

Software-Defined 2 Networking and Advanced Data Plane Network Control Programming

Kai Gao

kaigao@scu.edu.cn

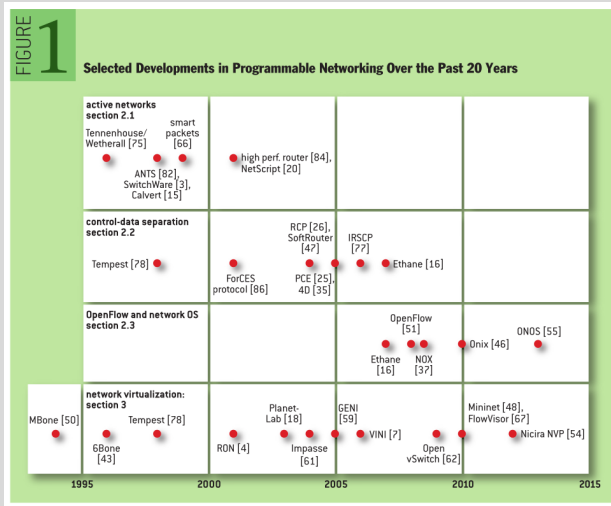
School of Cyber Science and Engineering
Sichuan University



Recap

A Brief History

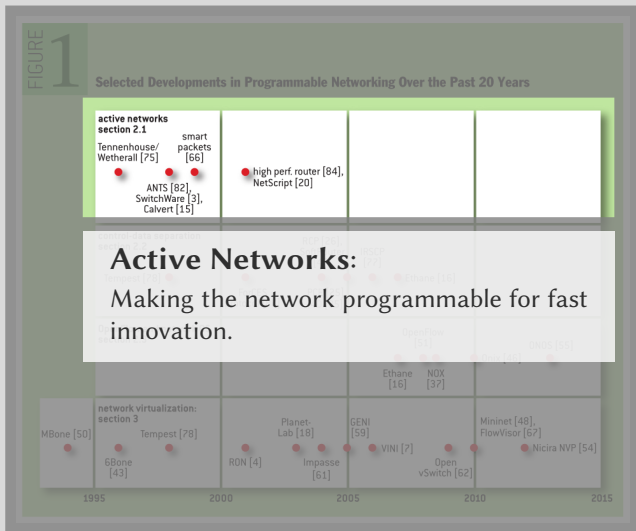
What makes the SDN today



Nick Feamster, Jennifer Rexford, and Ellen Zegura.
“The Road to SDN: An Intellectual History of Programmable Networks”. In: *Queue* 11.12 (Dec. 2013), 20:20–20:40. URL: <http://doi.acm.org/10.1145/2559899.2560327>

A Brief History

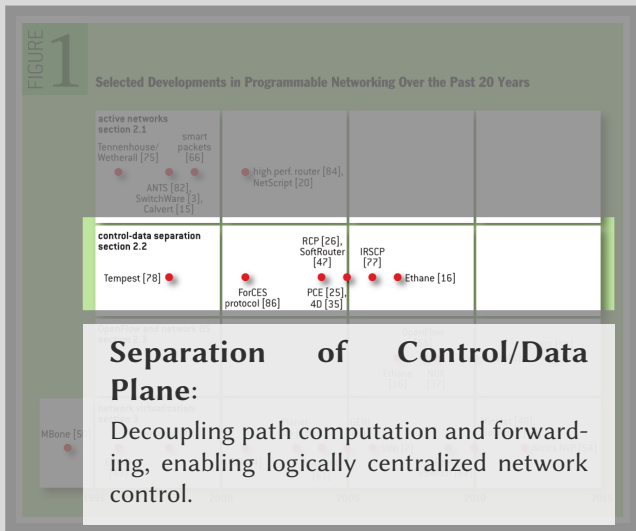
What makes the SDN today



Nick Feamster, Jennifer Rexford, and Ellen Zegura.
“The Road to SDN: An Intellectual History of Programmable Networks”. In: *Queue* 11.12 (Dec. 2013), 20:20–20:40. URL: <http://doi.acm.org/10.1145/2559899.2560327>

A Brief History

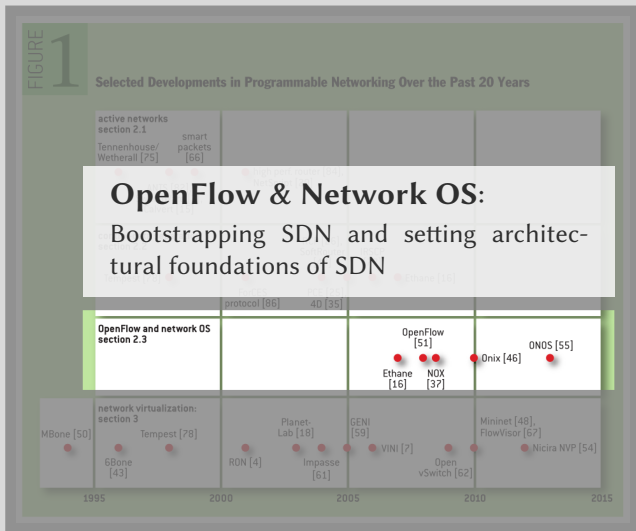
What makes the SDN today



Nick Feamster, Jennifer Rexford, and Ellen Zegura.
“The Road to SDN: An Intellectual History of Programmable Networks”. In: *Queue* 11.12 (Dec. 2013), 20:20–20:40. URL: <http://doi.acm.org/10.1145/2559899.2560327>

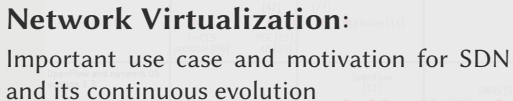
A Brief History

What makes the SDN today



Nick Feamster, Jennifer Rexford, and Ellen Zegura.
“The Road to SDN: An Intellectual History of Programmable Networks”. In: *Queue* 11.12 (Dec. 2013), 20:20–20:40. URL: <http://doi.acm.org/10.1145/2559899.2560327>

What makes the SDN today



Nick Feamster, Jennifer Rexford, and Ellen Zegura.
 “The Road to SDN: An Intellectual History of
 Programmable Networks”. In: *Queue* 11.12 (Dec.
 2013), 20:20–20:40. URL: [http:
 //doi.acm.org/10.1145/2559899.2560327](http://doi.acm.org/10.1145/2559899.2560327)

SDN as a Network Architecture

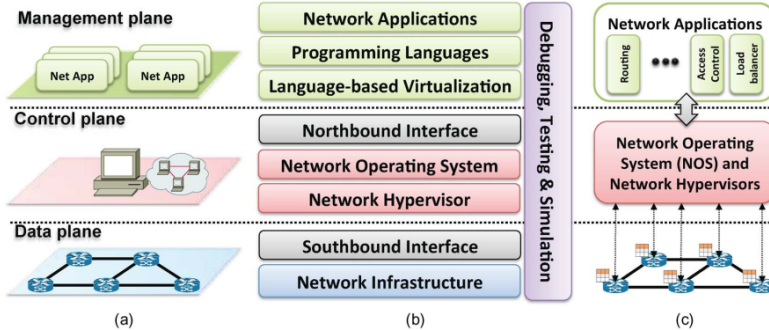


Fig. 6. Software-Defined Networks in (a) planes, (b) layers, and (c) system design architecture.

