P4: A Hands-on Introduction

Open-NFP:
Bapi Vinnakota, Mary Pham, Jici Gao
Tutorial

Focused on getting you started with P4 development
Is:
• An introduction to significant features
• An introduction to development in P4
Is not:
• A comprehensive description of P4 theory and evolution
• An introduction to software-defined networking
After this tutorial
• Use the more advanced examples at p4.org
• Use development resources at open-nfp.org
We cover P4-14, big changes coming for P4-16.
## Outline

<table>
<thead>
<tr>
<th>P4 Concepts:</th>
<th>Introduction to P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab infrastructure:</td>
<td>Development environment for the labs</td>
</tr>
<tr>
<td>Lab 1 Simple NIC:</td>
<td>First complete P4 program</td>
</tr>
<tr>
<td>A Deeper Dive:</td>
<td>Based on the P4 1.1 specification</td>
</tr>
<tr>
<td>Lab 2 Headers:</td>
<td>Protocol headers, custom actions</td>
</tr>
<tr>
<td>Lab 3 Match-Action:</td>
<td>Tables, control flow management</td>
</tr>
<tr>
<td>Advanced topics:</td>
<td>Language constructs, portability etc.</td>
</tr>
<tr>
<td>Lab 4 Adv topics:</td>
<td>QoS Metering</td>
</tr>
<tr>
<td>P4-16:</td>
<td>Significant changes</td>
</tr>
<tr>
<td>P4 Resources:</td>
<td>Resources to learn and participate</td>
</tr>
</tbody>
</table>
1. Simple TCP/IP NIC
   ▪ Headers, actions and tables
   ▪ Compiling and downloading a simple project

2. Custom headers to implement timestamps
   ▪ Defining and implementing a custom header

3. Custom actions to measure latency between timestamps
   ▪ Primitive actions unique to an architecture
   ▪ Compound actions

4. Advanced QoS metering lab
   ▪ Dynamically changing data plane parameters
P4 CONCEPTS
Software Defined Networking

- Better service delivery
- Faster service creation
- Network adapts to demand

Figures © Open Networking Foundation from: https://www.opennetworking.org/sdn-resources/sdn-definition
• Open Flow: control plane managed in software through OpenFlow Controllers
• The dataplane is described as a match-action dataflow implemented in OpenFlow Switches
• Standards
  • Dataplane protocols
  • Structure of the match-action flow
  • Interface to update flow tables

Figure © Open Networking Foundation from: https://www.opennetworking.org/sdn-resources/sdn-definition
# Match-Action Table

<table>
<thead>
<tr>
<th>Header Fields</th>
<th>Counters</th>
<th>Actions</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>If ingress port == 2</td>
<td></td>
<td>Drop packet</td>
<td>32768</td>
</tr>
<tr>
<td>if IP_addr == 129.79.1.1</td>
<td></td>
<td>re-write to 10.0.1.1, forward port 3</td>
<td>32768</td>
</tr>
<tr>
<td>if Eth Addr == 00:45:23</td>
<td></td>
<td>add VLAN id 110, forward port 2</td>
<td>32768</td>
</tr>
<tr>
<td>if ingress port == 4</td>
<td></td>
<td>forward port 5, 6</td>
<td>32768</td>
</tr>
<tr>
<td>if Eth Type == ARP</td>
<td></td>
<td>forward CONTROLLER</td>
<td>32768</td>
</tr>
<tr>
<td>If ingress port == 2 &amp;&amp; Eth Type == ARP</td>
<td></td>
<td>forward NORMAL</td>
<td>40000</td>
</tr>
</tbody>
</table>

Example from “OpenFlow 90 Minutes”, Indiana Center for Network Translational Research and Education  
OpenFlow Match-Action Tables
A little bit more complicated

A flow table consists of flow entries.

Match Fields  Priority  Counters  Instructions  Timeouts  Cookie  Flags

Figures © Open Networking Foundation

©2016 Open-NFP
Open Flow limitations

• Cloud/Virtualization:
  • Customer network inside the cloud (AWS/Azure/Google)
  • Led to a significant increase in the number of protocols
  • Large private networks employ their own protocols

• OpenFlow impact
  • Dataplane protocols need to be standardized
  • Slow cumbersome process

• SDN Needs a programmable Data Plane
Programmable Network Dataplane

- Multiple approaches to a programable dataplane
- Architecture diversity in the dataplane driven by performance and power requirements
- Typically tight link between programming model and the architecture.
Languages for the Data Plane

Examples

• x86 – restrict flexibility to improve performance
  • eBPF: Virtual machine in the unix kernel for safe execution
  • OVS: Match-action based switching datapath

• Hardware-based – necessary to use special-purpose HW
  • Protocol Oblivious Forwarding: To program/configure forwarding elements, driven by Huawei
  • Protocol-Independent Forwarding: Open-Flow compatible method to describe packet forwarding w/o requiring standards definition
  • Micro-C: C-based language to program network processors. From Netronome and Intel.

