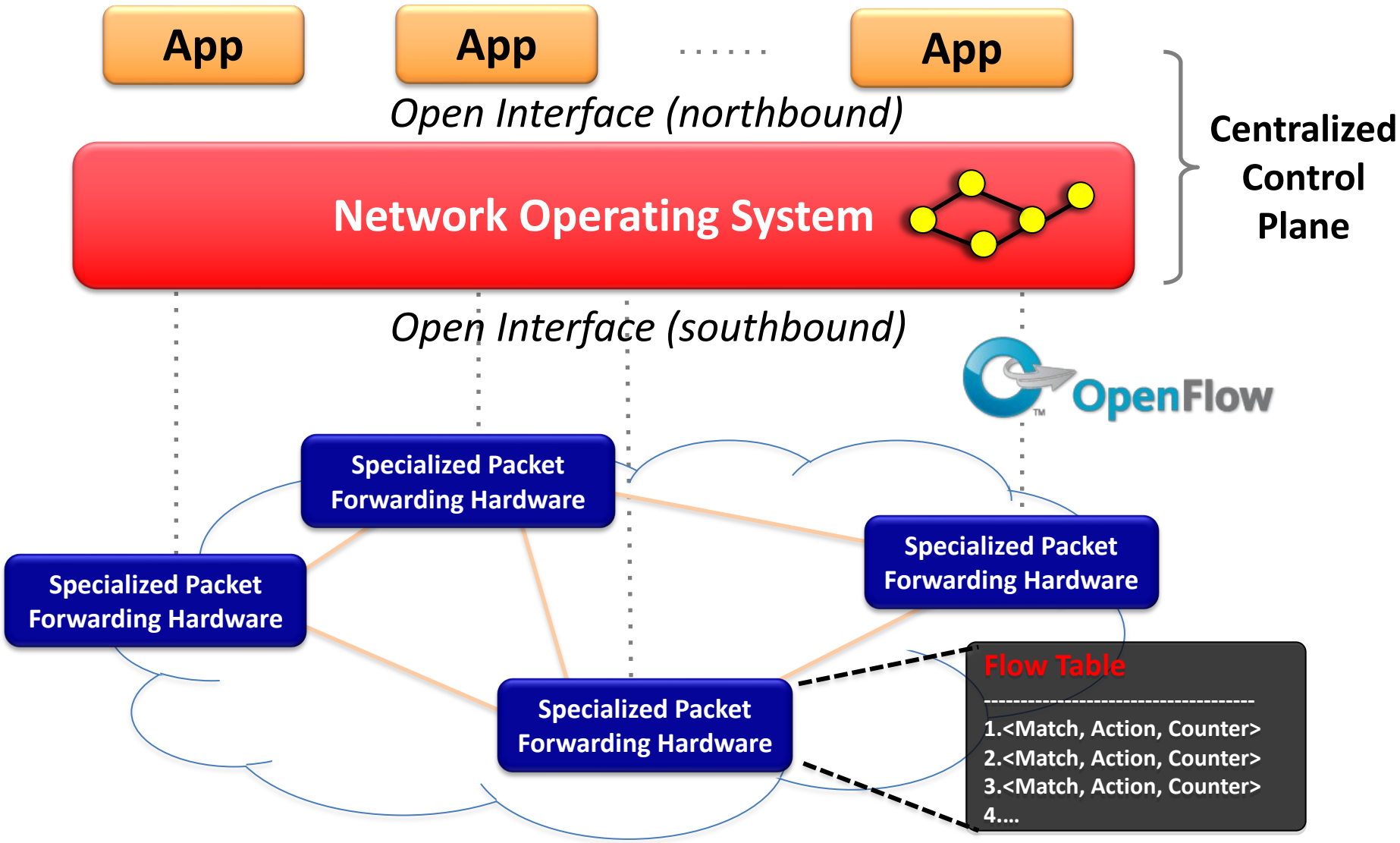
OpenFlow

Peng Zhang
School of Computer Science and Technology
Xi'an Jiaotong University

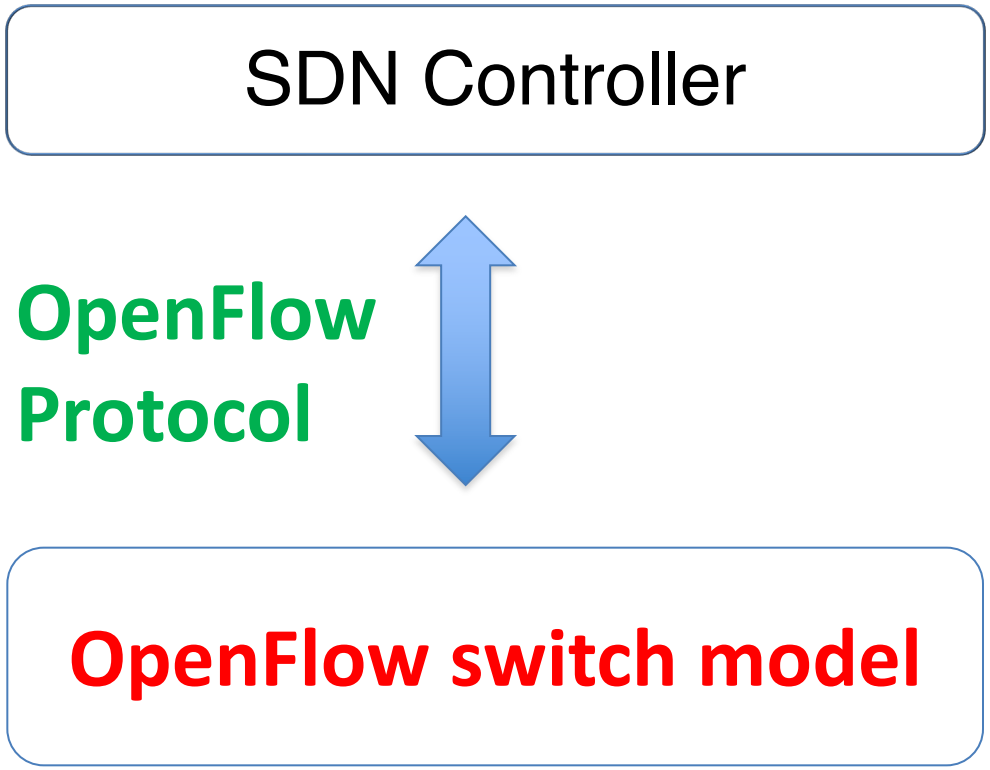
Spring 2024

Software Defined Network (SDN)



