual switch
  - NFs need to be rewritten to use NF framework

Figure 10: Virtualized framework-based NFs
NFV

Performance of NFs

- Tradeoff between isolation and performance requirements:
  - Isolation (high to low): Virtual machines, container, no virtualization
  - Performance (low to high): Virtual machines, container, no virtualization
Performance virtual switching solutions [4]

- Investigated 4 different setups involving physical/virtual pNICs/vNICs
- CPU: Intel Xeon E3-1230 V2 CPU (3.3 GHz, base clock)
- pNIC: 10 Gbit/s Intel X540
- SW: GRML Linux kernel v3.7, Open vSwitch v2.0, DPDK vSwitch v0.1
- Hypervisor: qemu-kvm 1.1.2
- Worst case measurement scenario: minimum-sized packets 64 B (14.88 Mpps @ 10 Gbit/s)

Figure 11: Investigated test setups
### Figure 12: Single Core Data Plane Performance Comparison

<table>
<thead>
<tr>
<th>Application</th>
<th>pNIC</th>
<th>vNIC</th>
<th>vNIC to pNIC</th>
<th>vNIC to vNIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux bridge</td>
<td>1.11</td>
<td>0.74</td>
<td>0.20</td>
<td>0.19</td>
</tr>
<tr>
<td>IP forwarding</td>
<td>1.58</td>
<td>0.78</td>
<td>0.19</td>
<td>0.16</td>
</tr>
<tr>
<td>Open vSwitch (OvS)</td>
<td>1.88</td>
<td>0.85</td>
<td>0.30</td>
<td>0.27</td>
</tr>
<tr>
<td>DPDK vSwitch</td>
<td>13.51</td>
<td>2.45</td>
<td>1.10</td>
<td>1.00</td>
</tr>
</tbody>
</table>

- DPDK vSwitch is the DPDK-accelerated version of OvS
- Network IO for VMs is quite expensive
Conclusion

<table>
<thead>
<tr>
<th></th>
<th>Traditional approach</th>
<th>Virtualized NFC</th>
<th>Non-virtualized NFC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>+</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td><strong>Isolation</strong></td>
<td>+++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td><strong>Chaining interface</strong></td>
<td>OS sockets</td>
<td>Framework-based</td>
<td>Framework-based</td>
</tr>
</tbody>
</table>

*Figure 13: Comparison between different NFC architectures*

Possible reasons for choosing different architectures

- Performance requirements
- Integration of legacy NF supporting only socket interface
- Integration of NFs from different vendors
- Stronger isolation requirements for untrusted customer code
Chapter 6: Software Defined Networking

Introduction

OpenFlow

NFV

P4
  
  Motivation
  P4 targets
  P4 Core
  P4 example: IPv4 router
  Active area of research

Acknowledgements

Bibliography
Motivation
OpenFlow versus P4

OpenFlow
- OpenFlow allows programmability on the control plane
- OpenFlow offers a standardized interface to configure the data plane
- OpenFlow only supports protocols known by the hardware or software used on the data plane
Motivation
OpenFlow versus P4

OpenFlow

- OpenFlow allows programmability on the control plane
- OpenFlow offers a standardized interface to configure the data plane
- OpenFlow only supports protocols known by the hardware or software used on the data plane
- Introducing a new protocol (e.g., NewP) fails without support from the data plane
OpenFlow

- OpenFlow allows programmability on the control plane
- OpenFlow offers a standardized interface to configure the data plane
- OpenFlow only supports protocols known by the hardware or software used on the data plane

P4 (Programming Protocol-Independent Packet Processors)

- P4 is a domain specific programming language to program data plane devices
- P4 allows programming switches to support entirely new protocols (e.g., NewP)
Motivation

OpenFlow versus P4

OpenFlow

• OpenFlow allows programmability on the control plane
• OpenFlow offers a standardized interface to configure the data plane
• OpenFlow only supports protocols known by the hardware or software used on the data plane

P4 (Programming Protocol-Independent Packet Processors)

• P4 is a domain specific programming language to program data plane devices
• P4 allows programming switches to support entirely new protocols (e.g., NewP)

OpenFlow vs. P4

• P4 is not a successor or a replacement of OpenFlow
• OpenFlow and P4 solve specific tasks on separate planes
• P4 can be used to implement a OpenFlow-capable applications for switches
Motivation
Data plane programmability

Goal: program your own data plane!