Proposed a simple RISC VM with a limited instruction set and control flow. Implemented in the kernel. Can treat it as a reference architecture.

Packet processing implemented as filters on this reference architecture.

From: The BSD Packet Filter: A New Architecture for User-level Packet Capture
Networking in the Linux kernel

Network programs are filters. Allow or drop packets. Filters expressed as flow graphs

BPF defines a reference architecture with a small instruction set. Filters are implemented with the instruction set.

Architecture implemented as a VM in the kernel

From: The BSD Packet Filter: A New Architecture for User-level Packet Capture
Combined parsing and decision making

From: The BSD Packet Filter: A New Architecture for User-level Packet Capture
New Language Requirements

- Expressive/Complete
  - Support flow switching and flow termination
  - Compact description of common
  - Unambiguous in expressing networking functionality
- Flexible
  - Support the expected evolution of the data plane
  - Orthogonalize the data plane HW and SW
- Performant
  - Trade flexibility for performance
  - Support architecture-specific performant features
- Portable
  - Span the range of architectures in the dataplane
P4: A Language for the Data Plane

Language designed for switches
• Match-action data plane
• Reference switch architecture

P4 Program specifies
• A parser that extracts information from packets
• A match-action dataflow that modifies the packets

P4 programs are used to program a “target”
• Configure a switch target matching reference architecture
• Download a binary to programmable architectures. Match-action is an intermediate representation.

The P4 program specifies the dataplane as a match-action dataflow.
• Table contents expected to be loaded separately. Almost the “inverse” of OpenFlow.
P4 Compilation Process

Compilation leverages work from the ONF

P4 Programs compiled to an architecture independent intermediate representation

Compiled to a platform-specific binary or configuration from the intermediate representation

Figure © Open Networking Foundation
From OF-PI: A Protocol Independent Layer
P4 Flow Abstraction

- **Parser**
  - Headers
    - 1234
    - abcd
  - Metadata
    - Time
    - Port
  - Headers
    - Wxyz
    - 6789

- **Match-action**
  - Headers
    - Wxyz
    - 6789

- **De-parser**
  - Headers
    - Wxyz
    - 6789
This will be Lab 1. Downloaded to a Netronome Agilio NIC.

Next:
- Brief commercial for Agilio NICs.
- Review the lab infrastructure.
- Build to this NIC
Agilio™ Platform
High Performance, Scalable, & Efficient Server-based Networking

April 2016
High-Performance Server-based Networking Leadership

Agilio™ Server-Based Networking Software

Agilio™ CX Intelligent Server Adapters
1x and 2x 10/40GbE Production Solutions Now
2x 25GbE Samples in Q4 2016

- Delivers up to 6X lower TCO for IT and NFV workloads
- Brings the speed of software innovation to hardware
- OpenStack-managed Open vSwitch, Linux Firewall, Contrail vRouter & P4 acceleration

Bringing the efficiencies of mega-scale data centers to mainstream server networking
NFP-4000 Silicon Used in the Agilio CX Solution

- Delivers highest scale & best price-performance
- Hardware accelerators perform compute-intensive functions such as hashing, crypto, CAM, atomic and other functions
- Optimal multi-threading between processing cores, H/W accelerators and memory banks
- Multi-threaded memory engines and banks of SRAM tightly coupled with atomic and other hardware accelerator functions
- Delivers multi-terabit bi-sectional bandwidth between processing elements
- Avoids bus contention and saturation issues

● Datapath implementations (OVS, etc.)
● Extensible using P4 and C
● Multi-threaded processor cores
● Concurrent IO operations

Highest Multi-threading Silicon Architecture
Agilio™ OVS Implementation

OpenStack Ready

- OpenStack Nova
- OpenStack Neutron OVS ML2
- Open Daylight Controller (ODL)