D. Kreutz et al. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76

SDN in Action

- Industry usage: VMWare/Nicira, Google
- Cloud platform: OpenStack
- Scientific research: CERN, GENI, CENI

本期学习目标

- 现有网络协议中有哪些常见的数据平面？

本期学习目标

- 现有网络协议中有哪些常见的数据平面？
- SDN 的数据平面相对现有数据平面设计在哪些方面进行了改进？

本期学习目标

- 现有网络协议中有哪些常见的数据平面？
- SDN 的数据平面相对现有数据平面设计在哪些方面进行了改进？
- OpenFlow 协议为什么要采取多表方式？

本期学习目标

- 现有网络协议中有哪些常见的数据平面？
- SDN 的数据平面相对现有数据平面设计在哪些方面进行了改进？
- OpenFlow 协议为什么要采取多表方式？
- OpenFlow 协议流表 (flow table) 具有哪些基本结构？流表匹配原理是什么？

本期学习目标

- 现有网络协议中有哪些常见的数据平面？
- SDN 的数据平面相对现有数据平面设计在哪些方面进行了改进？
- OpenFlow 协议为什么要采取多表方式？
- OpenFlow 协议流表 (flow table) 具有哪些基本结构？流表匹配原理是什么？
- OpenFlow 协议的组表 (group table) 具有哪些基本结构？如何选择最终的行为？

本期学习目标

- 现有网络协议中有哪些常见的数据平面？
- SDN 的数据平面相对现有数据平面设计在哪些方面进行了改进？
- OpenFlow 协议为什么要采取多表方式？
- OpenFlow 协议流表 (flow table) 具有哪些基本结构？流表匹配原理是什么？
- OpenFlow 协议的组表 (group table) 具有哪些基本结构？如何选择最终的行为？
- OpenFlow 协议的计量表 (meter table) 具有哪些基本结构？如何选择最终的行为？

Overview of SDN Data Plane

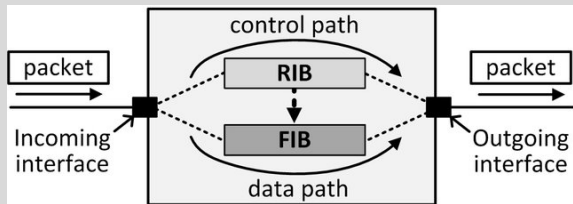
Data Plane in Networking
Data Plane of SDN
Hardware for Data Plane

IP

The data plane (数据平面) of IP is
Forwarding Information Base (FIB, 转发表).

RFC 4098 (Lepp et al. 2005):

*... At minimum, this contains the **interface identifier** and **next hop** information for each reachable **destination network prefix**... It is a **data plane construct** and is used for the forwarding of each packet ...*



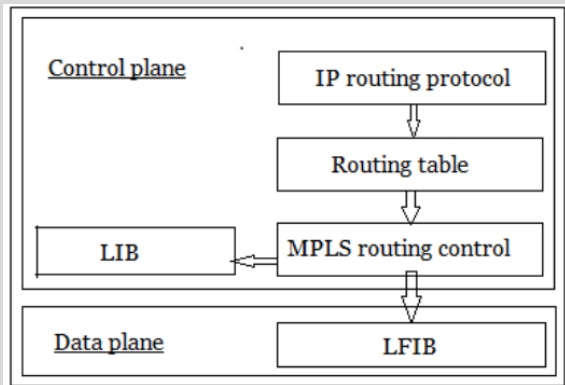
Miguel Elias M. Campista et al. "Challenges and Research Directions for the Future Internetworking". In: *IEEE Commun. Surv. Tutorials* 16.2 (Sum. 2014), pp. 1050–1079.
URL: <http://ieeexplore.ieee.org/document/6644748/> (visited on 09/04/2021)

MPLS

The data plane of Multiprotocol Label Switch (MPLS, 多协议标签交换) is **Label Forwarding Information Base (LFIB, 标签转发表)**.

RFC 3031 (Viswanathan, Rosen, and Callon 2001):

... (label swap is) the basic forwarding operation consisting of looking up an incoming label to determine the outgoing label, encapsulation, port, and other data handling information.



Madhulika Bhandure and Gaurang Deshmukh. "Comparative Analysis of Mpls and Non -Mpls Network". In: 3.4 (2013), p. 6

ForCES

In Forwarding and Control Element Separation (ForCES), the data plane is **Forwarding Element (FE)**.

RFC 5810 (Halpern et al. 2010)

FEs use the underlying hardware to provide per-packet processing and handling as directed/controlled by one or more CEs via the ForCES protocol.

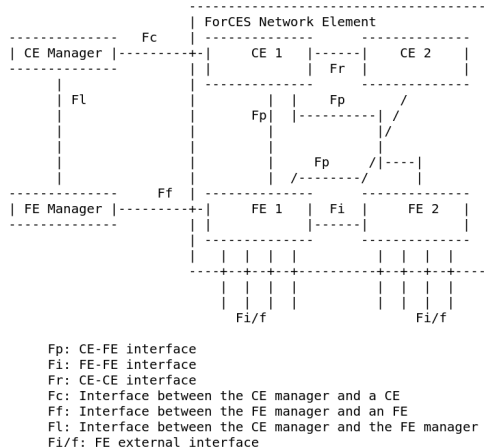


Figure 1: ForCES Architectural Diagram

Joel M. Halpern et al. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. RFC 5810. Mar. 2010. URL: <https://rfc-editor.org/rfc/rfc5810.txt>

Summary

What is Data Plane?

Data plane is

- software or **optimized hardware**

Summary

What is Data Plane?

Data plane is

- software or optimized hardware
- for packet processing and handling
(forwarding, encapsulation, label swapping, queuing, ECMP, etc.)

Summary

What is Data Plane?

Data plane is

- software or optimized hardware
- for packet processing and handling (forwarding, encapsulation, label swapping, queuing, ECMP, etc.)
- based on header fields (IP destination, MPLS label, etc.)

Summary

What is Data Plane?

Data plane is

- software or optimized hardware
- for packet processing and handling (forwarding, encapsulation, label swapping, queuing, ECMP, etc.)
- based on header fields (IP destination, MPLS label, etc.)
- directed by control plane

Summary

What is Data Plane?

Data plane is

- software or optimized hardware
- for packet processing and handling (forwarding, encapsulation, label swapping, queuing, ECMP, etc.)
- based on header fields (IP destination, MPLS label, etc.)
- directed by control plane

→ **Action (行为)**

Summary

What is Data Plane?

Data plane is

- software or optimized hardware
- for packet processing and handling (forwarding, encapsulation, label swapping, queuing, ECMP, etc.)
- based on header fields (IP destination, MPLS label, etc.)
- directed by control plane

→ **Action** (行为)

→ **Match** (匹配项)

Summary

What is Data Plane?

Data plane is

- software or optimized hardware
- for packet processing and handling (forwarding, encapsulation, label swapping, queuing, ECMP, etc.)
- based on header fields (IP destination, MPLS label, etc.)
- directed by control plane

————→ **Action** (行为)

————→ **Match** (匹配项)

————→ **Southbound Interface**
(南向接口)

SDN Data Plane

Functionality

<i>Architecture</i>	<i>Match</i>	<i>Action</i>	<i>Southbound</i>
IP	destination IP address	forward to egress port, etc.	FIB
MPLS	label	modify label, forward to egress port, etc.	LFIB
ForCES	depending on logical function type	forwarding, QoS, filtering, tunnel, sampling, etc.	ForCES protocol

SDN Data Plane

Functionality

<i>Architecture</i>	<i>Match</i>	<i>Action</i>	<i>Southbound</i>
IP	destination IP address	forward to egress port, etc.	FIB
MPLS	label	modify label, forward to egress port, etc.	LFIB
ForCES	depending on logical function type	forwarding, QoS, filtering, tunnel, sampling, etc.	ForCES protocol
OpenFlow	multiple protocol header fields	forwarding, QoS, filtering, modify header, etc.	OpenFlow protocol

SDN Data Plane

Functionality

<i>Architecture</i>	<i>Match</i>	<i>Action</i>	<i>Southbound</i>
IP	destination IP address	forward to egress port, etc.	FIB
MPLS	label	modify label, forward to egress port, etc.	LFIB
ForCES	depending on logical function type	forwarding, QoS, filtering, tunnel, sampling, etc.	ForCES protocol
OpenFlow	multiple protocol header fields	forwarding, QoS, filtering, modify header, etc.	OpenFlow protocol
PoF	header segments & flow metadata	forwarding, simple math, modify header/metadata	OpenFlow

SDN Data Plane

Functionality

<i>Architecture</i>	<i>Match</i>	<i>Action</i>	<i>Southbound</i>
IP	destination IP address	forward to egress port, etc.	FIB
MPLS	label	modify label, forward to egress port, etc.	LFIB
ForCES	depending on logical function type	forwarding, QoS, filtering, tunnel, sampling, etc.	ForCES protocol
OpenFlow	multiple protocol header fields	forwarding, QoS, filtering, modify header, etc.	OpenFlow protocol
PoF	header segments & flow metadata	forwarding, simple math, modify header/metadata	OpenFlow
P4	customizable header fields & metadata	customizable actions based on forwarding, simple math, modify header/metadata	P4 runtime

SDN Data Plane

Data Plane Characteristics

<i>Architecture</i>	<i>Match</i>	<i>Action</i>
IP	limited, fixed	limited, fixed
MPLS	limited, fixed	less limited, fixed
ForCES	less limited, fixed	flexible, fixed
OpenFlow	flexible, fixed	flexible, fixed
PoF	flexible, customizable	flexible, fixed
P4	flexible, customizable	flexible, customizable

Hardware for Data Plane

Two types of memory (used to realize look-up tables):

- Content Addressable Memory (CAM, 内容寻址内存): used to realize the “look-up” operation
- Random Access Memory (RAM): used to store state (meta data, actions, etc.)