OpenFlow = switch model + protocol

- OpenFlow defines **how switch forwards packets** and **how switches interact with the controller**



The OpenFlow Specification

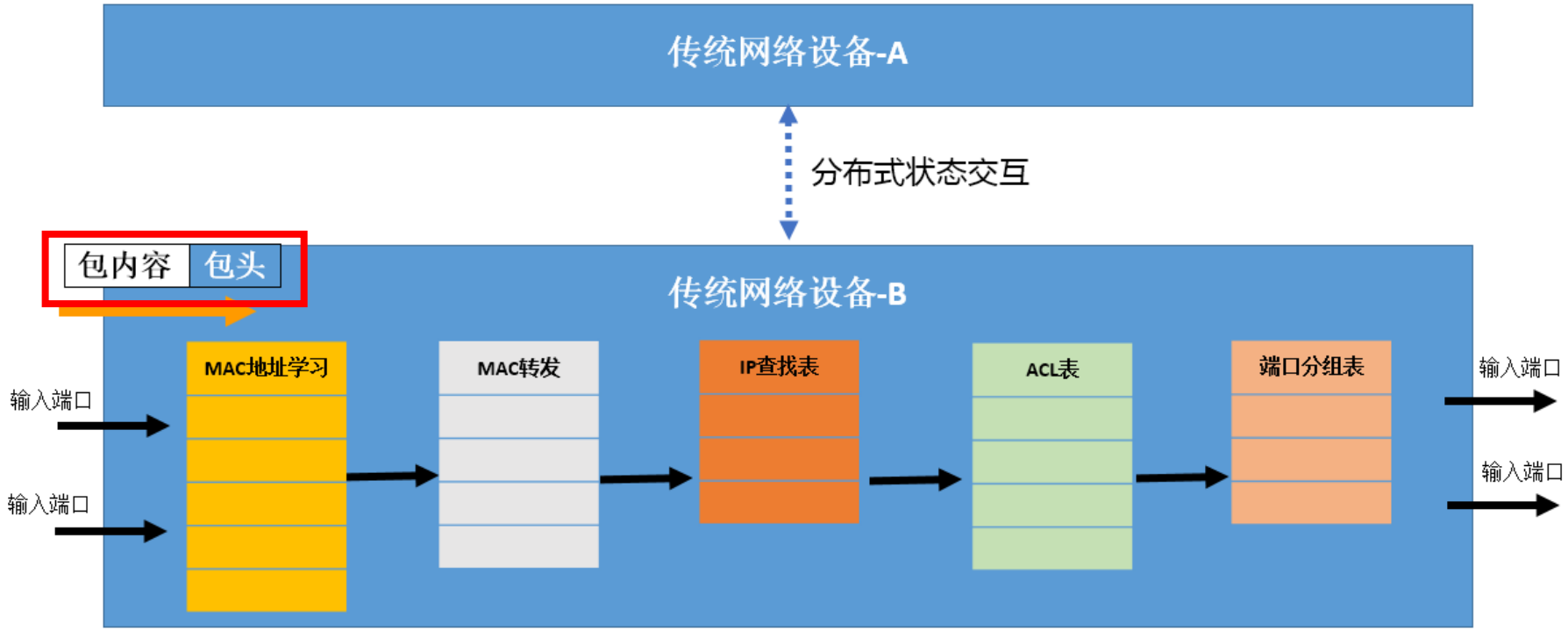
OpenFlow Switch Specification	OpenFlow Switch Specification
Version 1.0.0 (Wire Protocol 0x01)	Version 1.3.4 (Protocol version 0x04)
December 31, 2009	March 27, 2014
(42 Pages)	(171 Pages)

https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf
https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.4.pdf