Seamless Integration with Control Plane

Compute Node

- OpenStack Nova Agent
- ovs-dbserver
- ovs-vswitchd
- VM
- VM
- VM
- VM

Linux Kernel

- OVS Datapath
- Match Tables
- Actions
- Tunnels

SR-IOV & VirtIO Connectivity to VMs

Transparent Offload

- Agilio-CX
- OVS Datapath
- Match Tables
- Actions
- Tunnels

Deliver to Host
Update Statistics
LAB INFRASTRUCTURE
Introduction

• Programming with the Netronome SDK6 IDE on production hardware. Custom lab infrastructure to give you this experience.
• Best done as a shared experience. 2-3 people working on labs together as a group.
• Instructors from Netronome are here to assist you. Feel free to proceed at your own pace through the content.
Lab Infrastructure

Lab infrastructure
- IDE VMs, log in to yours to write P4 code, 2-3 developers share an IDE VM
- Lock a server from the pool to download/run code (BIG RED CARD)
- Use the IP address, release the server when you’re done
Working to make it available for hardware evaluation and R&D
RDP into your Windows VM (from your PC): Step 1

Name or IP address for your Cloud VM
RDP into your Windows VM (from your PC): Step 2

Select this
RDP into your Windows VM (from your PC): Step 3

uid: given to you
pwd: given to you

OK, Yes to all subsequent pop-ups
Microsoft Remote Desktop

With the Microsoft Remote Desktop app, you can connect to a remote PC and your work resources from almost anywhere. Experience the power of Windows with RemoteFX in a Remote Desktop client designed to help you get your work done wherever you are.

Getting Started
Configure your PC for remote access first. Download the RDP assistant to your PC and let it do the work for you: http://aka.ms/rdassistant
Learn more about remote desktop apps here: http://aka.ms/rdapps

Features
- Access to remote resources through the Remote Desktop Gateway
- Secure connection to your data and applications with breakthrough Network Layer Authentication (NLA) technology
- Simple management of all remote connections from the connection center
- High quality video and sound streaming with improved compression and bandwidth usage
- Easy connection to multiple monitors or projectors for presentations
- Print from Windows applications to any printer configured on your Mac
- Access local files on your Mac from your Windows applications
- Support for Azure RemoteApp

We're improving our app – learn more! http://aka.ms/rdmac-preview

What's New in Version 8.0.31
- Security: We are changing the folder permissions for the Remote Desktop application back to the default permissions. With the last release, the app was granted write...
RDP into your VM (from a Mac) Create a connection

- Save after creating the profile
- Can be anything
- Name or IP address for your cloud VM
- uid: given to you
- pwd: given to you
IDE VM: Slide 1 When you log in

Programmer’s Studio IDE

Terminal to log into servers with Agilio

Minimize (if you see it)

Open for Docs
1. Click here

2. This window opens, select “Local Drive”

3. This window opens, select “Users”

4. This window opens, select “Public”

5. This window opens, select “Public Documents”
C:\Users\Public\PublicDocuments\workshop_0416\p4lab

6. This window opens, select “workshop_0416”

7. Open pdf for Lab1

Folder also has source code for all the examples

8. Open Programmer Studio – start p4ing!

9. When you get to the step to use hardware, you will need an IP address. Get a token card with an IP address for a server. Release the server to the pool when you’re done by returning the token card.
open-nfp for Research and Development

- Register at [www.open-nfp.org](http://www.open-nfp.org).
  - Portal for research in dataplane acceleration
  - These tutorials and several other resources
- Hardware
  - Purchase directly from Netronome’s distributor, with a credit card
    Academic institutions get a significant discount.
  - Remote access with the cloud lab infrastructure
- Software
  - OVS acceleration software
  - PS/SDK used in today’s labs
  - Request access - please give us a project description.
- Join the 25+ universities, research institutes and companies currently on open-nfp
  - Engage with the community through the open-nfp Google Group.
LAB 1: A SIMPLE NIC
Lab 1: Overview

Develop a Simple NIC in P4.

NIC either drops packets or forwards them according to the input port.

Use the Netronome IDE to write the code, compile onto the Agilio ISA. Also instructions on debugging. The code is compiled onto a single micro-engine. Test with data from a pcap file.

