Spring 2026

COMP6103P
Advanced Computer Networking

Software Defined Network

From Control Plane to Data Plane Innovation

Topic Outline (~3 Lectures)

- **Background**
- **From Control Plane To Data Plane (1 Lecture)**
  - OpenFlow: Enabling Innovation in Campus Networks, SIGCOMM CCR 08
  - P4: Programming Protocol-independent Packet Processors, SIGCOMM CCR 14
- **In-Networking Computing (2 Lectures)**
  - ATP: In-network Aggregation for Multi-tenant Learning, NSDI 21 (Best Paper)
  - PINT: Probabilistic In-band Network Telemetry, SIGCOMM 20

Background: Switch Architecture

Adjacent router

Co

Routing

OSPF

Switching

Data plane

Router

Management / Policy plane

Configuration / CLI / GUI

Control plane

Static routes

OSPF

Neighbor table

Link state database

IP routing table

Forwarding table

Data plane

Adjacent router

Control plane

OSPF

Data plane

Background: Switch Data Plane

- **Fast** packet **forwarding**
- Hardware implementation
- **Match-Action**
  - TCAM is one possible hardware implementation of a Match-Action Table

Background: Switch Data Plane

Controller

Southbond Interface

Switch

Match Action Table

Entry

Match fields	Actions	Counter
...		

Packet In

Table 0

Table 1

...

Table n

Packet Out

Pipeline

Background
Switch Control Plane

- **Routing** protocols
- Topology discovery
- **Compute forwarding table**
  - **OSPF:** intra-domain routing
  - **BGP:** inter-domain routing

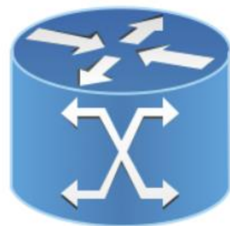
An Example

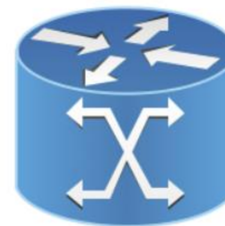
Subnet1: 10.0.1.0/24

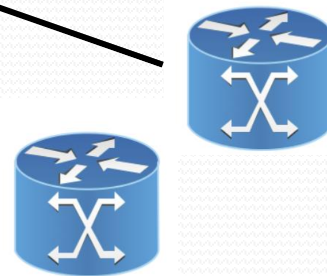
Subnet2: 10.0.2.0/24

Switch A

Switch B







Server A : 10.0.1.1

Server B: 10.0.2.1

Server A wants to send a message to Server B, how does Switch A know where to forward the packet?

An Example – Control Plane

- Switch A:
  - Send ***I have subnet 10.0.1.0/24***
- Switch B:
  - Send ***I have subnet 10.0.2.0/24***
- Both switches can use **routing protocol**, such as OSPF to **calculate the forwarding tables**

An Example – Control Plane

Switch A

Destination	Next Hop
10.0.1.0/24	Local
10.0.2.0/24	Switch 2

Switch B

Destination	Next Hop
10.0.2.0/24	Local
10.0.1.0/24	Switch 1

Control plane computes these tables!

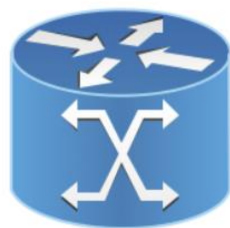
An Example – Data Plane

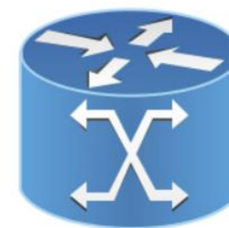
Subnet1: 10.0.1.0/24

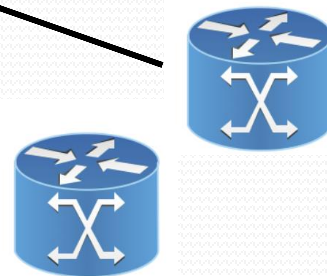
Subnet2: 10.0.2.0/24

Switch A

Switch B







Server A : 10.0.1.1

Server B: 10.0.2.1

Server A sending a packet
Src: 10.0.1.1
Dst: 10.0.2.1

→

Switch A:
Step 1:
Look up routing table
Step 2:
Forward to Switch B

→

Switch B:
Step 1:
Forward to Host B

Quick Summary

- Traditional networks rely on **distributed** routing protocols such as OSPF and BGP.
- Each switch **independently** computes forwarding decisions.

Quick Summary

Problems of Distribution Control Plane (more details later)

- **Complex/Cost:** all switches are required to run routing protocols
- **Hard to Change**: require updating all switches if want to upgrade an algorithm
- **Slow Innovation**: Not easy to evaluate new algorithms

Solution: From Distributed to Centralized Control

Reading Materials

- **More In-network Applications**
  - Congestion Control: HPCC: High Precision Congestion Control, SIGCOMM 2019
  - L4 Load Balancer: SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs, SIGCOMM 2017
  - KV Store: NetCache: Balancing Key-Value Stores with Fast In-Network Caching, SOSP 2017

OpenFlow: Enabling Innovation in Campus Networks

Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner

SIGCOMM CCR 2008

Outline

- **Background**
- Software Defined Networking
- The OpenFlow Protocol
- Using OpenFlow

New Trends

- **More complicated traffic patterns:**
  - A flurry of east-west machine-to-machine traffic before returning data to the end user device in the classic north-south traffic pattern.
  - Private/public cloud, resulting in additional traffic across the wide area network.
- **The complexity of IT:**
  - IT needs to accommodate various personal devices while protecting corporate data and intellectual property and meeting compliance mandates.