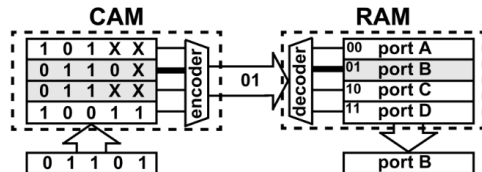


Fig. 3. CAM-based implementation of the routing table of Table I.

K. Pagiamtzis and A. Sheikholeslami. “Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey”. In: *IEEE J. Solid-State Circuits* 41.3 (Mar. 2006), pp. 712–727. URL: <http://ieeexplore.ieee.org/document/1599540/> (visited on 09/05/2021)

Content Addressable Memory (CAM)

Logical functionality of CAM: compare the **search word** with each **stored word** in parallel and return **the index with the highest priority**.

Two types of CAM:

- **CAM:** used to realize **exact** matching (精确匹配)
- **TCAM** (Ternary CAM, 三态内容寻址内存): used to realize **longest prefix** matching (最长前缀匹配) or more generally **wildcard** matching (模糊匹配)

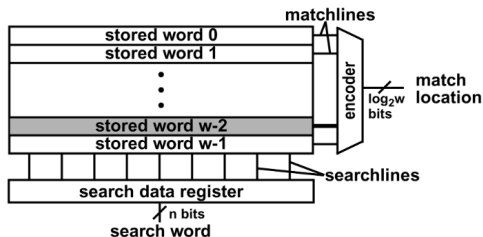


Fig. 1. Conceptual view of a content-addressable memory containing w words. In this example, the search word matches location $(w - 2)$ as indicated by the shaded box. The matchlines provide the row match results. The encoder outputs an encoded version of the match location using $\log_2 w$ bits.

K. Pagiamtzis and A. Sheikholeslami. "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey". In: *IEEE J. Solid-State Circuits* 41.3 (Mar. 2006), pp. 712–727. URL: <http://ieeexplore.ieee.org/document/1599540/> (visited on 09/05/2021)

CAM v.s. TCAM

CAM and TCAM have **similar structure** but **different cell circuits** (hence **different matching logic**):

- CAM: stored value can be either 0 or 1
- TCAM: stored value can be 0, 1 or x

	search bit	
	0	1
stored bit		
0	T	F
1	F	T
x	T	T

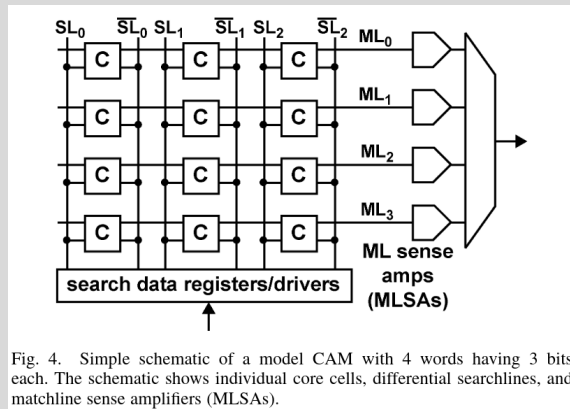


Fig. 4. Simple schematic of a model CAM with 4 words having 3 bits each. The schematic shows individual core cells, differential searchlines, and matchline sense amplifiers (MLSAs).

K. Pagiamtzis and A. Sheikholeslami. "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey". In: *IEEE J. Solid-State Circuits* 41.3 (Mar. 2006), pp. 712–727. URL: <http://ieeexplore.ieee.org/document/1599540/> (visited on 09/05/2021)

TCAM Example

Realizing IPv4 **longest prefix matching** with TCAM:

- For each IPv4 prefix $a.b.c.d/p$, add a TCAM entry
- stored bits:
 - upper p bits: same as in $a.b.c.d$;
 - lower $32 - p$ bits: x or *
- priority (larger means more prioritized):
 p

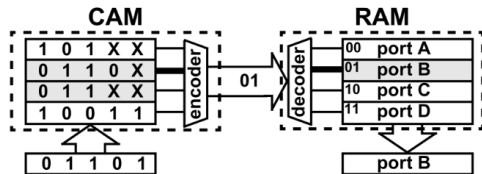


Fig. 3. CAM-based implementation of the routing table of Table I.

K. Pagiamtzis and A. Sheikholeslami. "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey". In: *IEEE J. Solid-State Circuits* 41.3 (Mar. 2006), pp. 712–727. URL: <http://ieeexplore.ieee.org/document/1599540/> (visited on 09/05/2021)

TCAM Example

Realizing IPv4 **longest prefix matching** with TCAM:

- For each IPv4 prefix $a.b.c.d/p$, add a TCAM entry
- stored bits:
 - upper p bits: same as in $a.b.c.d$;
 - lower $32 - p$ bits: x or *
- priority (larger means more prioritized):
 p

Example: $192.168.1.1/24 = 1100\ 0000\ 1010\ 1000\ 0000\ 0001\ 0000\ 0001/24$

Stored bits: $1100\ 0000\ 1010\ 1000\ 0000\ 0001\ xxxx\ xxxx$ **Priority:** 24

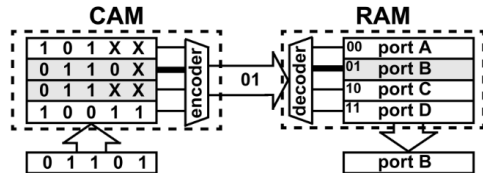


Fig. 3. CAM-based implementation of the routing table of Table I.

K. Pagiamtzis and A. Sheikholeslami. "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey". In: *IEEE J. Solid-State Circuits* 41.3 (Mar. 2006), pp. 712–727. URL: <http://ieeexplore.ieee.org/document/1599540/> (visited on 09/05/2021)

CAM v.s. TCAM

Cost Matters

Why not always use TCAM?

- Price: TCAM > CAM
- Power: TCAM > CAM
- Area: TCAM > CAM

Cisco documentation:

... CAM is most useful for building tables that search on exact matches such as MAC address tables ...

... TCAM is most useful for building tables for searching on longest matches such as IP routing tables organized by IP prefixes...

Thamaraiselvam Santhanam. CAM (Content Addressable Memory) VS TCAM (Ternary Content Addressable Memory). Mar. 25, 2011. URL: <https://community.cisco.com/t5/networking-documents/cam-content-addressable-memory-vs-tcam-ternary-content/ta-p/3107938> (visited on 09/05/2021)

Summary

- Data plane in networking typically follow a “match-action” paradigm

Summary

- Data plane in networking typically follow a “match-action” paradigm
- SDN data plane improves the flexibility in both the “match” part and the “action” part

Summary

- Data plane in networking typically follow a “match-action” paradigm
- SDN data plane improves the flexibility in both the “match” part and the “action” part
- Efficient data plane is built with specific hardware CAM & TCAM

OpenFlow

OpenFlow Data Plane
OpenFlow Flow Table
Communication Protocol

Overview

OpenFlow specification is standardized by Open Networking Foundation (ONF)

OpenFlow specification specifies

- **components** (组成单元)



OpenFlow Switch Specification

Version 1.0.0 (Wire Protocol 0x01)

December 31, 2009

ONF TS-001

ONF. *OpenFlow Switch Specification (Version 1.0.0)*. Open Networking Foundation, 2009

Overview

OpenFlow specification is standardized by Open Networking Foundation (ONF)

OpenFlow specification specifies

- components (组成单元)
- **flow table structure and functionalities**
(流表结构及功能)



OpenFlow Switch Specification

Version 1.0.0 (Wire Protocol 0x01)

December 31, 2009

ONF TS-001

ONF. *OpenFlow Switch Specification (Version 1.0.0)*. Open Networking Foundation, 2009

Overview

OpenFlow specification is standardized by Open Networking Foundation (ONF)

OpenFlow specification specifies

- components (组成单元)
- flow table structure and functionalities (流表结构及功能)
- communication protocol (通信协议).