Instructions in a pdf document in two stages. First compilable program simply drops all packets on all interfaces. Next version, forwards packets per their ingress port.
P4 Flow Abstraction

Ingress port

headers

ethernet

payload

headers

metadata

match-action

deparser

headers

wxyz

6789

wxyz

6789
Simple NIC: Headers

Defines an “Ethernet” header

An Ethernet header instance

Start of the parse tree

Extracts an Ethernet header, end of the parse tree

```c
header_type eth_hdr {
  fields {
    dst : 48;
    src : 48;
    etype : 16;
  }
}

header eth_hdr eth;

parser start {
  return eth_parse;
}

parser eth_parse {
  extract(eth);
  return ingress;
}
```
Simple NIC: Actions

Defines an action to drop packets

```c
action drop_act() {
    drop();
}
```

Defines an action to set the egress port

```c
action fwd_act() {
    modify_field(standard_metadata.egress_spec, prt);
}
```
Simple NIC: Tables

```c
table in_tbl {
    reads {
        standard_metadata.ingress_port : exact;
    }
    actions {
        drop_act;
        fwd_act;
    }
}
```

Defines a table. Reads the ingress port

Library of actions available to the rows in the table
Simple NIC: Control

Packet processing starts (and ends too!) here

control ingress {
apply(in_tbl);
}
Defines an “Ethernet” header
An Ethernet header instance
Start of the parse tree
Extracts an Ethernet header, end of the parse tree
Defines an action to drop packets, uses P4 primitive action
Defines an action to set the egress port
Defines a table. Reads the ingress port
Library of actions available to the rows in the table
Packet processing starts (and ends too!) here
Process to Develop Code

In each lab:
- You will develop code in the SDK in P4 (and C)
- Download it to the card
- Run the code
- Generate test packets to observe the functionality
- Monitor the results

Develop on a PC with Programmers Studio
Download to a server

SERVER
You will be programming the Agilio ISA (red) with the Netronome SDK 6.0 (blue).

The second NIC, the cable and PCAP are for the labs, not required for code development.
Build a telemetry lab using TCP timestamping in three parts

Part 1: Simple port forwarding. Start with a P4 program with Ethernet, IP, TCP headers defined.

Part 2: Defining, adding and processing a custom header. An optional extension to the TCP header to insert timestamps.

Part 3: Adding a custom action in C to process timestamps and calculate average latency between successive packets with timestamps.
Lab 1: Simple TCP Forwarding

1. Headers defined for Ethernet, IPv4, TCP
2. Simple forwarding action “do_forward”
3. Single table to forward port to port
4. Code to compute checksum (we will not review)
5. You will fill in the rules
Follow Part 1 of p4lab.p4 in P4Lab_TCP_TS

You will create a P4 project in Programmers Studio
Insert a pre-built P4 source code file into the project
Make a copy of the file before you insert it
Create a user configuration (match-action rules) file
Build and compile, load an Agilio ISA in debug mode
Inject tcp/ip packets through a NIC and capture them

Follow the worksheet
Ask an instructor for help if you need it.
P4: A DEEPER DIVE
P4 abstractions

Switch-oriented language. Structure follows the reference architecture

• Base types: Integers and strings and basic operations on them
• Headers: Types (i.e. specification) and instances
• Parser function: Instruction sequence describing stateful header processing
• Action function: To manipulate packet contents
• Table: Match-action tables with match criteria/actions
• Control flow function: How does data flow through the tables
• Stateful memories: Counters and registers
• Extern methods: Method to support architecture-specific functions e.g. crypto
LAB 2: HEADERS, PARSING
P4 Flow Abstraction: Headers

Define all the header types to be recognized and processed

- Defines the structure: fields and their location
- Defines the total length of the header
  - No “standard” list of headers defined.
  - Room for custom headers but
  - Known protocols have to be respecified

- Rules
  - Headers have a natural number of bytes
  - One variable length field per header type (not header)
  - Calculated fields (e.g. checksums) handled separately
Header Types and Header Instances

Declare an actual header instance for each type

- One instance per each occurrence expected. e.g. IP-over-L3 VPN would define an IP header type once, instantiate twice
- Without an instance, no space allocated to the header in the parsed representation
- Can also declare a “header stack” – an array of headers of the same type. Beyond the scope of this session.

Example from: https://github.com/open-nfsw/NetPaxos/blob/master/src
Define all the header types to be recognized and processed
- Defines the structure: fields and their location
- Defines the total length of the header
  - No “standard” list of headers defined.
  - Room for custom headers but
    - Known protocols have to be respecified
- Rules
  - Headers have a natural number of bytes
  - One variable length field per header type (not header)
  - Calculated fields(e.g. checksums) handled separately
- Declare an actual header instance for each type
  - One instance per each occurrence expected. e.g. IP-over-L3 VPN would define an IP header type once, instantiate twice
  - Without an instance, no space allocated to the header in the parsed representation
  - Can also declare a “header stack” – an array of headers of the same type. Beyond the scope of this session.