New Trends

- **The rise of cloud services (at that time: 2008):**
  - Elastic scaling of computing, storage, and network resources, ideally from a common viewpoint and with a common suite of tools.
  - Very few staff running very many machines
- **Big data (at that time: 2008)** means more bandwidth:
  - The rise of mega datasets is fueling a constant demand for additional network capacity in the data center.

Limitations of Current Networking Technologies

- **Complexity** that leads to **Slow Innovation**
- Protocols tend to be defined in isolation, with each solving a specific problem. This has resulted in one of the primary limitations of today's networks: **complexity**.
  - The **static nature of networks** is in stark contrast to the dynamic nature of today's server environment. Applications are distributed across VMs.
  - While existing networks can provide differentiated QoS levels for different applications, the provisioning of those resources is highly **manual**.

Limitations of Current Networking Technologies

- **Inconsistent policies**: To implement a network-wide policy, IT may have to configure thousands of devices and mechanisms.
  - Take hours.
  - Difficult to apply a consistent set of policies due to complexity.

Limitations of Current Networking Technologies

- **Inability to Scale:**
  - The network becomes vastly more complex, hundreds or thousands of network devices configured and managed.
  - Mega-operators, such as Google, Yahoo!, and Facebook, need **hyper-scale networks** that can provide high-performance, low-cost connectivity among a large number of physical servers.
    - Such scaling CANNOT be done with manual configuration.

Limitations of Current Networking Technologies

- **Vendor Dependence:**
  - Carriers and enterprises seek to deploy new capabilities and services in rapid response to changing business needs or user demands.
  - Vendors' equipment product cycle: 3 or more years.
  - Lack of standard, open interfaces limits the ability of network operators to tailor the network to their individual environments.

Outline

- Background
- **Software Defined Networking**
- The OpenFlow Protocol
- Using OpenFlow

Software Defined Networking

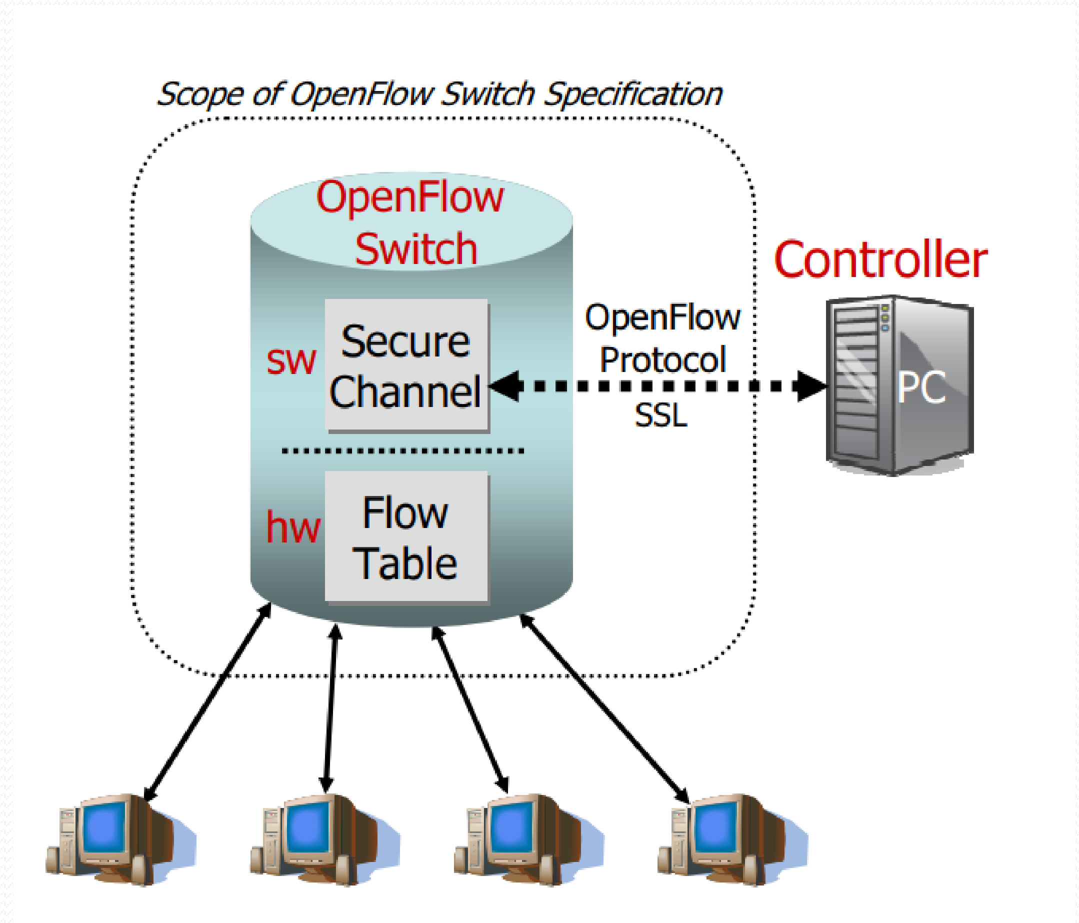
- Network control is physically **decoupled** from forwarding and is directly programmable.