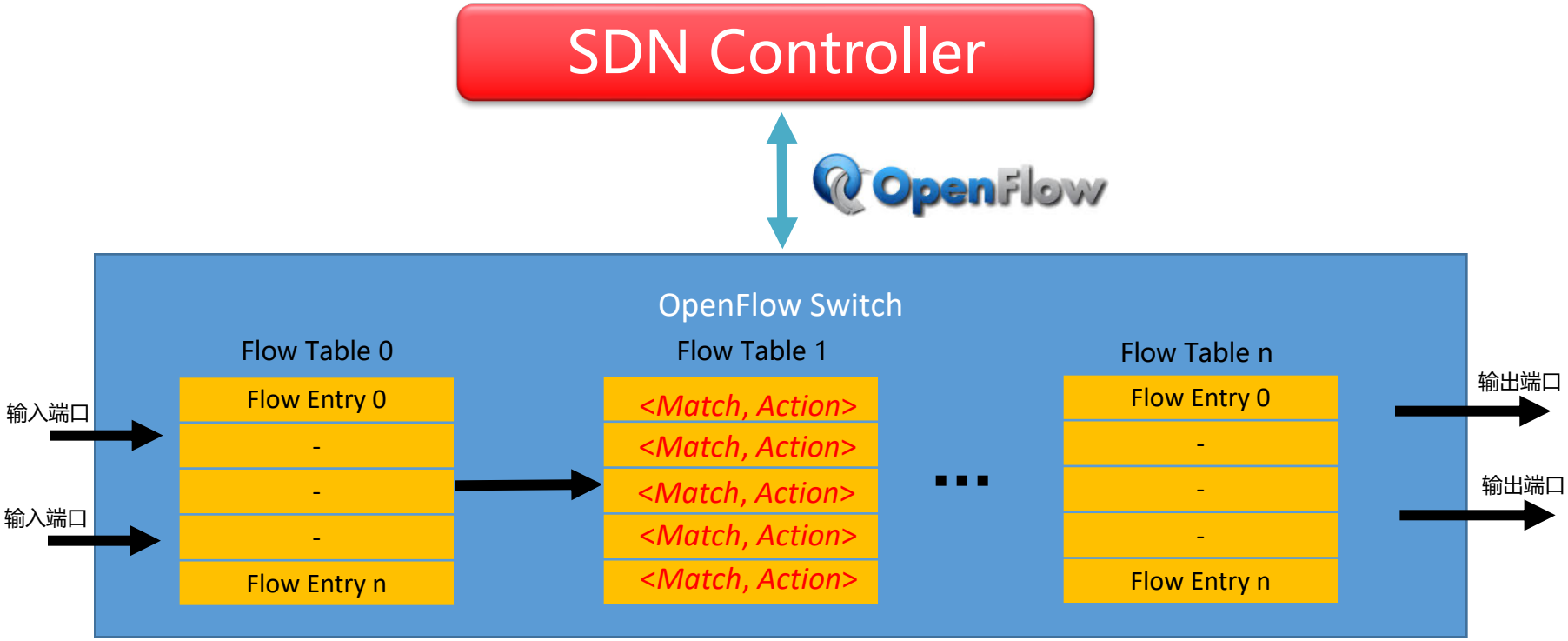
OpenFlow Data Plane

(v1.3.4)