OpenFlow Switch Specification

Version 1.0.0 (Wire Protocol 0x01)

December 31, 2009

ONF TS-001

ONF. *OpenFlow Switch Specification (Version 1.0.0)*. Open Networking Foundation, 2009

OpenFlow Specification: History

Major version history:

- 1.0.0: December 31, 2009
 - 1.0.2: November 1, 2013
- 1.1.0: February 28, 2011
- 1.2.0: December 5, 2011
- 1.3.0: June 25, 2012
 - 1.3.5: March 26, 2015
- 1.4.0: October 14, 2013
 - 1.4.1: March 26, 2015
- 1.5.0: December 19, 2014
 - 1.5.1: March 26, 2015

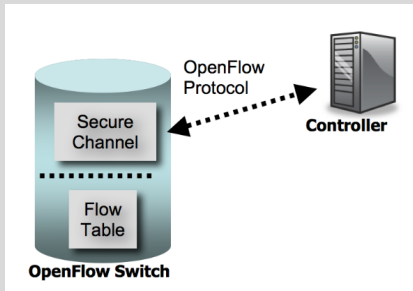
OpenFlow Specification: History

Major version history:

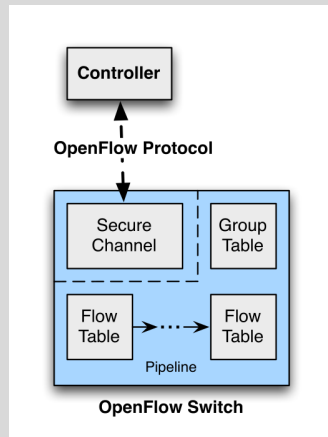
- 1.0.0: December 31, 2009
1.0.2: November 1, 2013
- 1.1.0: February 28, 2011
- 1.2.0: December 5, 2011
- 1.3.0: June 25, 2012
1.3.5: March 26, 2015
- 1.4.0: October 14, 2013
1.4.1: March 26, 2015
- 1.5.0: December 19, 2014
1.5.1: March 26, 2015

Components

In OF 1.0.0, an OpenFlow switch has a single flow table (BOTTOM). Since OF 1.1.0, an OpenFlow switch have multiple flow tables (maximum 256).



ONF. *OpenFlow Switch Specification (Version 1.0.0)*. Open Networking Foundation, 2009



ONF. *OpenFlow Switch Specification (Version 1.1.0)*. Open Networking Foundation, 2011

Why Multiple Tables?

The **cross-product** effect (叉积效应) for compound matches

Example (from (ONF 2015b)):

- Set VLAN ID based on ingress port
- Forward to egress port based on MAC address

Assume we have 48 ports and 10K MAC addresses

Why Multiple Tables?

The **cross-product** effect (叉积效应) for compound matches

Example (from (ONF 2015b)):

- Set VLAN ID based on ingress port
- Forward to egress port based on MAC address

Assume we have 48 ports and 10K MAC addresses

Single-table implementation:

<i>port</i>	<i>dst_mac</i>	<i>actions</i>	} $48 \times 10K = 480K$
1	m_1	$\text{set_vlan}(v_1), \text{output}(e_1)$	
1	m_2	$\text{set_vlan}(v_1), \text{output}(e_2)$	
\vdots	\vdots	\vdots	
1	m_N	$\text{set_vlan}(v_1), \text{output}(e_N)$	
2	m_1	$\text{set_vlan}(v_2), \text{output}(e_1)$	
\vdots	\vdots	\vdots	
K	m_N	$\text{set_vlan}(v_K), \text{output}(e_N)$	

Why Multiple Tables?

The **cross-product** effect (叉积效应) for compound matches

Example (from (ONF 2015b)):

- Set VLAN ID based on ingress port
- Forward to egress port based on MAC address

Assume we have 48 ports and 10K MAC addresses

Single-table implementation:

<i>port</i>	<i>dst_mac</i>	<i>actions</i>	} $48 \times 10K = 480K$
1	m_1	$\text{set_vlan}(v_1), \text{output}(e_1)$	
1	m_2	$\text{set_vlan}(v_1), \text{output}(e_2)$	
\vdots	\vdots	\vdots	
1	m_N	$\text{set_vlan}(v_1), \text{output}(e_N)$	
2	m_1	$\text{set_vlan}(v_2), \text{output}(e_1)$	
\vdots	\vdots	\vdots	
K	m_N	$\text{set_vlan}(v_K), \text{output}(e_N)$	

Multiple-table implementation:

48	<i>port</i>	<i>actions</i>	\times	<i>dst_mac</i>	<i>actions</i>	} 10K
	1	$\text{set_vlan}(v_1)$		m_1	$\text{output}(e_1)$	
	2	$\text{set_vlan}(v_2)$		m_2	$\text{output}(e_2)$	
	\vdots	\vdots		\vdots	\vdots	
	K	$\text{set_vlan}(v_K)$		m_N	$\text{output}(e_N)$	

Total = 48 + 10K = 10,048

Why Multiple Tables?

The **cross-product** effect (叉积效应) for compound matches

Example (from (ONF 2015b)):

- Set VLAN ID based on ingress port
- Forward to egress port based on MAC address

Assume we have 48 ports and 10K MAC addresses

Using multiple tables can **substantially reduce the number of entries to realize the same functionality.**

Single-table implementation:

port	dst_mac	actions	} $48 \times 10K = 480K$
1	m_1	set_vlan(v_1), output(e_1)	
1	m_2	set_vlan(v_1), output(e_2)	
\vdots	\vdots	\vdots	
1	m_N	set_vlan(v_1), output(e_N)	
2	m_1	set_vlan(v_2), output(e_1)	
\vdots	\vdots	\vdots	
K	m_N	set_vlan(v_K), output(e_N)	

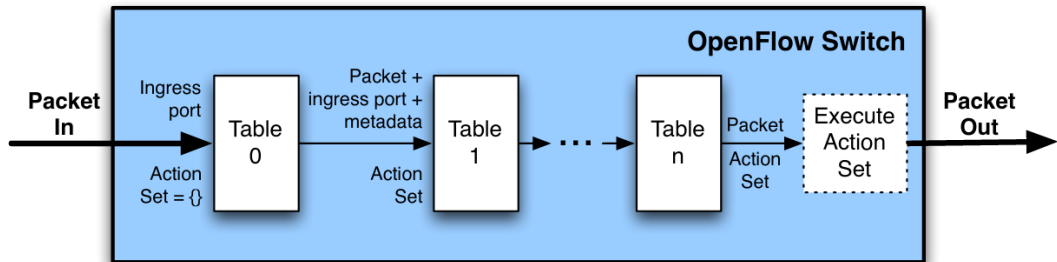
Multiple-table implementation:

48	port	actions	\times	dst_mac	actions	} 10K
	1	set_vlan(v_1)		m_1	output(e_1)	
	2	set_vlan(v_2)		m_2	output(e_2)	
	\vdots	\vdots		\vdots	\vdots	
	K	set_vlan(v_K)		m_N	output(e_N)	

Total = 48 + 10K = 10,048

OpenFlow Pipeline

Multiple tables **CAN** be used to create a pipeline (流水线)



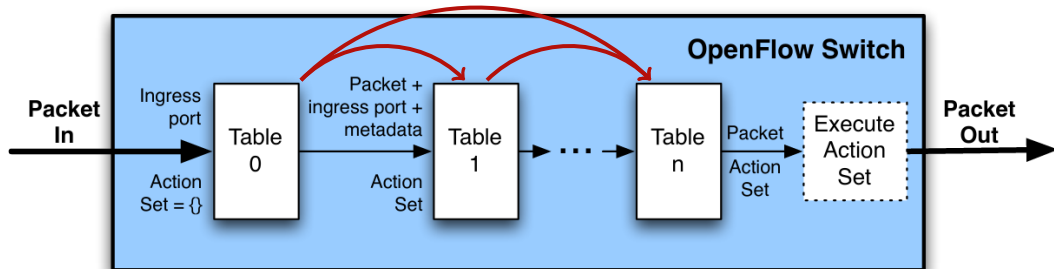
(a) Packets are matched against multiple tables in the pipeline

ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2015, p. 177. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/05/2021)

OpenFlow Pipeline

- Only jump forward
- Must be explicitly specified

Multiple tables **CAN** be used to create a pipeline (流水线)