Control plane

Data (forwarding) plane

Software Defined Networking

- **Centralized Controller(s)** can be used to break the distributed data plane.



Software Defined Networking

- Network operators and administrators can programmatically configure this simplified network abstraction **via centralized controller**
  - Write these programs themselves and not wait for features to be embedded in vendors' proprietary and closed software.
- SDN architectures **should support a set of APIs** that make it possible to implement common network services,
  - Routing, multicast, security, access control, bandwidth management, traffic engineering, quality of service, …, custom tailored to meet business objectives.

Overview

- Background
- Software Defined Networking
- **The OpenFlow Protocol**
- Using OpenFlow

OpenFlow Switches

- OpenFlow provides an open protocol to program the flow table in different switches and routers.
- An **OpenFlow Switch** consists of at least three parts
  - **A Flow Table**, with an action associated with each flow entry, to tell the switch how to process the flow.
  - **A Secure Channel** that connects the switch to a remote **Controller**, allowing commands and packets to be sent between a controller and the switch.
  - **The OpenFlow Protocol**, which provides an open and standard way for a controller to communicate with a switch.

OpenFlow Switches

- What is a flow?
  - A flow could be a TCP connection, or all packets from a particular MAC or IP address, or all packets with the same VLAN tag, or all packets from the same switch port.
  - A sequence of packets that can be identified with same features
- Each flow-entry has a simple **action** associated with it.
  - **Forward**: send this flow's packets out to a given port (or ports).
  - **Packet-In**: Report this flow's packets to a controller.
  - **Drop**: drop this flow's packets.
  - **Modify**: modify certain fields.

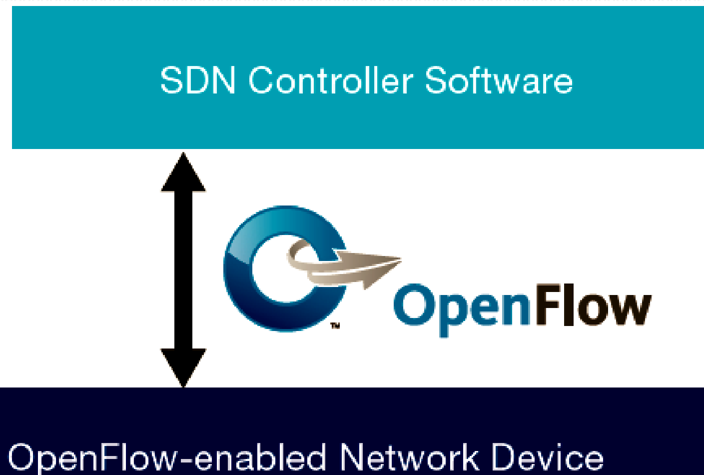
OpenFlow Switches

- An entry in the Flow-Table has three fields:
  - A packet **header** that defines the flow,
  - The **action**, which defines how the packets should be processed,
  - **Statistics**, which keep the number of packets and bytes for each flow, and the time since the last packet match
- A 10-tuple packet header

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

OpenFlow Controller

FIGURE 2
Example of OpenFlow
Instruction Set



OpenFlow-enabled Network Device

Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Benefit of OpenFlow-based SDN

- Centralized control of multi-vendor environments
- Reduced complexity through automation
- Higher rate of innovation
- Increased network reliability and security
  - Can ensure that access control, traffic engineering, quality of service, security, and other policies are enforced **consistently** across the wired and wireless network infrastructures,
- More granular network control
  - Per address block → per flow
- Better user experience
  - For example, automatic bandwidth detection video resolution adaption

Overview

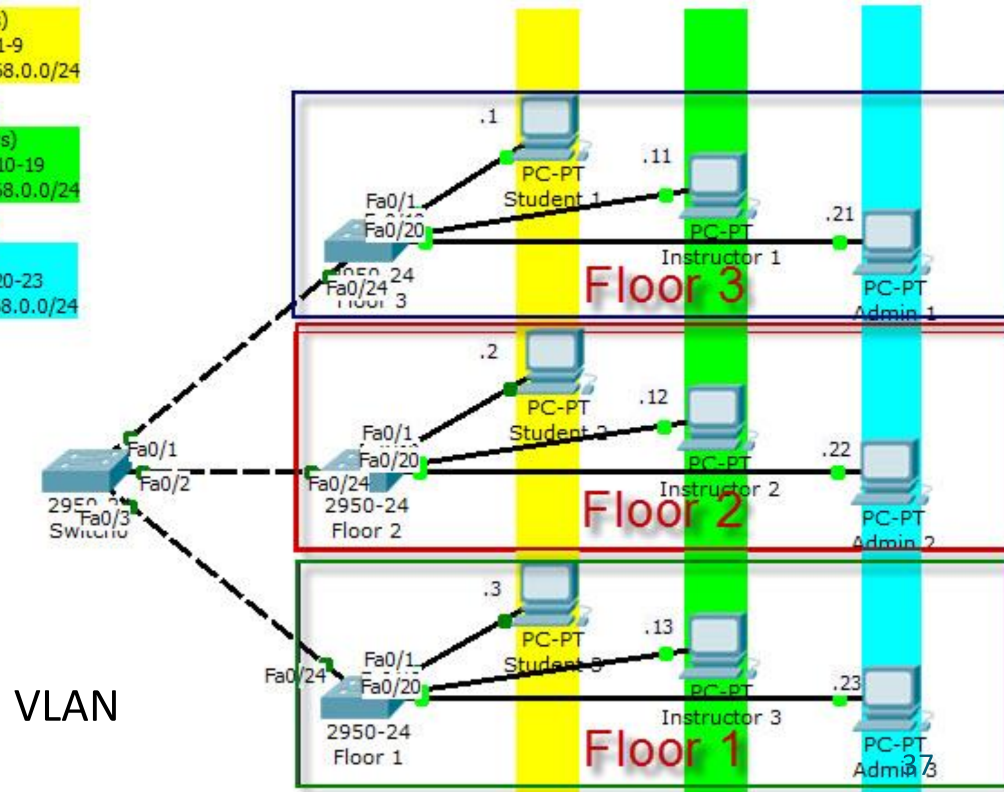
- Background
- Software Defined Networking
- The OpenFlow Protocol
- **Using OpenFlow**