Benefits:

- **Control and customization**: make the device behave exactly as you want, operators can hide internal protocols
- **Reliability**: include only the features you need
- **Efficiency**: reduce energy consumption and expand scale by doing only what you need
- **Update**: Add new features when you want
- **Telemetry**: See inside the data plane
- **Exclusivity**: Program your own features without the need for involving a chip vendor
- **Rapid Prototyping**: enables fast deployment of protocols for prototyping
- **Fast Development Cycles**: enables software upgrades for protocols
Motivation
Data plane programmability

Goal: program your own data plane!

Benefits:

- **Control and customization**: make the device behave exactly as you want, operators can hide internal protocols
- **Reliability**: include only the features you need
- **Efficiency**: reduce energy consumption and expand scale by doing only what you need
- **Update**: Add new features when you want
- **Telemetry**: See inside the data plane
- **Exclusivity**: Program your own features without the need for involving a chip vendor
- **Rapid Prototyping**: enables fast deployment of protocols for prototyping
- **Fast Development Cycles**: enables software upgrades for protocols

Challenges:

- **Performance**: data planes need to process millions of packets per second
- **Flexibility**: Enable the implementation of various protocols
- **Hardware independence**: keep the description high-level enough
An open source language allowing the specification of packet processing logic

Based on a Match+Action forwarding model

Multiple platforms supported:

- Software-based solution (e.g., using DPDK)
- NPUs - Network Processor Units
- FPGAs - Field Programmable Gate Arrays
- P4-specific ASICs
P4 targets

Software targets

p4c/bmv2

- "official" P4 reference implementation developed by p4.org
- used for teaching, testing, trying out new features
- no specific hardware required (mininet)
- slow, not optimized for performance

T₄P₄S (called tapas)

- compiles P4 for DPDK
- requires DPDK-compatible hardware
- decent performance (>10 Gbit/s)
P4 targets
Network Processor Unit (NPU)

Netronome Agilio SmartNIC

- purpose-built processor for packet processing
- specialized hardware accelerators (e.g. hashing, look up)
- highly parallelized architecture (>100 cores)
- supports several programming languages C, P4, eBPF
- up to $2 \times 100$ Gbit/s interfaces per network card

NFP-4000 architecture [source: netronome.com]

Netronome SmartNIC [source: colfaxdirect.com]
P4 targets
Field Programmable Gate Array (FPGA)

NetFPGA

- fully programmable NIC (down to the physical layer)
- utilizing hardware description languages such as Verilog or VHDL
- Xilinx Virtex 7 FPGA
- up to $4 \times 10$ Gbit/s interfaces (via SFP+ transceivers)

NetFPGA Sume [source: github.com/NetFPGA]
Barefoot Tofino

- Tofino ASIC: specifically designed switching ASIC with native P4 support
- capable of up to 6.5 Tbit/s throughput (unidirectional)
- for comparison: peak traffic at biggest Internet exchange DE-CIX in Frankfurt was 6.74 Tbit/s in 2018
- up to $64 \times 100$ Gbit/s interfaces (via QSFP28 transceivers)
## P4 targets

### Target comparison

<table>
<thead>
<tr>
<th></th>
<th>SW</th>
<th>NPU</th>
<th>FPGA</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td><strong>Flexibility</strong></td>
<td>+++</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td><strong>Ease of use</strong></td>
<td>+++</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Costs</strong></td>
<td>0 €</td>
<td>&gt; 500 €</td>
<td>&gt; 1000 €</td>
<td>&gt; 10 000 €</td>
</tr>
</tbody>
</table>

Did P4 achieve its goals?
- **Performance:** data planes need to process millions of packets per second: **accomplished ✓**
- **Flexibility:** Enable the implementation of various protocols: **accomplished ✓**
- **Hardware independence:** keep the description high-level enough: **development ongoing ...**
- Basic P4 functionality can be realized on any target
- Every target offers different additional capabilities not programmed in P4 (e.g. multicast support)
- These additional functionalities make P4 programs hardware dependent

Chapter 6: Software Defined Networking – P4
### P4 targets

#### Target comparison

<table>
<thead>
<tr>
<th></th>
<th>SW</th>
<th>NPU</th>
<th>FPGA</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td>Flexibility</td>
<td>+++</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Ease of use</td>
<td>+++</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Costs</td>
<td>0 €</td>
<td>&gt; 500 €</td>
<td>&gt; 1000 €</td>
<td>&gt; 10 000 €</td>
</tr>
</tbody>
</table>

Did P4 achieve its goals?