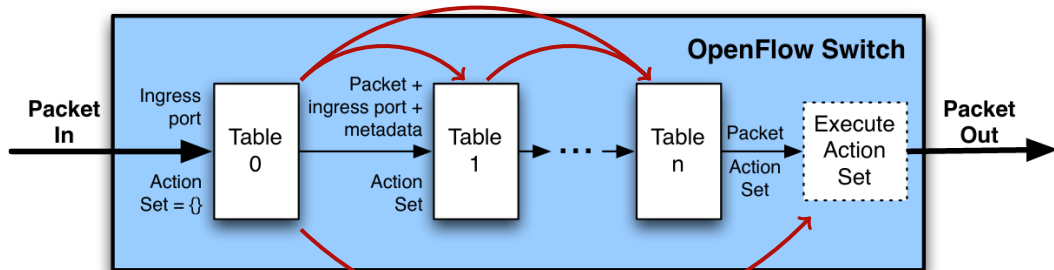
(a) Packets are matched against multiple tables in the pipeline

ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2015, p. 177. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/05/2021)

OpenFlow Pipeline

Multiple tables **CAN** be used to create a pipeline (流水线)

- Only jump forward
- Must be explicitly specified



(a) Packets are matched against multiple tables in the pipeline

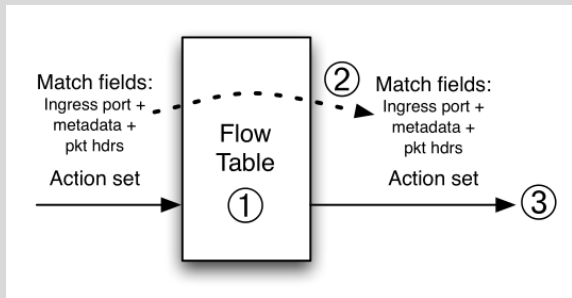
ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2014. <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/03/2021)

- Otherwise send for action execution

Packet Processing in a Single Table

Each OpenFlow flow table (流表) can

- ① Find the entry with the highest priority based on matching fields, metadata, ingress port

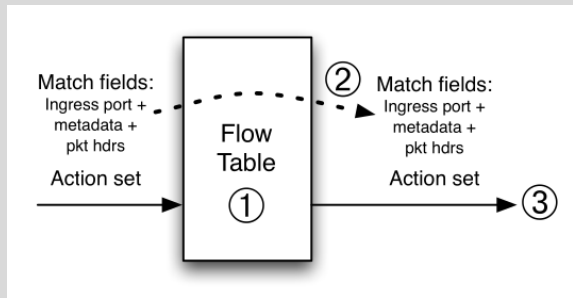


ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2015, p. 177. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/05/2021)

Packet Processing in a Single Table

Each OpenFlow flow table (流表) can

- ① Find the entry with the highest priority based on matching fields, metadata, ingress port
- ② Execute instructions
 - i Modify packet and/or the match fields (apply action)
 - ii Update action set (clear/write action)
 - iii Set next table (goto table)

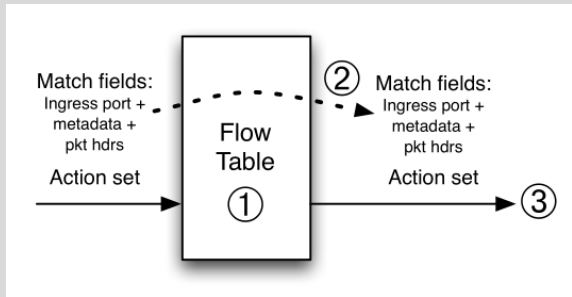


ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2015, p. 177. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/05/2021)

Packet Processing in a Single Table

Each OpenFlow flow table (流表) can

- ① Find the entry with the highest priority based on matching fields, metadata, ingress port
- ② Execute instructions
 - i Modify packet and/or the match fields (**apply action**)
 - ii Update action set (**clear/write action**)
 - iii Set next table (**goto table**)
- ③ Send to next table, or terminate the pipeline and carry out the action set



ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2015, p. 177. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/05/2021)

Flow Entry

Flow entry (流表项) is the basic unit in a flow table:

- **match fields & priority:** determine whether a packet will be matched by the rule
- **counters & instructions:** statistics and “actions” when a packet is matched by the rule
- **timeout, cookie & flag:** used for flow entry management

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags
--------------	----------	----------	--------------	----------	--------	-------

Table 1: Main components of a flow entry in a flow table.

ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2015, p. 177. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/05/2021)

Match Fields

OpenFlow 1.3 supports 40 matching fields (匹配域, 36 header fields and 4 metadata)

- Support multiple common protocols: Ethernet, VLAN, IPv4, IPv6, TCP, UDP, ICMP, ARP, MPLS
- Ingress port, physical port, etc.

Not all fields are supported by hardware OpenFlow switches

Field		Description
OXM_OF_IN_PORT	<i>Required</i>	Ingress port. This may be a p
OXM_OF_ETH_DST	<i>Required</i>	Ethernet destination address.
OXM_OF_ETH_SRC	<i>Required</i>	Ethernet source address. Can
OXM_OF_ETH_TYPE	<i>Required</i>	Ethernet type of the OpenFlo
OXM_OF_IP_PROTO	<i>Required</i>	IPv4 or IPv6 protocol numbe
OXM_OF_IPV4_SRC	<i>Required</i>	IPv4 source address. Can use
OXM_OF_IPV4_DST	<i>Required</i>	IPv4 destination address. Can
OXM_OF_IPV6_SRC	<i>Required</i>	IPv6 source address. Can use
OXM_OF_IPV6_DST	<i>Required</i>	IPv6 destination address. Can
OXM_OF_TCP_SRC	<i>Required</i>	TCP source port
OXM_OF_TCP_DST	<i>Required</i>	TCP destination port
OXM_OF_UDP_SRC	<i>Required</i>	UDP source port
OXM_OF_UDP_DST	<i>Required</i>	UDP destination port

Table 11: Required match f

ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2015, p. 177. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/05/2021)

Matching on Multiple Fields

OpenFlow can match on **multiple fields**.

- A packet is matched by a rule if **all** match condition is satisfied
- The matching rule with **the highest priority** will be executed

Example:

<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
sip=192.168.1.0/24 dip=192.168.2.0/24	100	{output(A)}
dip=192.168.2.0/24	200	{output(B)}

Consider the following packets:

<i>header</i>	<i>output port</i>
sip=192.168.1.2 dip=192.168.2.2	
sip=192.168.3.2 dip=192.168.2.2	

Matching on Multiple Fields

OpenFlow can match on **multiple fields**.

- A packet is matched by a rule if **all** match condition is satisfied
- The matching rule with **the highest priority** will be executed

Example:

<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
sip=192.168.1.0/24 dip=192.168.2.0/24	100	{output(A)}
dip=192.168.2.0/24	200	{output(B)}

Consider the following packets:

<i>header</i>	<i>output port</i>
sip=192.168.1.2 dip=192.168.2.2	B
sip=192.168.3.2 dip=192.168.2.2	

Matching on Multiple Fields

OpenFlow can match on **multiple fields**.

- A packet is matched by a rule if **all** match condition is satisfied
- The matching rule with **the highest priority** will be executed

Example:

<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
sip=192.168.1.0/24 dip=192.168.2.0/24	100	{output(A)}
dip=192.168.2.0/24	200	{output(B)}

Consider the following packets:

<i>header</i>	<i>output port</i>
sip=192.168.1.2 dip=192.168.2.2	B
sip=192.168.3.2 dip=192.168.2.2	B

Matching on Multiple Fields

OpenFlow can match on **multiple fields**.

- A packet is matched by a rule if **all** match condition is satisfied
- The matching rule with **the highest priority** will be executed

Example:

<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
sip=192.168.1.0/24 dip=192.168.2.0/24	100	{output(A)}
← dominated by the next rule		
dip=192.168.2.0/24	200	{output(B)}

Consider the following packets:

<i>header</i>	<i>output port</i>
sip=192.168.1.2 dip=192.168.2.2	B
sip=192.168.3.2 dip=192.168.2.2	B

Matching on Multiple Fields

OpenFlow can match on **multiple fields**.

- A packet is matched by a rule if **all** match condition is satisfied
- The matching rule with **the highest priority** will be executed

Example:

<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
sip=192.168.1.0/24 dip=192.168.2.0/24	300	{output(A)}
dip=192.168.2.0/24	200	{output(B)}

Consider the following packets:

<i>header</i>	<i>output port</i>
sip=192.168.1.2 dip=192.168.2.2	
sip=192.168.3.2 dip=192.168.2.2	

Matching on Multiple Fields

OpenFlow can match on **multiple fields**.