Using OpenFlow

- **Example 1: Network Management and Access Control**
  - Ethane: The basic idea is to allow network managers to define a network-wide policy in the central controller, which is enforced directly by making admission control decisions for each new flow.
  - A controller associates packets with their senders by managing all the bindings between names and addresses—it essentially takes over DNS, DHCP and authenticates all users when they join, keeping track of which switch port (or access point) they are connected to.

Using OpenFlow

- **Example 2: VLANs**
  - The simplest approach is to statically declare a set of flows which specify the ports accessible by traffic on a given VLAN ID.
  - A more dynamic approach might use a controller to manage authentication of users and use the knowledge of the users' locations for tagging traffic at runtime.



VLAN

Using OpenFlow

- **Example 3: Mobile wireless VoIP clients**
  - Support call-handoff mechanism for WiFi-enabled phones
  - A controller is implemented to **track the location of clients**, re-routing connections — by **reprogramming the Flow Tables** —as users move through the network, allowing seamless handoff from one access point to another

Using OpenFlow

- **Example 4: A non-IP network**
  - process **any packet format** as long as the switch can **match fields in the packet header**
    - Ethernet headers
    - new IP version number
  - **Still reply on switch hardware/ASIC capacity**

Using OpenFlow

- **Example 5: Processing packets rather than flows**
  - Approach 1: Force all of a flow's packets to pass through a controller by default. More flexible, at the cost of performance
  - Approach 2: Route them to a programmable switch that does packet processing
    - NetFPGA-based programmable router/switch

Is OpenFlow Enough?

- **No! OpenFlow is not enough**
  - Still rely on Switch data plane functionality
- **Can we make data plane programmable too?**

P4: Programming Protocol-Independent Packet Processors

Pat Bosshartу, Dan Daly, Glen Gibbу, Martin Izzardу, Nick McKeownz, Jennifer Rexford, Cole Schlesinger, Dan Talaycoу, Amin Vahdat, George Varghesex, David Walker

SIGCOMM CCR 2014

Overview

- **Motivation**
- Abstract Forwarding Model
- A Programming Language
- An Example

Motivation

- Over the past five years, OpenFlow has grown increasingly more complicated

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

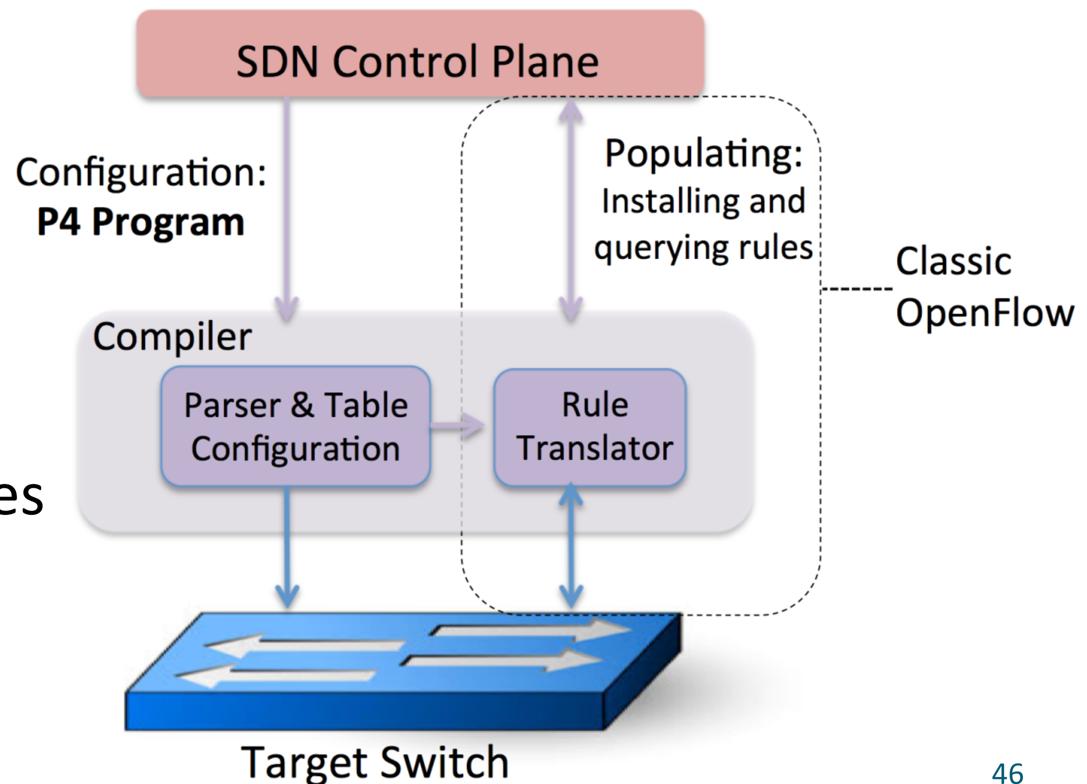
- The proliferation of new header fields shows no signs of stopping.