Example from:
https://github.com/open-nfsw/NetPaxos/blob/master/src
Parser

Function produces a parsed representation that lists valid headers

Function that describes all supported header stacks

All other packets default
• Undefined header types
• Undefined header stacks
• Undefined field values

Complex parsers possible (e.g. parser value sets), beyond the scope of this tutorial

Example from the p4 1.1 spec
Valid headers in the parsed representation on three possible paths in the parse function

Example from the p4 1.1 spec
Parser implementation

```c
parser ethernet{
    extract(ethernet_type_instance); // Start with the ethernet header
    return select(latest.ethertype) {
        0x8100: vlan;
        0x800: ipv4;
        0x0806: arp;
        0x86DD: ipv6;
        default: ingress;
    }
}

parser vlan {
    extract(vlan_type_instance); // Extract vlan header
    return select(latest.ethertype) {
        0x800: ipv4;
        0x0806: arp;
        0x86DD: ipv6;
        default: ingress;
    }
}
```

Modified example from the p4 1.1 spec
Defines a node in the parse tree
Instance should have been defined
shorthand for “current” packet
Must be specified in the “Ethernet” type
definition in the same p4 program
Pass unrecognized packets to the control flow

Extracts the entire header though only a portion is used in the parse tree
Must be specified in the “vlan” type
definition in the same p4 program

```
parser ethernet{
  extract(ethernet_type_instance); // Start with the ethernet header
  return select(latest.ethertype) {
    0x8100: vlan;
    0x800: ipv4;
    0x0806: arp;
    0x86DD: ipv6;
    default: ingress;
  }
}
```

```
parser vlan {
  extract(vlan_type_instance); // Extract vlan header
  return select(latest.ethertype) {
    0x800: ipv4;
    0x0806: arp;
    0x86DD: ipv6;
    default: ingress;
  }
}
```
Metadata

Non-packet data useful in processing decisions

Standard metadata expected to be available in all architectures

Each P4 target may provide additional metadata
Lab 1: Simple TCP Forwarding

1. Headers defined for Ethernet, IPv4, TCP
2. Simple forwarding action “do_forward”
3. Single table to forward port to port
4. Code to compute checksum (we will not review)
5. You will fill in the rules
Lab 2: Defining, Parsing Timestamp Headers

1. Expand on Lab 1
2. Define a TCP optional timestamp header field
3. Add the header to the parse tree (deparser needs be the same as parser)
4. Create a metadata field to measure the time just before a packet is parsed
5. Add a table to add the timestamp header to a packet without it
6. Populate the timestamp value from the metadata using an action and table
7. Fill in the rules for the new table in a new user config file
Continue the project you created for Lab 1
Define the timestamp header as an option in a TCP header
Add code for the metadata, ts header, ts table
Create a user configuration file for the new table
Build and compile, load an Agilio ISA in debug mode
Inject tcp/ip packets through a NIC and capture them

Follow the worksheet
Ask an instructor for help if you need it.
LAB 3: MATCH-ACTION FLOW
P4 Flow Abstraction: Match-Action

Actions – functions to modify packet contents
- Primitive actions
- Externs (custom actions)
- Compound actions

Tables
- Fields processed
- Action set

Control Flow
- Sequencing packet flow through tables
Actions

Functions that alter the parsed representation of a packet

- **Primitive actions**: Native to the language
  - Built off operations on base types
- **Custom actions**: Unique to a target architecture. Built off “black box” methods.
  - Expose features unique to an architecture with “extern” methods
- **Compound actions**:
  - Custom actions formed by combining primitive and/or custom actions
- More complex concepts – action profiles, action selectors are beyond our scope