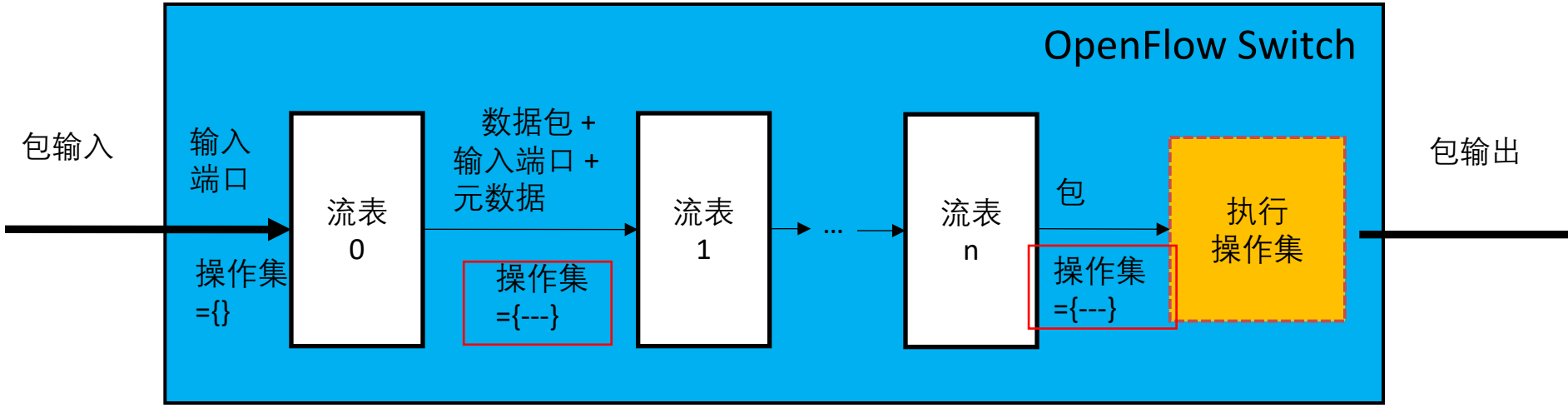
Traditional Switch Model



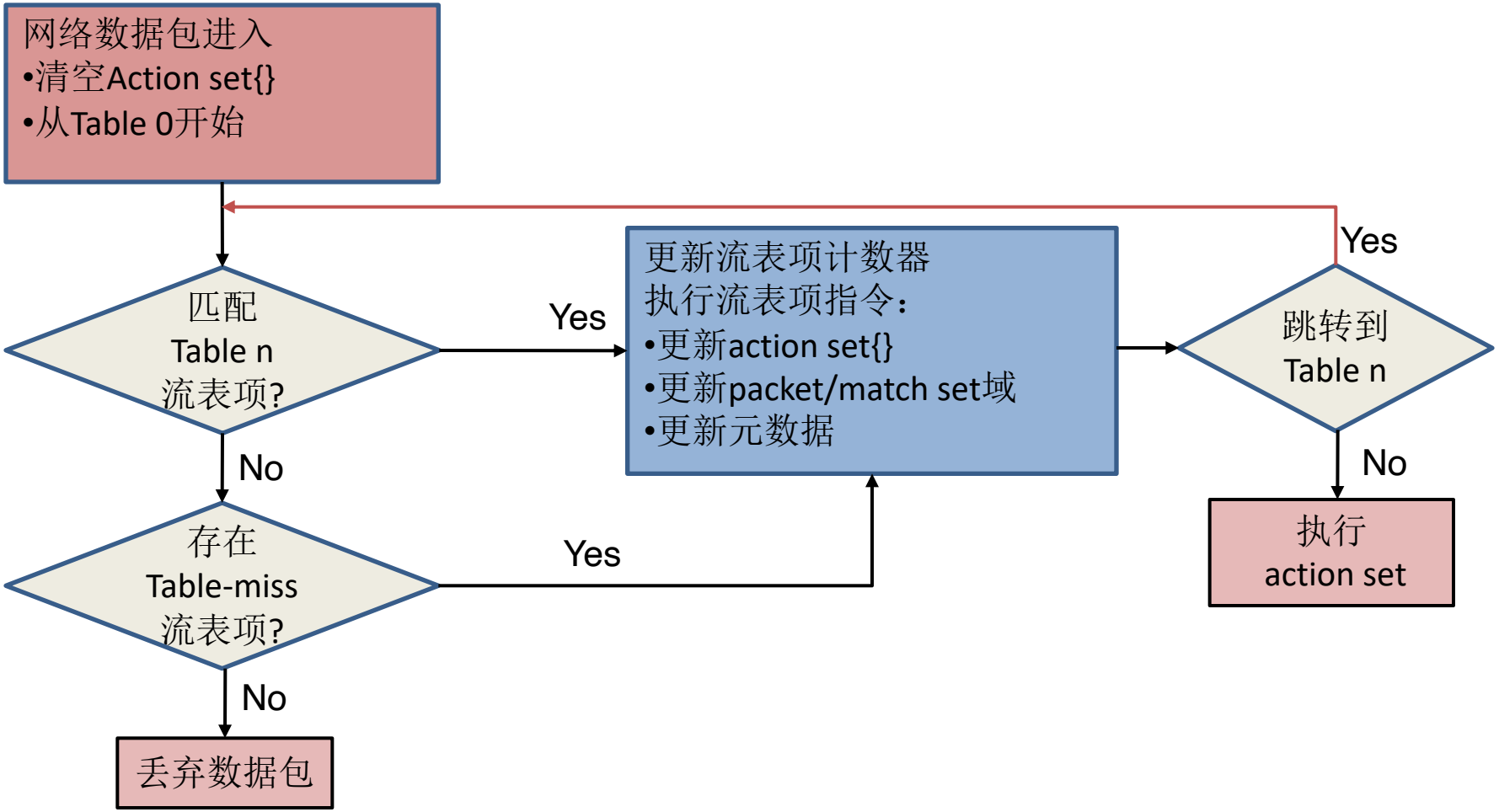
OpenFlow Switch Model



Flow Table Pipeline



Flow Table Processing



OpenFlow Entry: Match Fields

匹配域 (Match Fields)	**优先级 (Priority)**	计数器 (Counters)	指令 (Instructions)	失效时间 (Timeouts)	Cookie	Flags

Ingress Port	Meta data	Ethernet			VLAN		MPLS		IP				TCP/UDP/ICMP	
		src	dst	type	id	prio.	lab.	t.c.	src	dst	proto.	ToS	src port /icmp type	dp port /icmp code

struct ofp_match

```
uint16_t type;            /* One of OFPMT_* */
uint16_t length;          /* Length of ofp_match (excluding
padding) */
uint8_t oxm_fields[0];  /* 0 or more OXM match fields */
uint8_t pad[4];           /* Zero bytes - see above for sizing
*/
```

```
enum ofp_match_type {
OFPMT_STANDARD = 0,   /*
Deprecated*/
OFPMT_OXM = 1,          /* OpenFlow
Extensible Match */};
```

OpenFlow Entry: Match Fields

- OpenFlow 1.3.4: 40 fields

```
/* OXM Flow match field types for OpenFlow basic class. */
enum oxm_ofb_match_fields {
OFPXMT_OFB_IN_PORT = 0, /* Switch input port. */
OFPXMT_OFB_IN_PHY_PORT = 1, /* Switch physical input port. */
OFPXMT_OFB_METADATA = 2, /* Metadata passed between tables. */
OFPXMT_OFB_ETH_DST = 3, /* Ethernet destination address. */
OFPXMT_OFB_ETH_SRC = 4, /* Ethernet source address. */
OFPXMT_OFB_ETH_TYPE = 5, /* Ethernet frame type. */
OFPXMT_OFB_VLAN_VID = 6, /* VLAN id. */
OFPXMT_OFB_VLAN_PCP = 7, /* VLAN priority. */
OFPXMT_OFB_IP_DSCP = 8, /* IP DSCP (6 bits in ToS field). */
OFPXMT_OFB_IP_ECN = 9, /* IP ECN (2 bits in ToS field). */
OFPXMT_OFB_IP_PROTO = 10, /* IP protocol. */
OFPXMT_OFB_IPV4_SRC = 11, /* IPv4 source address. */
OFPXMT_OFB_IPV4_DST = 12, /* IPv4 destination address. */
OFPXMT_OFB_TCP_SRC = 13, /* TCP source port. */
OFPXMT_OFB_TCP_DST = 14, /* TCP destination port. */
OFPXMT_OFB_UDP_SRC = 15, /* UDP source port. */
OFPXMT_OFB_UDP_DST = 16, /* UDP destination port. */
OFPXMT_OFB_SCTP_SRC = 17, /* SCTP source port. */
OFPXMT_OFB_SCTP_DST = 18, /* SCTP destination port. */
OFPXMT_OFB_ICMPV4_TYPE = 19, /* ICMP type. */
OFPXMT_OFB_ICMPV4_CODE = 20, /* ICMP code. */
OFPXMT_OFB_ARP_OP = 21, /* ARP opcode. */
OFPXMT_OFB_ARP_SPA = 22, /* ARP source IPv4 address. */
OFPXMT_OFB_ARP_TPA = 23, /* ARP target IPv4 address. */
OFPXMT_OFB_ARP_SHA = 24, /* ARP source hardware address. */
OFPXMT_OFB_ARP_THA = 25, /* ARP target hardware address. */
OFPXMT_OFB_IPV6_SRC = 26, /* IPv6 source address. */
OFPXMT_OFB_IPV6_DST = 27, /* IPv6 destination address. */
OFPXMT_OFB_IPV6_FLABEL = 28, /* IPv6 Flow Label */
OFPXMT_OFB_ICMPV6_TYPE = 29, /* ICMPv6 type. */
OFPXMT_OFB_ICMPV6_CODE = 30, /* ICMPv6 code. */
OFPXMT_OFB_IPV6_ND_TARGET = 31, /* Target address for ND. */
OFPXMT_OFB_IPV6_ND_SLL = 32, /* Source link-layer for ND. */
OFPXMT_OFB_IPV6_ND_TLL = 33, /* Target link-layer for ND. */
OFPXMT_OFB_MPLS_LABEL = 34, /* MPLS label. */
OFPXMT_OFB_MPLS_TC = 35, /* MPLS TC. */
OFPXMT_OFP_MPLS_BOS = 36, /* MPLS BoS bit. */
OFPXMT_OFB_PBB_ISID = 37, /* PBB I-SID. */
OFPXMT_OFB_TUNNEL_ID = 38, /* Logical Port Metadata. */
OFPXMT_OFB_IPV6_EXTHDR = 39, /* IPv6 Extension Header pseudo-
field */
};
```