Motivation

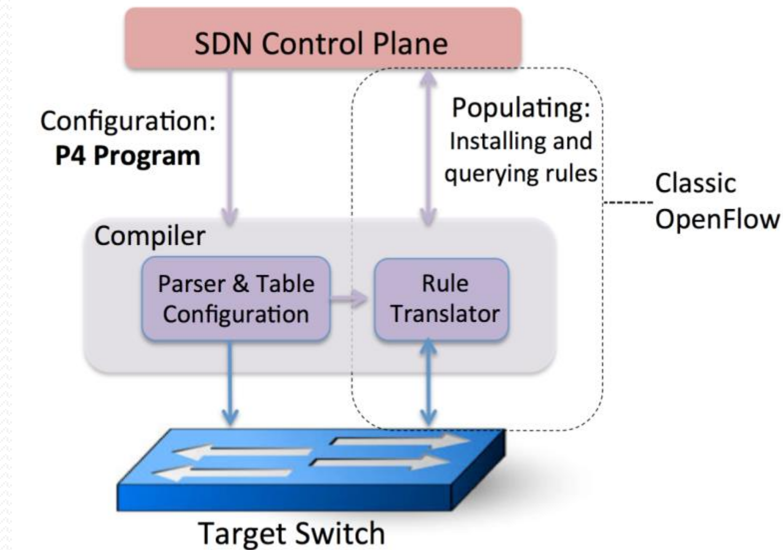
- Rather than repeatedly extending the OpenFlow specification, we argue that **future switches should support flexible mechanisms for parsing packets and matching header fields**, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new OpenFlow 2.0 API).
- Such a general, extensible approach would be simpler, more elegant, and more future-proof than today's OpenFlow 1.x standard.

Motivation

- P4: a higher-level language for Programming Protocol-independent Packet Processors
  - Configure a switch, telling it how packets are to be processed
  - Populate the forwarding tables in fixed function switches



Three Goals



- **Reconfigurable**
  - Redefine the packet parsing and processing in the field.
- **Protocol independent**
  - the controller should be able to specify
    - a packet parser for extracting header fields with particular names and types
    - a collection of typed match+action tables that process these header
- **Target independent**
  - Compiler should take the switch's capabilities into account when turning **a target-independent description** (written in P4) into **a target-dependent program**

Outline

- Motivation
- **Abstract Forwarding Model**
- A Programming Language
- An Example

Abstract Forwarding Model

- A programming parser
  - Allow **new headers** to be defined
- Multiple stages of **match+action**
  - In series, parallel, or combination of both
- Compare with OpenFlow
  - Fixed parser
  - Fixed series of actions

Abstract Forwarding Model

Switch Configuration

Parse Graph

Control Program

Table Config

Action Set

Forwarding rules

Forwarding rules

INPUT

PARSER

Match Action

Ingress Pipeline

Packet Mods + Egress Selection

BUFFER

Match Action

Egress Pipeline

Packet Mods

OUTPUT

Abstract Forwarding Model

- Two types of operations:
  - **Configure** operations program the parser, set the order of match+action stages, and specify the header fields processed by each stage.
    - Static, offline way
  - **Populate** operations add (and remove) entries to the match+action tables that were specified during configuration.
    - Runtime

Abstract Forwarding Model

- Arriving packets are first handled by the parser.
  - Recognize and extract fields from the header
- The extracted header fields are then passed to the match+action tables.
  - **Ingress** match+action table: determines the egress port(s) and determines the queue into which the packet is placed. The packet may be forwarded, replicated, dropped, or trigger flow control.
  - **Egress** match+action table: performs per-instance modifications to the packet header

Abstract Forwarding Model

- Packets can carry additional information between stages, called **metadata**, which is treated identically to packet header fields.
  - Example: inport, timestamp, …, etc
- Queueing: an action maps a packet to a queue

Outline

- Motivation
- Abstract Forwarding Model
- **A Programming Language**
- An Example

A Programming Language

- Dependencies between the fields
  - Determine which tables can be executed in parallel
  - **Table Dependency Graphs (TDG)**
  - TDG nodes map directly to match+action tables, and a dependency analysis identifies where each table may reside in the pipeline.

An Example Table Dependency Graph for an L2/L3 Switch

A Programming Language

- Two-step compilation
  - At the highest level, programmers express packet processing programs using an imperative language representing the control flow (P4);
  - Below this, a compiler translates the P4 representation to TDGs to facilitate dependency analysis and then maps the TDG to a specific switch target.