---

<table>
<thead>
<tr>
<th>primitive name</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_header</td>
<td>Add a header to the packet’s Parsed Representation</td>
</tr>
<tr>
<td>copy_header</td>
<td>Copy one header instance to another</td>
</tr>
<tr>
<td>remove_header</td>
<td>Mark a header instance as invalid</td>
</tr>
<tr>
<td>modify_field</td>
<td>Set the value of a field in the packet’s Parsed Representation</td>
</tr>
<tr>
<td>modify_field_with_hash_based_offset</td>
<td>Apply a field list calculation and use the result to generate an offset value</td>
</tr>
<tr>
<td>truncate</td>
<td>Truncate the packet on egress</td>
</tr>
<tr>
<td>drop</td>
<td>Drop a packet (in the egress pipeline)</td>
</tr>
<tr>
<td>no_op</td>
<td>Placeholder action with no effect</td>
</tr>
<tr>
<td>push</td>
<td>Push all header instances in an array down and add a new header at the top</td>
</tr>
<tr>
<td>pop</td>
<td>Pop header instances from the top of an array, moving all subsequent array elements up</td>
</tr>
<tr>
<td>count</td>
<td>Update a counter</td>
</tr>
<tr>
<td>meter</td>
<td>Execute a meter operation</td>
</tr>
<tr>
<td>generate_digest</td>
<td>Generate a packet digest and send to a receiver</td>
</tr>
<tr>
<td>resubmit</td>
<td>Resubmit the original packet to the parser with metadata</td>
</tr>
<tr>
<td>recirculate</td>
<td>Resubmit the packet after all egress modifications</td>
</tr>
<tr>
<td>clone_ingress_pkt_to_ingress</td>
<td>Send a copy of the original packet to the parser. Alias: clone_i2i</td>
</tr>
<tr>
<td>clone_egress_pkt_to_ingress</td>
<td>Send a copy of the egress packet to the parser. Alias: clone_e2i</td>
</tr>
<tr>
<td>clone_ingress_pkt_to_egress</td>
<td>Send a copy of the original packet to the Buffer Mechanism. Alias: clone_i2e</td>
</tr>
<tr>
<td>clone_egress_pkt_to_egress</td>
<td>Send a copy of the egress packet to the Buffer Mechanism. Alias: clone_e2e</td>
</tr>
</tbody>
</table>

Table from the p4 1.1 spec
**Primitive Action**

```plaintext
modify_field(dest, value [, mask ])
```

**Summary**
Set the value of the given field in packet’s Parsed Representation

**Parameters**
- **dest** (FLD or R-REF) The name of the field instance to modify (destination).
- **value** (VAL, FLD or R-REF) The value to use (source).
- **mask** (VAL) An optional mask to use identifying the bits to change.

**Description**
Update the indicated field's value. The `value` parameter may be any of:
- An immediate value (a number).
- A value from the matching entry's action parameter data; in this case, the name of a parameter from the enclosing function is used.
- A Parsed Representation field reference.
- A register reference.

This allows the programmer to copy one field to another. An implicit cast is inserted by the compiler if the types of the source and destination differ, as described in Section 2.4.

If the parent header instance of `dest` is not valid, the action has no effect. If `value` is a field reference and its parent header is not valid, the operation has no effect.

If `mask` is specified, then the field becomes `(current_value & ~ mask) | (value & mask)`. If `mask` is not specified, the operation has the effect of a "set", modifying all bits of the destination.

Figure © The P4 Language Consortium
from: The P4 Language Specification
Objects to express behavior not expressible with P4

Externs are methods used as “black boxes”. Used for primitive actions unique to an architecture.

In ASICs, externs access accelerators unique to the accelerator.

In programmable NPUs, a library is a collection of externs. Two P4 programs on the same NPU may have completely different externs if they use different libraries.
int pif_plugin_get_forwarding_start_time(EXTRACTED_HEADERS_T *headers, MATCH_DATA_T *data)
{
    PIF_PLUGIN_paxos_T *paxos;
    timestamp_low = local_csr_read(local_csr_timestamp_low);
    timestamp_high = local_csr_read(local_csr_timestamp_high);

    if (!pif_plugin_hdr_paxos_present(headers)) { return PIF_PLUGIN_RETURN_FORWARD; }
    
    paxos = pif_plugin_hdr_get_paxos(headers);

    paxos->fsl = timestamp_low;
    paxos->fsh = timestamp_high;
    return PIF_PLUGIN_RETURN_FORWARD;
}

primitive_action get_forwarding_start_time();
primitive_action get_forwarding_end_time();
primitive_action get_coordinator_start_time();
primitive_action get_coordinator_end_time();

..

action increase_sequence() {
    get_coordinator_start_time();
    register_read(local_metadata.inst, inst_register, 0);
    modify_field(paxos.inst, local_metadata.inst);
    modify_field(local_metadata.inst, local_metadata.inst + 1);
    register_write(inst_register, 0, local_metadata.inst);
    modify_field(paxos.msgtype, PHASE_2A);
    modify_field(udp.checksum, 0);
    get_coordinator_end_time();
}
Custom actions defined