- **Performance**: data planes need to process millions of packets per second → accomplished ✓
- **Flexibility**: Enable the implementation of various protocols → accomplished ✓
- **Hardware independence**: keep the description high-level enough → development ongoing . . .
  - Basic P4 functionality can be realized on any target
  - Every target offers different additional capabilities not programmed in P4 (e.g. multicast support)
  - These additional functionalities make P4 programs hardware dependent
P4 targets

Organization

P4 open source efforts are centralized on:

- Official website: https://p4.org
- Github: https://github.com/p4lang

P4 consortium members
Two versions available:

- **P4\textsubscript{14}, released in March, 2015**
  - unified language for all targets
  - development driven by hardware developers

- **P4\textsubscript{16}, released in May, 2017**
  - concentrating P4 language on core functionalities
  - development driven by software developers (P4 becoming a more C-like programming language)

Note: the following slides are based on the P4 tutorial from P4.org
P4 Core Overview

Figure 15: P4 model architecture
P4 Core
Different switch models

- P4 models present the capabilities of a P4-enabled device
- Models typically reflect the different features of different P4 targets
**P4 Core**

**Parser**

**Parser tasks**
- Finite State Machine (FSM)
- Produces a parsed representation of valid headers
- Describes all supported headers
- Describes the order in which headers may appear

**Deparser tasks**
- Executed before sending a frame
- Assemble the different fields and their order in a frame

Abstract representation of a packet parser [source: open-nfp.org]
Metadata

Tasks

- Data structures associated with every packet
- **Standard metadata:**
  - Default metadata provided by all P4 targets for every packet
  - e.g. ingress_port
- **Intrinsic metadata:**
  - Additional target-specific metadata provided for every packet
  - e.g. receive_timestamp
- **User-defined metadata**
  - Data created by the P4 program during runtime for every packet
  - e.g. new_tunnel_id
### P4 Core

#### Match tables

<table>
<thead>
<tr>
<th>name</th>
<th>field</th>
<th>match_kind</th>
<th>match_value</th>
<th>action</th>
<th>action data</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>encaps</td>
<td>ingress_port</td>
<td>exact</td>
<td>encapsulate_act</td>
<td>vlantag = 123</td>
</tr>
<tr>
<td>[1]</td>
<td>default</td>
<td>port_0</td>
<td></td>
<td>drop</td>
<td></td>
</tr>
</tbody>
</table>

**Example table**

**Tasks**

- Each table contains one or more entries
- An entry contains a specific key to match on (field) and a single action (action) to be executed, and additional data (action data)
- The match operation supports different types (match_kind):
  - **exact**: select the entry exactly matching match_value
  - **lpm**: select the entry with the longest prefix matching
  - **ternary**: select with some ignored bits e.g. match_value of 10*1 → 1011 or 1001
- P4 targets may define additional match types, e.g. **range**
- If no entry matches, the mandatory default entry is selected
Tasks

- Similar to C functions without any loops or pointers
- Modification of field values and headers (add or remove)
- Besides the packet/header data, the action also may get additional data from tables
- Primitives for metering, registers, counters, hashes and random numbers

Extern objects

- New in P4\textsubscript{16}
- Externs perform additional tasks which are either not written in or not supported by P4
- Architecture specific:
  - Software/NPU targets: extension via programmed functions (C, Python, \ldots)
  - FPGA: extension via VHDL/Verilog-defined functions
  - ASIC: no extension possible
Figure 19: Illustration of P4 match-action process [source: p4.org]
P4 Core
P4 Portable Switch Architecture (PSA)

Goal:

• Reference architecture for P4 switches
• Separate PSA specification available on p4.org
• Architecture describes common capabilities of network switch devices

Common capabilities

• Metadata definitions
• Hashes and checksums (only simple hashes e.g. CRC, no cryptographic hashes such as SHA)
• Counters and meters
• Registers
• Random number generators
• Access to timestamps

Example for non-common capabilities

• Capabilities of the traffic manager, such as packet generation
P4 example: IPv4 router

Disclaimer

- Basic P4 example
- Essential features are missing, no ARP/ICMP/VLAN/IPv6 handling
  → do not use this router for the project ;)

Chapter 6: Software Defined Networking – P4
P4 example: IPv4 router
Headers and fields definition

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

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

---

```
bit<n> defines unsigned int of length n
typedef introduces a shorter label for field declarations

header declares a new header. The following operations can be called on a header: isValid(), setValid(), and setInvalid().
```
P4 example: IPv4 router
Headers and fields definition

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

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

bit<n> defines unsigned int of length n
typedef introduces a shorter label for field declarations

header declares a new header. The following operations can be called on a header: isValid(), setValid(), and setInvalid().

What about the frame check sequence?
P4 example: IPv4 router

Headers and fields definition

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

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

- `bit<n>` defines unsigned int of length n
- `typedef` introduces a shorter label for field declarations

- `header` declares a new header. The following operations can be called on a header: `isValid()`, `setValid()`, and `setInvalid()`.