Required Matching Fields

- OpenFlow 1.3.4: required fields

Layer 1

Field		Description
OXM_OF_IN_PORT	*Required*	Ingress port. This may be a physical or switch-defined logical port.
OXM_OF_ETH_DST	*Required*	Ethernet destination address. Can use arbitrary bitmask
OXM_OF_ETH_SRC	*Required*	Ethernet source address. Can use arbitrary bitmask
OXM_OF_ETH_TYPE	*Required*	Ethernet type of the OpenFlow packet payload, after VLAN tags.
OXM_OF_IP_PROTO	*Required*	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	*Required*	IPv4 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV4_DST	*Required*	IPv4 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_SRC	*Required*	IPv6 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_DST	*Required*	IPv6 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_TCP_SRC	*Required*	TCP source port
OXM_OF_TCP_DST	*Required*	TCP destination port
OXM_OF_UDP_SRC	*Required*	UDP source port
OXM_OF_UDP_DST	*Required*	UDP destination port

Required Matching Fields

- OpenFlow 1.3.4: required fields

Layer 2

Field		Description
OXM_OF_IN_PORT	*Required*	Ingress port. This may be a physical or switch-defined logical port.
OXM_OF_ETH_DST	*Required*	Ethernet destination address. Can use arbitrary bitmask
OXM_OF_ETH_SRC	*Required*	Ethernet source address. Can use arbitrary bitmask
OXM_OF_ETH_TYPE	*Required*	Ethernet type of the OpenFlow packet payload, after VLAN tags.
OXM_OF_IP_PROTO	*Required*	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	*Required*	IPv4 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV4_DST	*Required*	IPv4 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_SRC	*Required*	IPv6 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_DST	*Required*	IPv6 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_TCP_SRC	*Required*	TCP source port
OXM_OF_TCP_DST	*Required*	TCP destination port
OXM_OF_UDP_SRC	*Required*	UDP source port
OXM_OF_UDP_DST	*Required*	UDP destination port

Required Matching Fields

- OpenFlow 1.3.4: required fields

Field		Description
OXM_OF_IN_PORT	*Required*	Ingress port. This may be a physical or switch-defined logical port.
OXM_OF_ETH_DST	*Required*	Ethernet destination address. Can use arbitrary bitmask
OXM_OF_ETH_SRC	*Required*	Ethernet source address. Can use arbitrary bitmask
OXM_OF_ETH_TYPE	*Required*	Ethernet type of the OpenFlow packet payload, after VLAN tags.
OXM_OF_IP_PROTO	*Required*	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	*Required*	IPv4 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV4_DST	*Required*	IPv4 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_SRC	*Required*	IPv6 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_DST	*Required*	IPv6 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_TCP_SRC	*Required*	TCP source port
OXM_OF_TCP_DST	*Required*	TCP destination port
OXM_OF_UDP_SRC	*Required*	UDP source port
OXM_OF_UDP_DST	*Required*	UDP destination port

Layer 3

Required Matching Fields

- OpenFlow 1.3.4: required fields

Field		Description
OXM_OF_IN_PORT	*Required*	Ingress port. This may be a physical or switch-defined logical port.
OXM_OF_ETH_DST	*Required*	Ethernet destination address. Can use arbitrary bitmask
OXM_OF_ETH_SRC	*Required*	Ethernet source address. Can use arbitrary bitmask
OXM_OF_ETH_TYPE	*Required*	Ethernet type of the OpenFlow packet payload, after VLAN tags.
OXM_OF_IP_PROTO	*Required*	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	*Required*	IPv4 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV4_DST	*Required*	IPv4 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_SRC	*Required*	IPv6 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_DST	*Required*	IPv6 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_TCP_SRC	*Required*	TCP source port
OXM_OF_TCP_DST	*Required*	TCP destination port
OXM_OF_UDP_SRC	*Required*	UDP source port
OXM_OF_UDP_DST	*Required*	UDP destination port

Layer 4

OpenFlow Entry: Instructions

匹配域 (Match Fields)	优先级 (Priority)	计数器 (Counters)	**指令 (Instructions)**	失效时间 (Timeouts)	Cookie	Flags

Instructions (指令)

Apply-Actions
Write-Actions
Clear-Actions
Goto table
-

Actions (动作)

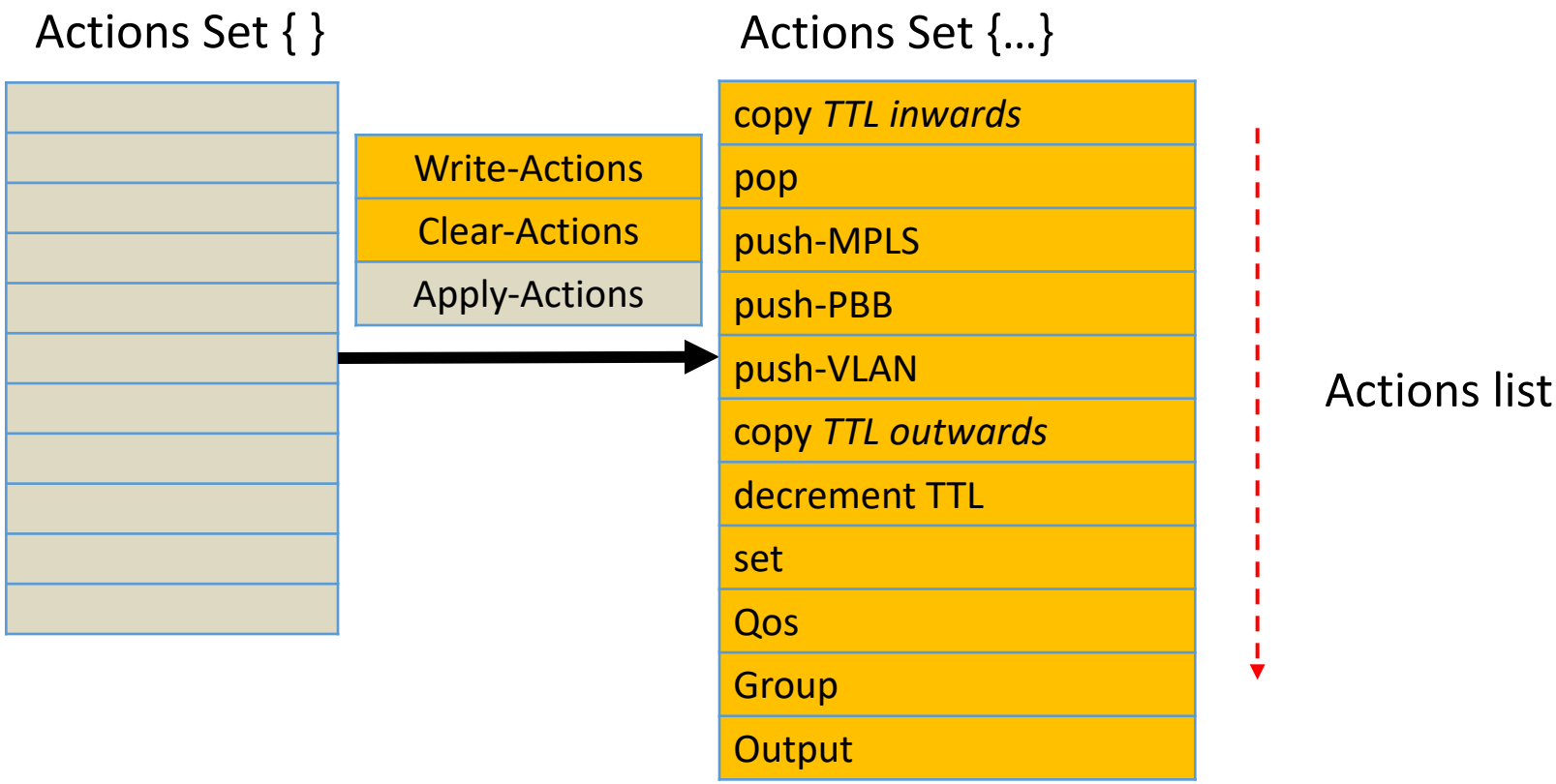
Output *port_no*
Drop
Group *group_id*
Set-Queue *queue id*
Push-Tag/Pop-Tag *ethertype*
Set-Field *field type value*
Change-TTL *ttl*
Meter *meter id*
Copy-Field *src dst field type*