Outline

- Motivation
- Abstract Forwarding Model
- A Programming Language
- **An Example**

An Example

- Consider an example L2 network deployment with top-of-rack (ToR) switches at the edge connected by a two-tier core.
  - mTag: **a custom header used for switches to determine the port to send the packet**
- The routes through the core are encoded by a 32-bit tag composed of four single-byte fields.
  - Each core switch needs only to examine one byte of the tag and switch on that information.
  - The tag is added by the first edge switch.

P4 Concepts

- A P4 program contains the following key components
  - **Headers**: describes the sequence and structure of a series of fields.
  - **Parsers**: specify how to identify headers and valid header sequences within packets.
  - **Tables**: Match+action tables are the mechanism for performing packet processing.
  - **Actions**: Construction of complex actions from simpler protocol-independent primitives.
  - **Control Programs**: determine the order of match+action tables that are applied to a packet.

Header Formats

- An ordered list of names together with their width
- Ethernet and VLAN

```
header ethernet {
    fields {
        dst_addr : 48; // width in bits
        src_addr : 48;
        ethertype : 16;
    }
}

header vlan {
    fields {
        pcp : 3;
        cfi : 1;
        vid : 12;
        ethertype : 16;
    }
}
```

Header Formats

- The **mTag** header can be added without altering existing declarations.
- Each core switch is programmed with rules to examine one of these bytes determined by its location in the hierarchy and the direction of travel.
  - Output port on first and second levels up switch
  - Output port on first and second levels down switch

```
header mTag {
    fields {
        up1 : 8;
        up2 : 8;
        down1 : 8;
        down2 : 8;
        ethertype : 16;
    }
}
```

The Packet Parser

- P4 assumes the underlying switch can implement a state machine that traverses packet headers from start to finish, extracting field values as it goes.
- P4 describes this state machine directly as the set of transitions from one header to the next.

The Pac[...]

- Starts in th[...] explicit stop state is rea[...]untered

```
parser sta[...]
    ethern[...]
}

parser eth[...]
    switch[...]
        ca[...]
        ca[...]
        ca[...]
        // Other cases
    }
}
```

```
[...] {
    [...](ethertype) {
        [...]e 0xaaaa: mTag;
        [...]e  0x800: ipv4;
        // Other cases
```

```
[...] {
    switch(ethertype) {
        case 0x800: ipv4;
        // Other cases
    }
}
```

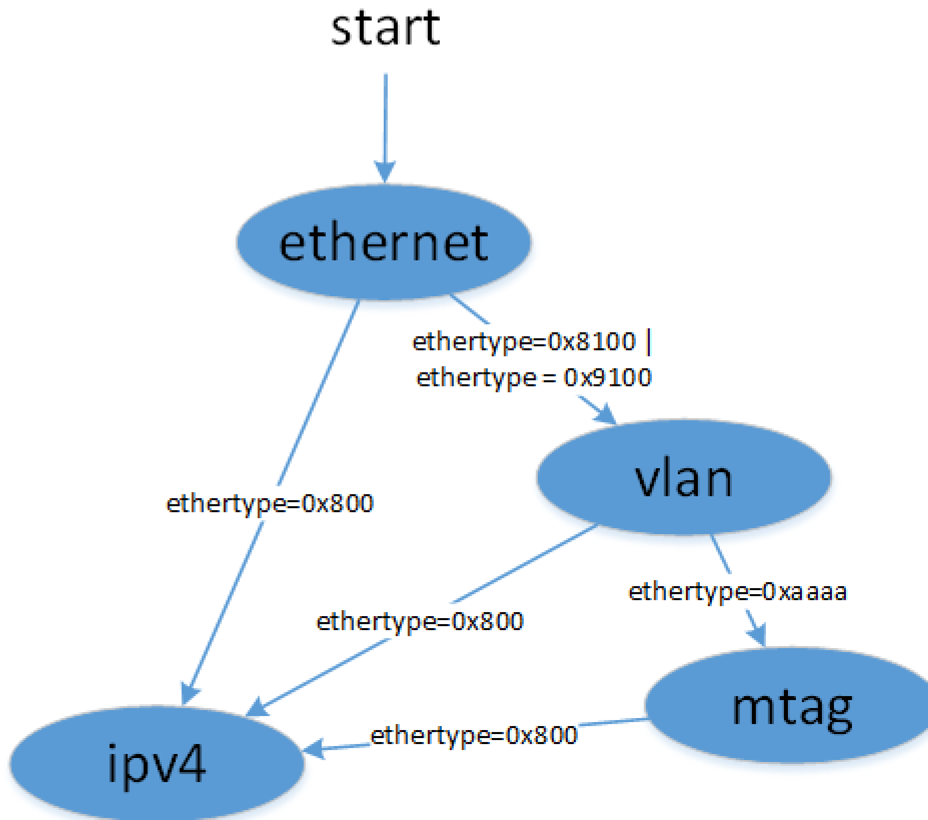


Table Specification

- The edge switch matches on the L2 destination and VLAN ID, and selects an mTag to add to the header.
  - The **reads** attribute declares which fields to match, qualified by the match type (exact, ternary, etc).
  - The **actions** attribute lists the possible actions which may be applied to a packet by the table.
  - The **max size** attribute specifies how many entries the table should support.
- The table specification allows a compiler to decide how much memory it needs, and the memory type (e.g., TCAM or SRAM) to implement the table.
- **Note that this is NOT the run-time packet processing logic.**