  What about the frame check sequence?
  → Checked and added automatically
P4 example: IPv4 router

Headers and fields definition

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

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

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;
}

bit<n> defines unsigned int of length n
typedef introduces a shorter label for field declarations

header declares a new header. The following operations can be called on a header: isValid(), setValid(), and setInvalid().

What about the frame check sequence? → Checked and added automatically
**P4 example: IPv4 router**

**Metadata definition**

```c
/* Architecture */
struct standard_metadata_t {
    bit <9> ingress_port;
    bit <9> egress_spec;
    bit <9> egress_port;
    bit <32> clone_spec;
    bit <32> instance_type;
    bit <1> drop;
    bit <16> recirculate_port;
    bit <32> packet_length;
    ...
}

/* User program */
struct metadata {
    ...
}

struct headers {
    ethernet_t ethernet;
    ipv4_t ipv4;
}
```

*struct* defines an unsorted collection of members.
P4 example: IPv4 router

P4_{16} Parsers

- Parsers map packets to headers and metadata
- Parsers are written as state machines
- Each parser has three predefined stats:
  - start
  - accept
  - reject
- Additional states may be defined by the programmer
- Each state may execute statements and then transition to another state
- Loops are allowed
P4 example: IPv4 router
Parser definition

```
parser MyParser ( packet_in packet ,
    out headers hdr ,
    inout metadata meta ,
    inout standard_metadata_t std_meta ) {

  state start {
    transition parse_ethernet ;
  }

  state parse_ethernet {
    packet.extract ( hdr.ethernet );
    transition select (hdr.ethernet.ethType) {
      TYPE_IPV4: parse_ipv4 ; // 0x800
      default: accept ;
    }
  }

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

- `select` works similar to case statements in Java/C
- `select` ends after successful match (`default` is not executed after successful `TYPE_IPV4` match)
- `extract` set header and its fields to valid
P4 example: IPv4 router
Ingress and table definition

```p4
control MyIngress(inout headers hdr,
inout metadata meta,
inout standard_metadata_t std_meta) {
    action drop() { mark_to_drop(); }

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

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

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

A **control** block contains the functionality of the program.

Control blocks can represent different kinds of processing:
- Match-Action pipelines
- Deparsers
- Additional forms of processing (checksums)

Typically headers and metadata act as interfaces between control blocks.

Execution starts with **apply()** statement.
### P4 example: IPv4 router

#### IPv4 Table example

<table>
<thead>
<tr>
<th></th>
<th>field</th>
<th>match_kind</th>
<th>key</th>
<th>action</th>
<th>action data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hdr.ipv4.dstAddr</td>
<td>lpm</td>
<td>10.0.1.1/32</td>
<td>ipv4_forward</td>
<td>dstAddr=00:00:00:00:01:01, port=1</td>
</tr>
<tr>
<td>1</td>
<td>hdr.ipv4.dstAddr</td>
<td>lpm</td>
<td>10.0.1.2/32</td>
<td>drop</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>NoAction</td>
<td></td>
</tr>
</tbody>
</table>
P4 example: IPv4 router

Checksum verification

```p4
control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
  apply {
    verify_checksum(
      hdr.ipv4.isValid(), // check validity of header
      { // list of inputs
        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, // output
      HashAlgorithm.csum16 // hash calculation
    );
  }
}
```
P4 example: IPv4 router

Checksum calculation

control MyComputeChecksum(inout headers hdr, inout metadata meta) {
    apply {
        update_checksum(
            hdr.ipv4.isValid(), // check validity of header
            // list of inputs
            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, // output
        HashAlgorithm.csum16 // hash calculation
    );
}
}
P4 example: IPv4 router
Egress, deparser and switch definition

```p4
control MyEgress(inout headers hdr,
    inout metadata meta,
    inout standard_metadata_t std_meta) {
    apply {
        //no egress processing needed
    }
}

// no explicit deparser object => control
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}

Router(
    MyParser(),
    MyVerifyChecksum(),
    MyIngress(),
    MyEgress(),
    MyComputeChecksum(),
    MyDeparser()
) main;
```
Active area of research

P4, like OpenFlow, has attracted a lot of researchers

- Extension of the P4 language itself
- Proposition of new platforms supporting P4
- New protocols and services on top of P4
- Other open programming languages for common network functionalities (e.g., packet scheduling)
- …

Theses offered at the chair

- P4 benchmarking
- P4 extensions
- …
Acknowledgements

- Slides partially based on work by Cornelius Diekmann