OpenFlow Entry: Instructions

必须支持	描述
Goto-Table *next-table-id*	指示processing pipeline中的下一个table，其中table-id必须大于当前的table-id，即向后跳转，最后一个table不能包含该Instruction
Write-Actions *actions*	把指定的actions添加到当前的action set，如果当前set中存在系统类型的action，就重写并添加
Clear-Action	立即清除action set中的所有actions

必须支持	描述
Output *port_no*	这种action转发packet到指定openflow端口，OFS必须支持转发到物理端口，switch定义的logical端口和required保留端口
Drop	没有明确的action时，Drop
Group *group_id*	通过指定的group处理packet，精确地解释跟group type有关

Action Set


Actions Set { }
Write-Actions
Clear-Actions
Apply-Actions
Actions Set {...}
copy *TTL inwards*
pop
push-MPLS
push-PBB
push-VLAN
copy *TTL outwards*
decrement TTL
set
Qos
Group
Output
Actions list

Why Instructions and Actions

- OpenFlow 1.0 has only a single flow table
  - All wildcard entries have a priority associated with them

匹配域 (Match Fields)	计数器 (Counters)	**动作 (Actions)**

- Exposing multiple tables has many advantages
  - Better flexibility: a lot of hardware has multiple tables internally (L2 table, L3 table)
  - Smaller table: forcing orthogonal processing (MAC, routing, ACL) into a single table creates huge ruleset due to cross product of rules

Why Instructions and Actions

- OpenFlow 1.1 introduces a more flexible pipeline with multiple tables
  - Packets are processed through the *pipeline*
  - As a packet goes through the pipeline, it is associated with an *action set*
  - Actions which were directly attached to flows in previous versions are now encapsulated in *instructions*
  - Instructions may apply those actions between tables or accumulate them in the packet action set

Port Model

Actions

Output *port_no*

struct ofp_port

```
uint32_t port_no;
uint16_t length;
uint8_t pad[2];
uint8_t hw_addr[OFP_ETH_ALEN];
uint8_t pad2[2]; /* Align to 64 bits. */
char name[OFP_MAX_PORT_NAME_LEN]; /* Null-terminated */
uint32_t config;        /* Bitmap of OFPPC_* flags. */
uint32_t state;         /* Bitmap of OFPPS_* flags. */
        /* Port description property list - 0 or more properties */
struct ofp_port_desc_prop_header properties[0];
};OFP_ASSERT(sizeof(struct ofp_port) == 40);
```

enum ofp_port_no {
/* Maximum number of physical and logical switch ports*/
OFPP_MAX = 0xffffff00,

/* Reserved OpenFlow Port (fake output "ports"). */
OFPP_UNSET = 0xfffffff7, /* Output port not set in action-set.used only in OXM_OF_ACTSET_OUTPUT
OFPP_IN_PORT = 0xfffffff8, /* Send the packet out the input port. This reserved port must be explicitly used in order to send back out of the input port. */
OFPP_TABLE = 0xfffffff9, /* Submit the packet to the first flow table NB: This destination port can only be used in packet-out messages. */
OFPP_NORMAL = 0xfffffffa, /* Forward using non-OpenFlow pipeline. */
OFPP_FLOOD = 0xfffffffb, /* Flood using non-OpenFlow pipeline. */
OFPP_ALL = 0xfffffffc, /* All standard ports except input port. */
OFPP_CONTROLLER = 0xfffffffd, /* Send to controller. */
OFPP_LOCAL = 0xfffffffe, /* Local openflow "port". */
OFPP_ANY = 0xffffffff /* Special value used in some requests when no port is specified (i.e. wildcarded). */};