Compound action start

p4 primitive actions

Modifies headers in custom protocol

```p4
primitive_action get_forwarding_start_time();
primitive_action get_forwarding_end_time();
primitive_action get_coordinator_start_time();
primitive_action get_coordinator_end_time();
...
...
action increase_sequence() {
    get_coordinator_start_time();
    register_read(local_metadata.inst, inst_register, 0);
    modify_field(paxos.inst, local_metadata.inst);
    modify_field(local_metadata.inst, local_metadata.inst + 1);
    register_write(inst_register, 0, local_metadata.inst);
    modify_field(paxos.msgtype, PHASE_2A);
    modify_field(udp.checksum, 0);
    get_coordinator_end_time();
}
```

Example from: https://github.com/open-nfpsw/NetPaxos/blob/master/src/headers.p4
Match-Action Tables

Structures specifying match-action operations:

- Match against the contents of metadata and the parsed representation
- Action functions update the contents of the parsed representation and metadata and interact with the system
- Actions on the first highest-priority match

P4 Programs

- Declare all the tables. Do not specify the contents
- Tables updated through an API (not standardized yet)
Table Declaration in P4

A Table definition:
• Assumes a table row consists of keys, counters and action sets
• Specifies the fields to be read from the parsed representation
• Specifies the type of comparison (match) of key to field values
• Lists all the actions supported in that table for a key matches
• The actual actions executed vary from row to row

Match types supported in table definition
• Exact
• Ternary
• Longest prefix
• Index
• Range
• True/False

The rows of the table are loaded through an API after the code is compiled to the target
Example Table & Contents in P4

**Table 1:**

<table>
<thead>
<tr>
<th>name</th>
<th>field</th>
<th>match_type</th>
<th>match_value</th>
<th>action</th>
<th>parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>encap_0</td>
<td>standard_metadata.ingress_port</td>
<td>exact</td>
<td>p0</td>
<td>encap_act</td>
<td>prt = v0.0, tag=43</td>
</tr>
<tr>
<td>default</td>
<td></td>
<td></td>
<td></td>
<td>drop_act</td>
<td></td>
</tr>
</tbody>
</table>

Actions have to be invoked by rules, not used automatically even if defined in the table.
In each lab
- You will develop code in the SDK in P4 (and C)
- Download it to the card
- Run the code
- Generate test packets to observe the functionality
- Monitor the results
Global state objects that span packets.

Counters declared as arrays. Associated with tables. Can be updated implicitly on matches or explicitly through actions.

Meters measure data rates in packets or bytes. Specification is loosely defined.

Registers are global memory. (Challenges to highly multi-threaded architectures.) Used for one-off events or to make decisions.
Control Flow

Processes input packets through a match-action flow. Two modes to select actions with an apply statement

- Global action on a match
- Action selected inside the table on a match
  - Key on each row is compared (matched) with parsed header contents
  - If multiple rows match, the actions in the highest priority row are executed

Three modes to sequence “apply” statements

- Sequential: apply statements executed sequentially
- Multi-path: control flow specifies execution order. Path selected using
  - Hit-Miss selection: “if-else” on headers/metadata
  - Action selection: actions executed on table matches
Control Flow: if-else sequencing

Start of the control block
Custom packet type
check header validity
Apply statements can also be applied sequentially without if-else constructs

From: https://github.com/open-nfpsw/P4Probe/blob/master/p4Probe1.p4
Control Flow: action-based sequencing

apply((routing_table) {
  ipv4_route_action { // IPv4 action was used
    apply(v4_rpf);
    apply(v4_acl);
  }
  ipv6_route_action { // IPv6 action was used
    apply(v6_option_check);
    apply(v6_acl);
  }
  default { // Some other action was used
    if (packet_metadata.ingress_port == 1) {
      apply(cpu_ingress_check);
    }
  }
})

Selected depending on the action chosen in routing_table

Need a default path

from p4 1.1 specification
1. Headers defined for Ethernet, IPv4, TCP
2. Simple forwarding action “do_forward”
3. Single table to forward port to port
4. Code to compute checksum (we will not review)
5. You will fill in the rules
Lab 2: Defining, Parsing Timestamp Headers

1. Expand on Lab 1
2. Define a TCP optional timestamp header field
3. Add the header to the parse tree (deparser needs be the same as parser)
4. Create a metadata field to measure the time just before a packet is parsed
5. Add a table to add the timestamp header to a packet without it
6. Populate the timestamp value from the metadata using an action and table
7. Fill in the rules for the new table in a new user config file
Lab 3:

1. Expand on Lab 2
2. Create a bridge in the host to “reflect” packets with the ts header back to the ISA
3. Add an action to remove the ts header from a packet with the header
4. Create a custom action, instantiate as a primitive, to compute gap between timestamps
5. Add a table with these two new actions
6. The delta is the time taken to go into the host and come back
7. Inject packets to make the roundtrip, measure the distribution of deltas
Follow Part 3 of p4lab.p4 in P4Lab_TCP_TS

Continue the project you created for Labs 1 and 2
Create a new primitive action for a custom C plugin (already created)
Add code for a compound action to remove the ts header
Create a table for these two actions
Create a user configuration file for the new table
Build and compile, load an Agilio ISA in debug mode
Inject tcp/ip packets through a NIC and capture them

Follow the worksheet
Ask an instructor for help if you need it.
Miscellaneous

Interesting things we haven’t covered
• Concepts
  • Cloning and recirculation
  • Checksums, hash tables and streaming objects
  • Counters and saturation arithmetic
  • Parser value sets
  • Header stacks
• Systems issues
  • Target architecture specification
  • Virtualization/Isolation
  • API to update rules
• Portability
  • No target is expected to support the entire specification
  • Portability is expected across a family of devices
  • Two targets supporting the same P4 code expected to work “the same way”
Advanced Features, Going to “Production”

Debugging P4 programs with Programmers Studio
- Insert breakpoints, monitor them on microengines
- Simple NIC lab has examples of using breakpoints
- Follow the Lab 1 worksheet

CLI to update rules, Linux toolchain
- Update rules, inspect memory
“P4 gives a seamless way to build networks using a mix of programmable and fixed-function switches, and lets us introduce new features and protocols into the network in software, rather than waiting for new hardware.”

Significant effort to map P4 to different data plane machines:

**OVS**
- VMWare: PISCES: A Programmable, Protocol-Independent Software Switch, Sigcomm 2016
- eBPF
- Intel: P4 on the Edge, P4 Workshop, May 2016

**FPGAs**
- Xilinx, P4 for an FPGA Target, P4 Workshop, May 2015

**GPUs**
- UMass Lowell:

**NPUs**
- Netronome: SDK6

**Switches**
- Barefoot: Barefoot Capilano
- HP: P4 and OpenSwitch, P4 Workshop, May 2016
Lab: QoS Metering

1. VLANs to manage traffic
2. Add a table that implements QoS for traffic
3. Table uses native P4 primitives for QoS
4. On each VLAN, throttles traffic if it exceeds rate or burst
5. On the host run pings to "see" the impact of throttling
6. Attach the v0.1 interfaces to VMs (bonus!)
Follow p4lab.p4 in P4QoS_Metering

Create a new P4 project in Programmers Studio
Import the entire P4 source code
Import the complete rules file (user configuration file)
Build and compile, load an Agilio ISA in debug mode
Run pings from the host on the VLAN
Use the Linux CLI to change QoS parameters
Run more pings to see the effect

Follow the worksheet
Ask an instructor for help if you need it.
• Significant changes, backwards incompatible. Syntax of the language changing.
• Language being transformed to core + libraries. Many first class elements being transformed to libraries.
• Intent is to preserve a stable core. Have the libraries evolve as networking requirements evolve
• Ambiguity in target specification being reduced. Aim to produce more predictable behavior across targets.
• Specification expected to be complete by end of 2016.
p4.org

- content: specification, github, blog, tutorials
- organization: developer days, conferences, language dev
- github: switch model, tools, example code
Interesting Research

- Load balancer: Implementing server load balancing as a network function
- Consensus as a service: PAXOS protocol for consensus across multiple agents
- In-Network Telemetery: Protocol to analyze network performance
- Bloom-Filter Based Source Routing
open-nfp for Research and Development

- Register at [www.open-nfp.org](http://www.open-nfp.org).
  - Portal for research in dataplane acceleration
  - These tutorials and several other resources
- Hardware
  - Purchase directly from Netronome’s distributor, with a credit card
    Academic institutions get a significant discount.
  - Remote access with the cloud lab infrastructure
- Software
  - OVS acceleration software
  - PS/SDK used in today’s labs
  - Request access - please give us a project description.
- Join the 25+ universities, research institutes and companies currently on open-nfp
  - Engage with the community through the open-nfp Google Group.