- A packet is matched by a rule if **all** match condition is satisfied
- The matching rule with **the highest priority** will be executed

Example:

<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
sip=192.168.1.0/24 dip=192.168.2.0/24	300	{output(A)}
dip=192.168.2.0/24	200	{output(B)}

Consider the following packets:

<i>header</i>	<i>output port</i>
sip=192.168.1.2 dip=192.168.2.2	A
sip=192.168.3.2 dip=192.168.2.2	

Matching on Multiple Fields

OpenFlow can match on **multiple fields**.

- A packet is matched by a rule if **all** match condition is satisfied
- The matching rule with **the highest priority** will be executed

Example:

<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
sip=192.168.1.0/24 dip=192.168.2.0/24	300	{output(A)}
dip=192.168.2.0/24	200	{output(B)}

Consider the following packets:

<i>header</i>	<i>output port</i>
sip=192.168.1.2 dip=192.168.2.2	A
sip=192.168.3.2 dip=192.168.2.2	B

Actions

OpenFlow 1.3 supports the following instructions

- **Meter** instruction (计量指令, to be explained later)
- **Apply-Actions** instruction (执行行为指令): execute actions **immediately**
- **Clear/Write-Actions** instructions (清除、写入行为指令): update the action set but **do not execute the actions**
- **Write-Metadata** instruction (元数据更新指令): update the metadata register
- **Goto-Table** instruction (表跳转指令): set the next table

Actions for Apply-Actions Instruction

The Apply-Actions instruction **immediately** executes the specified actions **in the specified order**

The following actions can be specified

- **output port** action (转发行为): set the egress port to port (to be explained later)
- **group group_id** action (组表转发指令): process the packet by a specific group (to be explained later)
- **drop** action (丢弃指令): discard the packet
- **set-queue queue_id** action (队列设置指令): set the egress queue when sending to a port
- **push/pop-tag ethertype** action (标签弹出行为): modify the tag stack (for VLAN, MPLS, etc.)
- **set-field field value** action (修改报头域行为): set the value of a specified field
- **change-ttl ttl_value** action (修改 TTL 值): modify the TTL of the packet

Actions for Apply-Actions Instruction

The Apply-Actions instruction **immediately** executes the specified actions **in the specified order**

The following actions can be specified

- **output port action** (转发行为): set the egress port to port (to be explained later)
- **group group_id action** (组表转发指令): process the packet by a specific group (to be explained later)
- **drop action** (丢弃指令): discard the packet
- **set-queue queue_id action** (队列设置指令): set the egress queue when sending to a port
- **push/pop-tag ethertype action** (标签弹出行为): modify the tag stack (for VLAN, MPLS, etc.)
- **set-field field value action** (修改报头域行为): set the value of a specified field
- **change-ttl ttl_value action** (修改 TTL 值): modify the TTL of the packet

Actions for Clear/Write-Actions Instruction

The Clear/Write-Actions instruction modifies the action set that is executed **after the processing pipeline**. Each action type **only keeps the last action** and is executed **in a fixed order**

The following actions can be specified

- copy TTL to inner protocol header
- pop tag
- push MPLS label
- push PBB
- push VLAN
- copy TTL to outer protocol header
- decrement TTL
- set field
- set QoS
- specify the group id (if a group action is set)
- specify the egress port (if a group action is not set)

Ports

OpenFlow has defined three types of ports (端口, 转发目的地):

- Physical port (物理端口): 1-to-1 mapping with a switches' interfaces

Ports

OpenFlow has defined three types of ports (端口, 转发目的地):

- Physical port (物理端口): 1-to-1 mapping with a switches' interfaces
- Logical port (逻辑端口): n-to-m mapping with a switches' interfaces, identified by tunnel-id


Ports

OpenFlow has defined three types of ports (端口, 转发目的地):

- Physical port (物理端口): 1-to-1 mapping with a switches' interfaces
- Logical port (逻辑端口): n-to-m mapping with a switches' interfaces, identified by tunnel-id
- Reserved port (保留端口)
 - ALL: all the ports (only as an output port)
 - CONTROLLER: the control channel to the OpenFlow controller (can be a physical port or a logical port)
 - TABLE: the first flow table of the pipeline (used in *packet-out* message)
 - IN_PORT: the ingress port of a packet (only as an output port)
 - ANY: specifying no constraints for certain messages (cannot be used as ingress port or output port)
 - LOCAL: the local interface of the switch OS
 - NORMAL & FLOOD: send packets to non-openflow pipeline for routing or flooding


Ports

OpenFlow has defined three types of ports (端口, 转发目的地):

- Physical port (物理端口): 1-to-1 mapping with a switches' interfaces
 - Logical port (逻辑端口): n-to-m mapping with a switches' interfaces, identified by tunnel-id
 - Reserved port (保留端口)
 - **ALL: all the ports (only as an output port)**
 - CONTROLLER: the control channel to the OpenFlow controller (can be a physical port or a logical port)
 - TABLE: the first flow table of the pipeline (used in *packet-out* message)
 - IN_PORT: the ingress port of a packet (only as an output port)
 - ANY: specifying no constraints for certain messages (cannot be used as ingress port or output port)
 - LOCAL: the local interface of the switch OS
 - NORMAL & FLOOD: send packets to non-openflow pipeline for routing or flooding
- Used for flooding 

Ports

OpenFlow has defined three types of ports (端口, 转发目的地):

- Physical port (物理端口): 1-to-1 mapping with a switches' interfaces
 - Logical port (逻辑端口): n-to-m mapping with a switches' interfaces, identified by tunnel-id
 - Reserved port (保留端口)
 - ALL: all the ports (only as an output port)
 - **CONTROLLER: the control channel to the OpenFlow controller (can be a physical port or a logical port)**
 - TABLE: the first flow table of the pipeline (used in *packet-out* message)
 - IN_PORT: the ingress port of a packet (only as an output port)
 - ANY: specifying no constraints for certain messages (cannot be used as ingress port or output port)
 - LOCAL: the local interface of the switch OS
 - NORMAL & FLOOD: send packets to non-openflow pipeline for routing or flooding
- Used for sending packets to controller 

Ports

OpenFlow has defined three types of ports (端口, 转发目的地):

- Physical port (物理端口): 1-to-1 mapping with a switches' interfaces
- Logical port (逻辑端口): n-to-m mapping with a switches' interfaces, identified by tunnel-id
- Reserved port (保留端口)
 - ALL: all the ports (only as an output port)
 - CONTROLLER: the control channel to the OpenFlow controller (can be a physical port or a logical port)
 - TABLE: the first flow table of the pipeline (used in *packet-out* message)
 - IN_PORT: the ingress port of a packet (only as an output port)
 - ANY: specifying no constraints for certain messages (cannot be used as ingress port or output port)
 - LOCAL: the local interface of the switch OS
 - NORMAL & FLOOD: send packets to non-openflow pipeline for routing or flooding

Used for traffic reflection →

Group Table

Group table (组表) is used for special-purpose routing such as ECMP, back-up routing. There are 4 group types:

- **indirect**: execute the **single** action bucket
- **all**: execute **all** buckets (used for multicast or broadcast)
- **select**: execute **one** bucket (used for load balancing or ECMP)
- **fast failover**: execute **the first** live bucket (used for fast rerouting)

Group Identifier	Group Type	Counters	Action Buckets
------------------	------------	----------	----------------

Table 2: Main components of a group entry in the group table.

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports A, B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
		-

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports A, B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
indirect	(invalid)	-

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports A, B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
indirect	(invalid)	-
all	A, B, & C	3

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports A, B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
indirect	(invalid)	-
all	A, B, & C	3
select	A, B, or C	1

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports A, B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
indirect	(invalid)	-
all	A, B, & C	3
select	A, B, or C	1
fast failover	A	1

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
indirect	(invalid)	-

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
indirect	(invalid)	-
all	B & C	2

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
indirect	(invalid)	-
all	B & C	2
select	B or C	1

Group Table Example

Assume we have the following group entry
(组表项):

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(A)} {output(B)} {output(C)}

When ports B and C are alive:

<i>group type</i>	<i>output port</i>	<i>#packets</i>
indirect	(invalid)	-
all	B & C	2
select	B or C	1
fast failover	B	1

Meter Table

Meter table (计量表) is used to achieve rate limiting (限流) and QoS control

A single **meter table entry** (计量表项) (TOP) has multiple **meter bands** (计量带) (BOTTOM).