Port Model

必须支持	描述
All	描述通用转发模型中能转发某个指定数据包的所有端口，只能用作输出端口。在这种情况下，这个数据包会被复制然后发送给所有的标准端口，当然不包括数据包的输入端口和配置为OFPPC_NO_FWD的端口
CONTROLLER	描述控制器的控制通道，可用作输入/输出端口，当用在输出端口，将数据包封装在packet-in消息中，按照协议规定的方式发送出去；当用在输入端口，标识一个来自控制器的数据包
TABLE	描述通用转发模型处理流水线的起始位置，只有在packet-out消息操作列表的output操作中才有效，将数据包提交个流水线的第一个flow table来处理
IN_PORT	描述数据包的输入端口，只能用作输出端口，发送数据包到自己的输入端口
ANY	当OpenFlow命令没有指定端口时使用的类型，不能用作输入端口和输出端口
可选支持	**描述**
LOCAL	描述通用转发模型本地的网络栈和管理栈，可用作输入/输出端口。使得远端设备通过OpenFlow网络本身与转发模型交互，使用其网络服务，而不是通过一个独立的控制网络。采用一些默认流选项可以实现网内控制器链接，不需要独立的控制器通道。
NORMAL	描述传统的非OpenFlow转发处理流水线，只能用作输出端口，使用传统流水线处理数据包。当转发模型不支持从OpenFlow流水线到传统处理流水线的转发时，必须指定不支持这种操作
FLOOD	描述传统处理流水线中的泛洪操作，只能用作输出端口，通常发送数据包给所有的标准端口，不包括输入端口和OFPPS_BLOCKED状态的端口。转发模型需要用数据包VLAN ID选择向那些端口执行flood操作

OpenFlow Entry: Priority

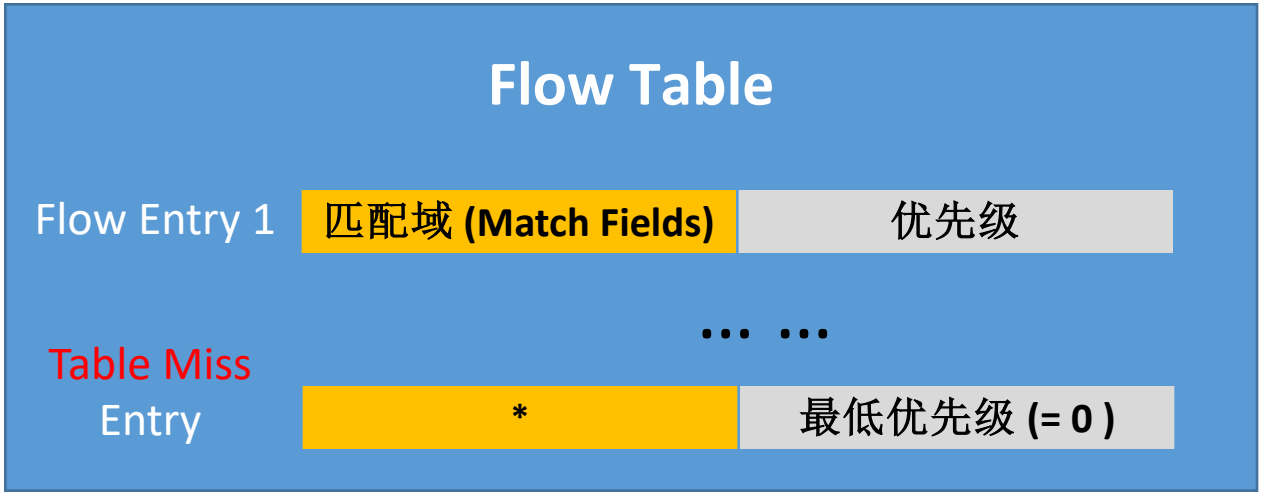
匹配域 (Match Fields)	**优先级 (Priority)**	计数器 (Counters)	指令 (Instructions)	失效时间 (Timeouts)	Cookie	Flags

0~65535, Larger values indicate higher priority

Why is priority needed here?
(compare to IP table and MAC table)

OpenFlow Entry: Priority

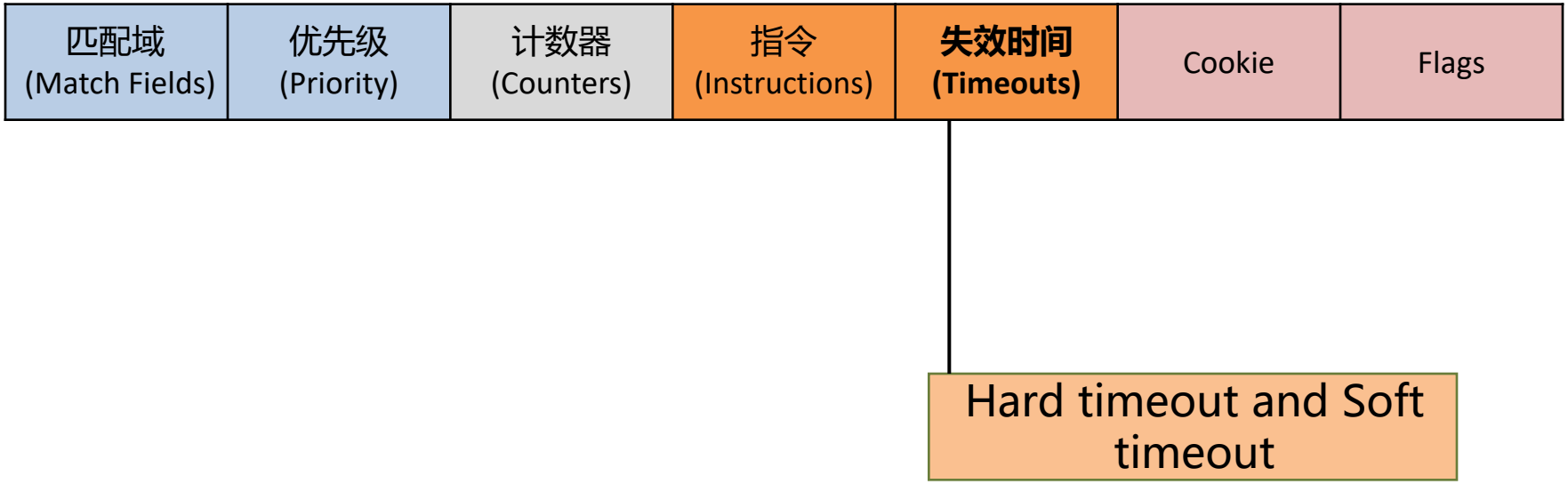
- Every flow table must support a table-miss flow entry to process table misses
  - wildcards all match fields (all fields omitted) and has the lowest priority



OpenFlow Entry: Priority

- The table-miss flow entry behaves in most ways like any other flow entry
  - it does not exist by default in a flow table
  - the controller may add it or remove it at any time
  - If the table-miss flow entry does not exist, by default packets unmatched by flow entries are dropped (discarded)
  - Question: does Ryu installs table-miss flow entry, and if so, when?