Table Specification

```
table mTag_table {
    reads {
        ethernet.dst_addr : exact;
        vlan.vid : exact;
    }
    actions {
        // At runtime, entries are programmed with params
        // for the mTag action.  See below.
        add_mTag;
    }
    max_size : 20000;
}
```

dest_addr	vlan.vid	Actions
10.0.1.0/24	1	add_mTag
0.0.0.0/0	0	none

Other Tables

```
table source_check {
    // Verify mtag only on ports to the core
    reads {
        mtag : valid; // Was mtag parsed?
        metadata.ingress_port : exact;
    }
    actions { // Each table entry specifies *one* action

        // If inappropriate mTag, send to CPU
        fault_to_cpu;

        // If mtag found, strip and record in metadata
        strip_mtag;

        // Otherwise, allow the packet to continue
        pass;
    }
    max_size : 64; // One rule per port
}
```

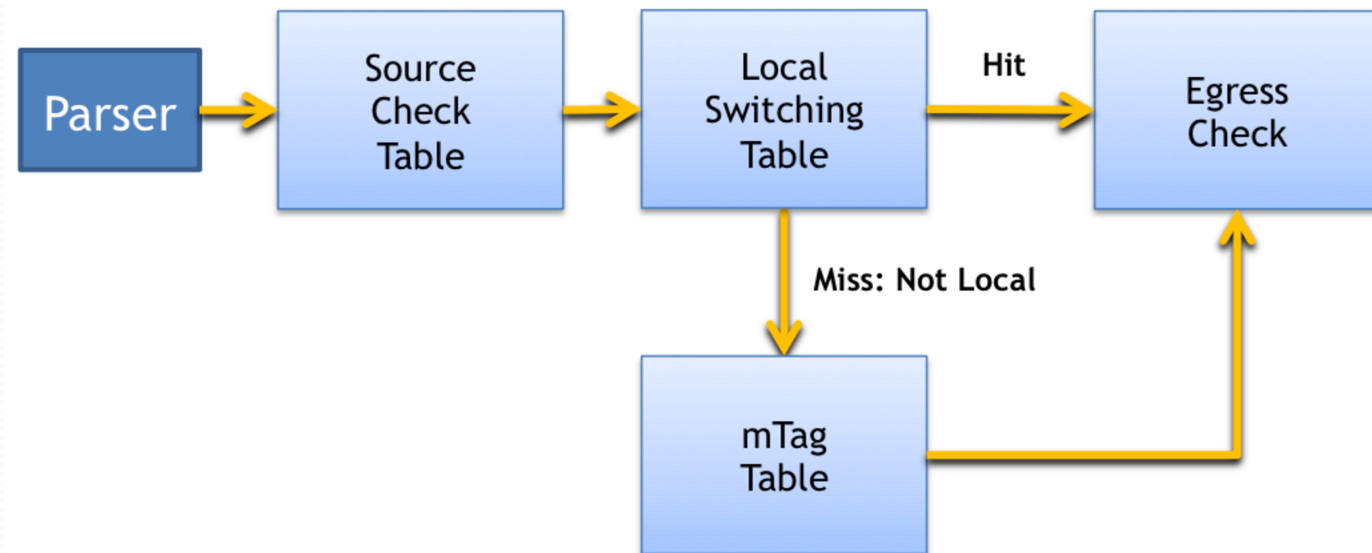
Other Tables

```
table local_switching {
    // Reads destination and checks if local
    // If miss occurs, goto mtag table.
}

table egress_check {
    // Verify egress is resolved
    // Do not retag packets received with tag
    // Reads egress and whether packet was mTagged
}
```

The Control Program

- Specify the flow of control from one table to the next.