Meter Identifier	Meter Bands	Counters
------------------	-------------	----------

Table 3: Main components of a meter entry in the meter table.

Band selection:

- The band **with the largest rate that is lower than the current measured rate** will be applied.
- Two types of meter bands:
 - **drop**: discard the packet
 - **dscp remark**: mark the IP DSCP field (used for DiffServ)

Band Type	Rate	Burst	Counters	Type specific arguments
-----------	------	-------	----------	-------------------------

Table 4: Main components of a meter band in a meter entry.

Meter Table Example

Assume a meter table entry has 3 bands:

<i>band id</i>	<i>meter type</i>	<i>rate</i>
1	dscp-remark 1	100
2	dscp-remark 3	200
3	drop	300

Assume the measured rate is

- 99: no band will be applied
- 150: band 1 will be applied

Meter Table Example

Assume a meter table entry has 3 bands:

<i>band id</i>	<i>meter type</i>	<i>rate</i>
1	dscp-remark 1	100
2	dscp-remark 3	200
3	drop	300

Assume the measured rate is

- 99: no band will be applied
- 150: band 1 will be applied
- 250: band 2 will be applied

Meter Table Example

Assume a meter table entry has 3 bands:

<i>band id</i>	<i>meter type</i>	<i>rate</i>
1	dscp-remark 1	100
2	dscp-remark 3	200
3	drop	300

Assume the measured rate is

- 99: no band will be applied
- 150: band 1 will be applied
- 250: band 2 will be applied
- 350: band 3 will be applied

OpenFlow Messages

OpenFlow switches exchange information with the controller with the OpenFlow messages.

Common controller-to-switch messages include:

- **Packet-out:** Encapsulate a packet in the message and send to a switch
- **Flow-mod:** Insert/Update/Delete a flow rule in a flow table
- **Barrier:** Setting barriers (屏障) for messages (to be explained later)

Common switch-to-controller messages include:

- **Packet-in:** Encapsulate a packet in the message and send to the controller
- **Flow-removed:** Notify the controller that a flow rule is removed (because of timeout)
- **Port-status:** Notify the controller that the status of a port has changed (up to down or down to up)

Barrier Mechanism

Without barrier messages, a switch **may process the messages in arbitrary order** (ONF 2015a).

Example: Consider the initial flow table and the following messages:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

- FlowMod(Delete(192.168.1.0/24))
- FlowMod(Insert(192.168.1.0/24, 2, output(C)))

Result in the right order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Barrier Mechanism

Without barrier messages, a switch **may process the messages in arbitrary order** (ONF 2015a).

Example: Consider the initial flow table and the following messages:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

- FlowMod(Delete(192.168.1.0/24))
- FlowMod(Insert(192.168.1.0/24, 2, output(C)))

Result in the right order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Barrier Mechanism

Without barrier messages, a switch **may process the messages in arbitrary order** (ONF 2015a).

Example: Consider the initial flow table and the following messages:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

- FlowMod(Delete(192.168.1.0/24))
- FlowMod(Insert(192.168.1.0/24, 2, output(C)))

Result in the right order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(C)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Barrier Mechanism

Without barrier messages, a switch **may process the messages in arbitrary order** (ONF 2015a).

Example: Consider the initial flow table and the following messages:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

- FlowMod(Delete(192.168.1.0/24))
- FlowMod(Insert(192.168.1.0/24, 2, output(C)))

Result in the right order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(C)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Result in the wrong order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Barrier Mechanism

Without barrier messages, a switch **may process the messages in arbitrary order** (ONF 2015a).

Example: Consider the initial flow table and the following messages:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

- FlowMod(Delete(192.168.1.0/24))
- FlowMod(Insert(192.168.1.0/24, 2, output(C)))

Result in the right order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(C)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Result in the wrong order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(C)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Barrier Mechanism

Without barrier messages, a switch **may process the messages in arbitrary order** (ONF 2015a).

Example: Consider the initial flow table and the following messages:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

- FlowMod(Delete(192.168.1.0/24))
- FlowMod(Insert(192.168.1.0/24, 2, output(C)))

Result in the right order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(C)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Result in the wrong order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Barrier Mechanism

Without barrier messages, a switch **may process the messages in arbitrary order** (ONF 2015a).

Example: Consider the initial flow table and the following messages:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(B)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

- FlowMod(Delete(192.168.1.0/24))
- **Barrier** ← **enforce the message order**
- FlowMod(Insert(192.168.1.0/24, 2, output(C)))

Result in the right order:

<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.1.0/24	2	output(C)
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Result in the wrong order:

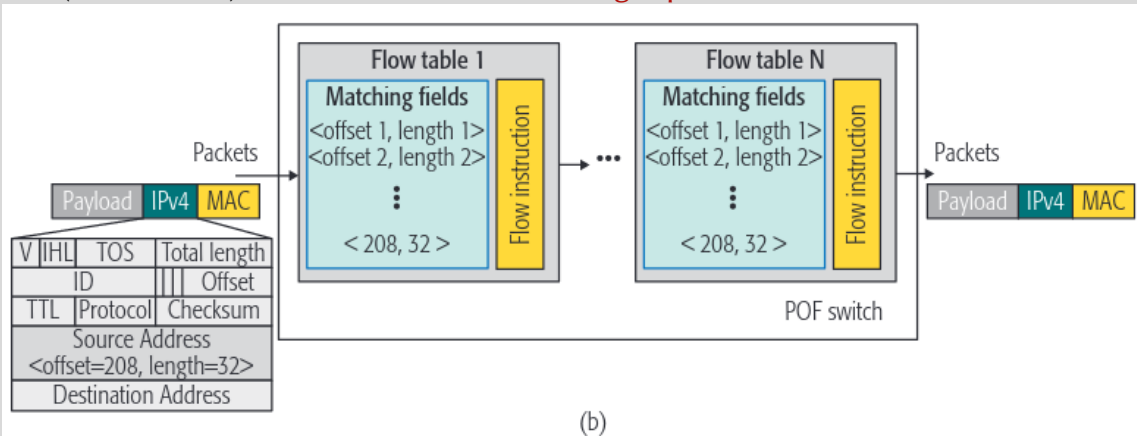
<i>match</i>	<i>priority</i>	<i>action</i>
dip=192.168.0.0/16	1	output(A)
dip=0.0.0.0/0	0	drop

Alternative SDN Data Planes

Protocol Oblivious Forwarding
Reconfigurable Match-Action Table

Protocol Oblivious Forwarding

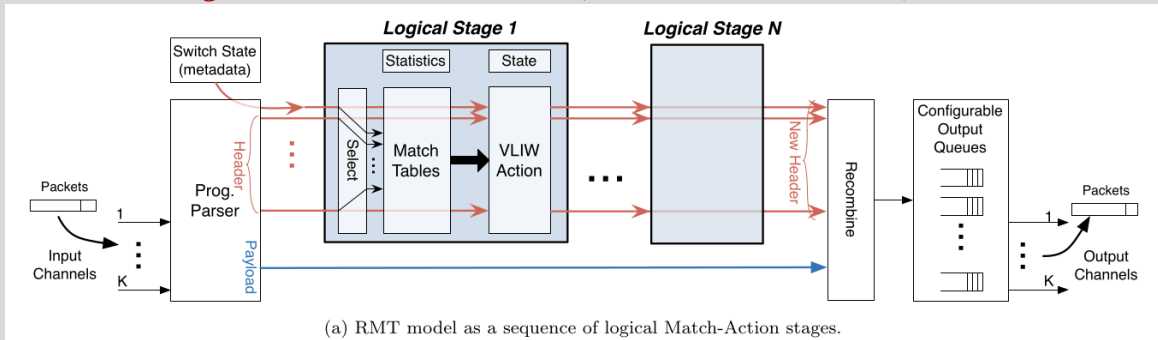
PoF (协议无关转发) uses customizable offset-length pairs instead of fixed fields



Shengru Li et al. "Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability". In: *IEEE Network* 31.2 (Mar. 2017), pp. 58–66

Reconfigurable Match-Action Tables

P4 uses **reconfigurable match-action tables** (RMT, 可重配置匹配行为表)



Pat Bosshart et al. "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 99–110. URL: <http://doi.acm.org/10.1145/2486001.2486011>