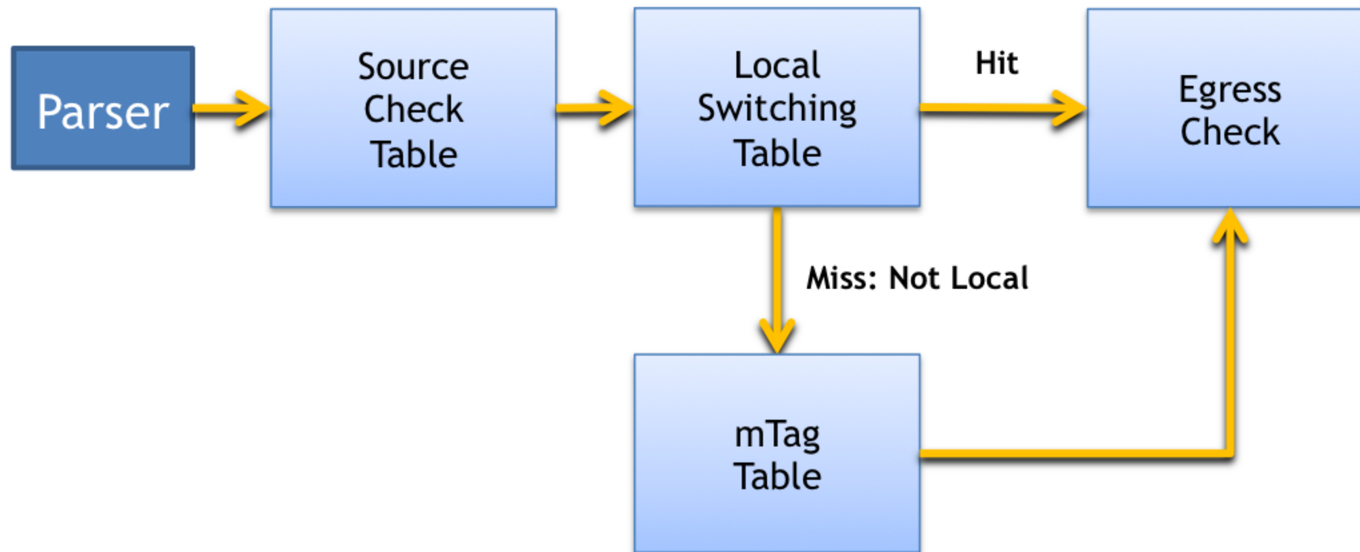


The Control Program

- The **source check table** verifies consistency between the received packet and the ingress port. It also strips mTags from the packet, recording whether the packet had an mTag in metadata.
- The **local switching table** is then executed. If this table misses, it indicates that the packet is not destined for a locally connected host.
- The **mTag table** is applied to the packet.
- Both local and core forwarding control can be processed by the **egress check table** which handles the case of an unknown destination by sending a notification up the SDN control stack.

The Control Program

```
control main() {
    // Verify mTag state and port are consistent
    table(source_check);
```




```
        // bad retagging with mTag.
        table(egress_check);
    }
}
```

Action Specification

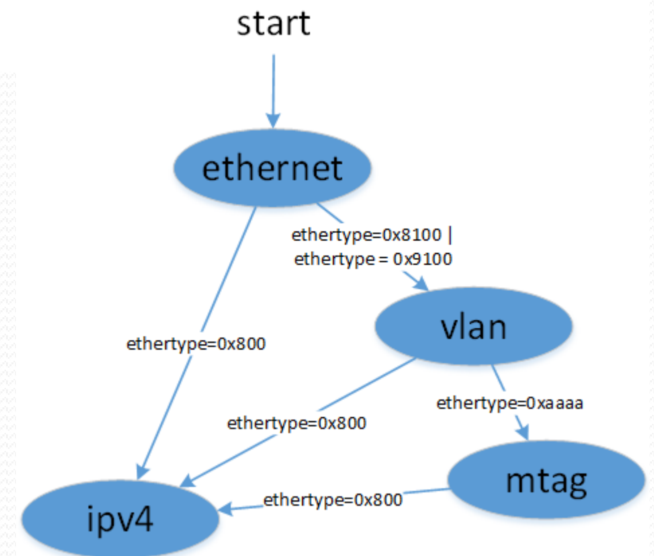
- If an action needs parameters (e.g., the up1 value for the mTag), it is supplied from the match table at **runtime**.
- P4's primitive actions include:
  - set field: Set a specific field in a header to a value.
  - copy field: Copy one field to another.
  - add header: Set a specific header instance (and all its fields) as valid.
  - remove header: Delete (“pop”) a header (and all its fields) from a packet.
  - increment: Increment or decrement the value in a field.
  - checksum: Calculate a checksum over some set of header fields (e.g., an IPv4 checksum).

Action Specification

- P4 defines a collection of primitive actions from which more complicated actions are built.

```
action add_mTag(up1, up2, down1, down2, egr_spec) {
    add_header(mTag);
    // Copy VLAN ethertype to mTag
    copy_field(mTag.ethertype, vlan.ethertype);
    // Set VLAN's ethertype to signal mTag
    set_field(vlan.ethertype, 0xaaaa);
    set_field(mTag.up1, up1);
    set_field(mTag.up2, up2);
    set_field(mTag.down1, down1);
    set_field(mTag.down2, down2);

    // Set the destination egress port as well
    set_field(metadata.egress_spec, egr_spec);
}
```



Compiling Control Program

- Control program: not explicitly call out dependencies between tables or opportunities for concurrency.
- Employ a compiler to analyze the control program to identify dependencies and look for opportunities to process header fields in parallel.
- Finally, the compiler generates the target configuration for the switch.

Compiling Control Program

- How mTag can be implemented on different kinds of switches
  - **Software switch**: The compiler directly maps the mTag table graph to switch tables.
  - **Hardware switches with RAM and TCAM**: perform efficient exact-matching using RAM, matches on a subset of bits with TCAM.
    - Barefoot Tofino chip
  - **Switches supporting parallel tables**: the tables mTag table and local switching can execute in parallel up to the execution of the action of setting an mTag.

Compiling Control Program

- **Switches that apply actions at the end of the pipeline**: In the mTag example, whether the mTag is added or removed could be represented in metadata.
- **Switches with a few tables**: Map a large number of P4 tables to a smaller number of physical tables. In the mTag example, the local switching could be combined with the mTag table.

Tofino Chip

- P4 programmable switch
  - Tofino chip
    - PISA switch architecture
    - 100GbE port speed
    - 25.6Tbps total bandwidth
    - 8 pipelines
    - SRAM: 1280 pages of 128b × 1k entries
    - TCAM: 384 blocks of 44b × 512 entries
- **All Tofino chips have not been discontinued…**
  - But some other vendors, such as Broadcom, are developing similar products

Review

- Background
  - Switch architecture
  - Control plane v.s. data plane
- Software-defined Networking
  - Control plane programmability
    - OpenFlow prototol
    - OpenFlow switch
  - Data plane programmability
    - P4 language