Protocol Parsing in P4

Protocols are specified as **parse graph** (解析图) (BOTTOM) and are **used to program the hardware parser components** (RIGHT)

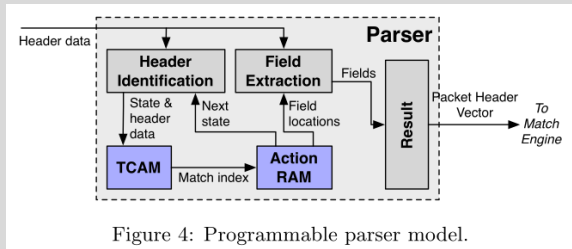
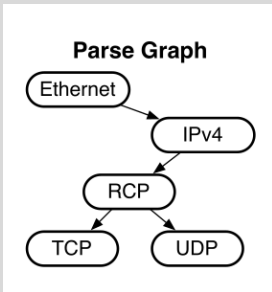
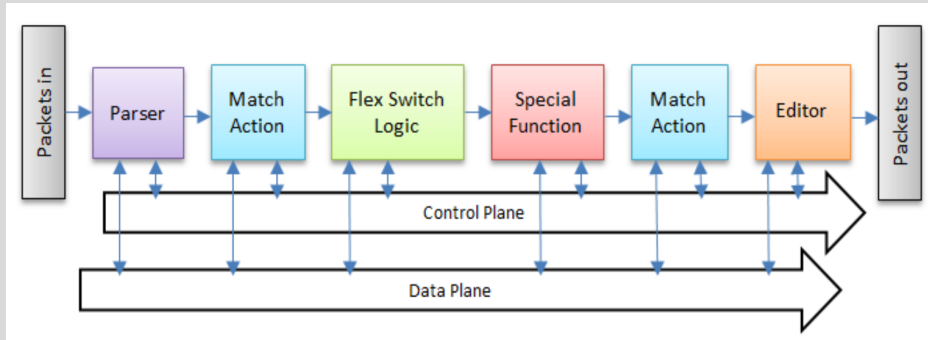


Figure 4: Programmable parser model.

Pat Bosshart et al. "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 99–110. URL: <http://doi.acm.org/10.1145/2486001.2486011>

NPL

NPL allows **customizable protocol parsing and actions**. It is similar to RMT but with more advanced functionalities.



The End

Summary

In this lecture, we cover the following topics:

- What is data plane in networking and in SDN and how it is realized.
- OpenFlow switch specification: table layout, flow tables, group table, meter table, communication messages
- Other SDN data planes

Summary

In this lecture, we cover the following topics:

- What is data plane in networking and in SDN and how it is realized.
- OpenFlow switch specification: table layout, flow tables, group table, meter table, communication messages
- Other SDN data planes

You should

- Understand the match-action paradigm

Summary

In this lecture, we cover the following topics:

- What is data plane in networking and in SDN and how it is realized.
- OpenFlow switch specification: table layout, flow tables, group table, meter table, communication messages
- Other SDN data planes

You should

- Understand the match-action paradigm
- Understand (roughly) the packet processing in flow table, group table and meter table

Summary

In this lecture, we cover the following topics:

- What is data plane in networking and in SDN and how it is realized.
- OpenFlow switch specification: table layout, flow tables, group table, meter table, communication messages
- Other SDN data planes

You should

- Understand the match-action paradigm
- Understand (roughly) the packet processing in flow table, group table and meter table
- Know common match fields and actions (especially the output action and ports)

Setup: Lab 1

Please

- Bring you laptop next week
- Create a virtual machine image with Ubuntu 18.04+ **before class**
- Set the apt mirror to a Chinese site

Thanks!

kaigao@scu.edu.cn

Quiz

Question 1

假设一个 OpenFlow 交换机的流表如下，回答下面的问题

<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
sip=192.168.1.0/24	300	{output(A)}
dip=192.168.2.0/24	200	{output(B)}
sip=192.168.1.0/24, dip=192.168.2.0/24	100	{output(C)}

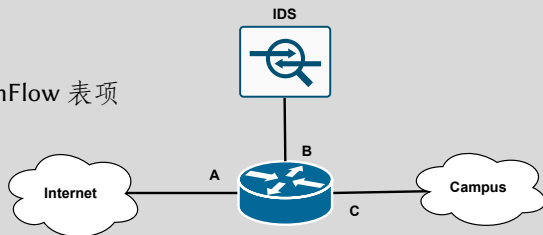
1. 数据包 $p_1 = \{\text{sip} : 192.168.1.2, \text{dip} : 192.168.2.2\}$ 会往哪个端口转发？为什么？
2. 请写出一个向 B 端口转发的数据包可能的源 IP 地址和目的 IP 地址

Quiz

Question 2

假设一个网络要实现如下的功能，补充下面的 OpenFlow 表项

- 互联网的流量进入校园网时复制一份给 IDS
- 校园网的流量进入互联网时复制一份给 IDS



<i>match</i>	<i>priority</i>	<i>Apply-Actions</i>
ingress=A	300	{group(1)}
?	?	?

<i>group ID</i>	<i>group type</i>	<i>action buckets</i>
1	?	{output(B)} {output(C)}
?	?	?

References I

- [1] Madhulika Bhandure and Gaurang Deshmukh. “Comparative Analysis of Mpls and Non -Mpls Network”. In: 3.4 (2013), p. 6.
- [2] Pat Bosshart et al. “Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN”. In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM ’13. New York, NY, USA: ACM, 2013, pp. 99–110. URL: <http://doi.acm.org/10.1145/2486001.2486011>.
- [3] Broadcom. *NPL - Network Programming Language Specification v1.3*. 2019.
- [4] Miguel Elias M. Campista et al. “Challenges and Research Directions for the Future Internet networking”. In: *IEEE Commun. Surv. Tutorials* 16.2 (Sum. 2014), pp. 1050–1079. URL: <http://ieeexplore.ieee.org/document/6644748/> (visited on 09/04/2021).
- [5] Nick Feamster, Jennifer Rexford, and Ellen Zegura. “The Road to SDN: An Intellectual History of Programmable Networks”. In: *Queue* 11.12 (Dec. 2013), 20:20–20:40. URL: <http://doi.acm.org/10.1145/2559899.2560327>.
- [6] Joel M. Halpern et al. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. RFC 5810. Mar. 2010. URL: <https://rfc-editor.org/rfc/rfc5810.txt>.
- [7] D. Kreutz et al. “Software-Defined Networking: A Comprehensive Survey”. In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76.
- [8] Marianne Lepp et al. *Terminology for Benchmarking BGP Device Convergence in the Control Plane*. RFC 4098. June 2005. URL: <https://rfc-editor.org/rfc/rfc4098.txt>.
- [9] Shengru Li et al. “Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability”. In: *IEEE Network* 31.2 (Mar. 2017), pp. 58–66.
- [10] ONF. *OpenFlow Switch Specification (Version 1.0.0)*. Open Networking Foundation, 2009.

References II

- [11] ONF. *OpenFlow Switch Specification (Version 1.1.0)*. Open Networking Foundation, 2011.
- [12] ONF. *OpenFlow Switch Specification (Version 1.3.5)*. Open Networking Foundation, Mar. 26, 2015, p. 177. URL: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> (visited on 09/05/2021).
- [13] ONF. *The Benefits of Multiple Flow Tables and TTPs*. Open Networking Foundation, Feb. 2, 2015. URL: https://opennetworking.org/wp-content/uploads/2014/10/TR_Multiple_Flow_Tables_and_TTPs.pdf (visited on 09/05/2021).
- [14] K. Pagiamtzis and A. Sheikholeslami. “Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey”. In: *IEEE J. Solid-State Circuits* 41.3 (Mar. 2006), pp. 712–727. URL: <http://ieeexplore.ieee.org/document/1599540/> (visited on 09/05/2021).
- [15] Thamaraiselvam Santhanam. *CAM (Content Addressable Memory) VS TCAM (Ternary Content Addressable Memory)*. Mar. 25, 2011. URL: <https://community.cisco.com/t5/networking-documents/cam-content-addressable-memory-vs-tcam-ternary-content/ta-p/3107938> (visited on 09/05/2021).
- [16] Arun Viswanathan, Eric C. Rosen, and Ross Callon. *Multiprotocol Label Switching Architecture*. RFC 3031. Jan. 2001. URL: <https://rfc-editor.org/rfc/rfc3031.txt>.