OpenFlow Entry: Timeout



the flow entry will timeout after **idle_timeout** seconds with no traffic, or **hard_timeout** seconds, whichever comes first.

OpenFlow Entry: Counter

匹配域 (Match Fields)	优先级 (Priority)	**计数器 (Counters)**	指令 (Instructions)	失效时间 (Timeouts)	Cookie	Flags

Per Flow Entry	
Received Packets	64
Received Bytes	64
Duration (s，ns)	32

OpenFlow Switches

- Hardware-based OpenFlow Switches
  - Commercial hardware switches with OpenFlow capability
    - Network abstraction is realized by firmware upgrading
  - Show high processing speed
  - Have space limitation on saving the flow table entries
  - Not easy to upgrade
    - Most switches only support OpenFlow up to version 1.0
- Software-based OpenFlow Switches
  - OpenFlow enabled software switch (runs on x86 commodity computer)
  - Performance is relatively low
  - Store large amount of flow entries without bound
  - Under active development, support most recent OpenFlow spec.

Hardware-basd OpenFlow Switches

Juniper MX-series

NEC IP8800

WiMax (NEC)

HP Procurve 5400

Netgear 7324

PC Engines

Pronto 3240/3290

Ciena Coredirector

Software-based OpenFlow Switches

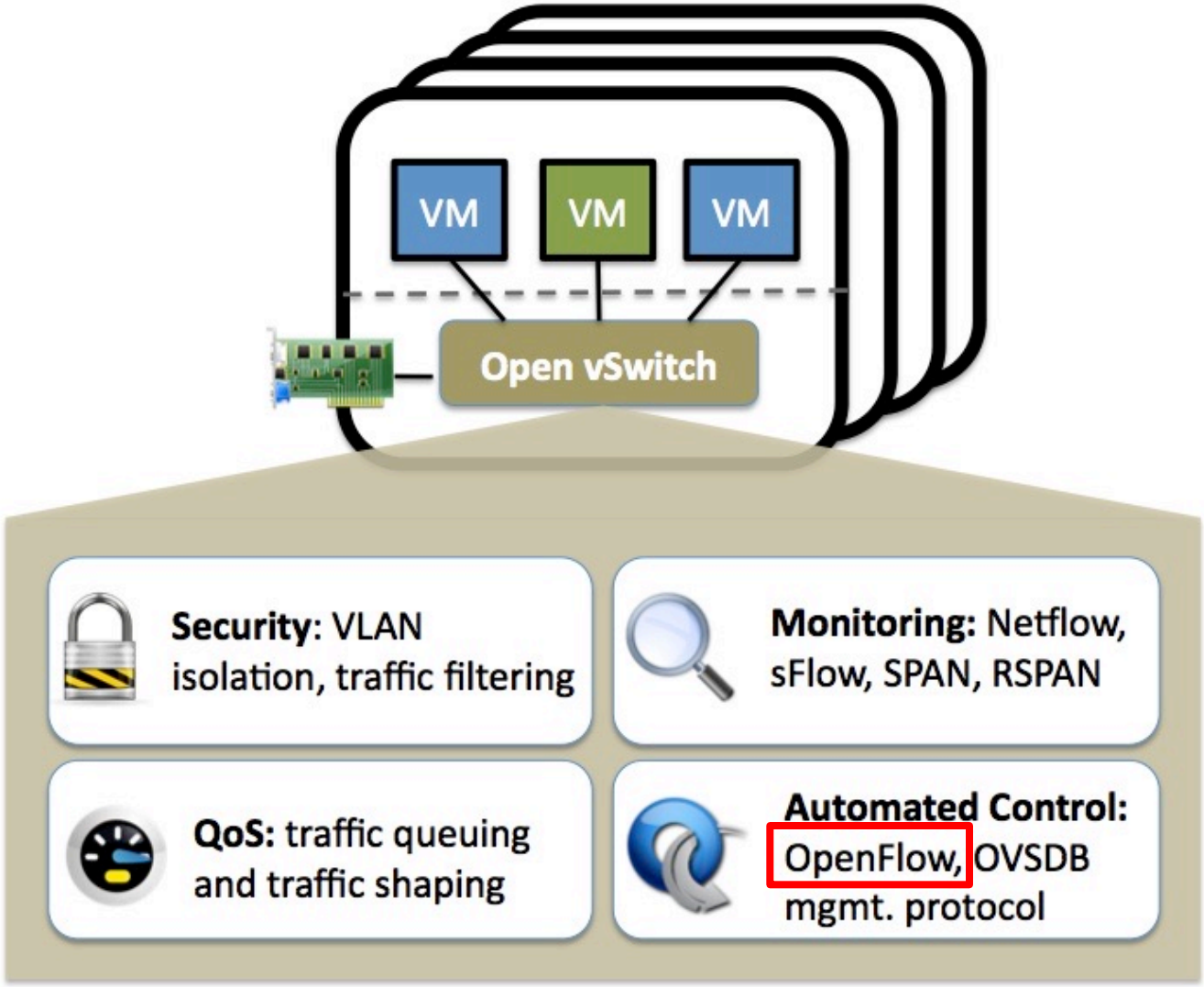
- OpenFlow Software Switch
  - An OpenFlow compatible user-space software switch implementation
  - The original code is based on the Stanford 1.0 reference switch
  - Initially developed by Ericsson Research Lab
  - The implementation is feature-complete, and passes the available oftest 1.1 test case
    - Compatible with most of OpenFlow controllers which support OF 1.1
  - CPqD version supports OpenFlow up to v1.3

- Open vSwitch (OVS)
  - A virtual switch
  - User-space: configuration, control
  - Kernel-space: datapath (included in main Linux kernel from v3.3)

Where is OVS Used?

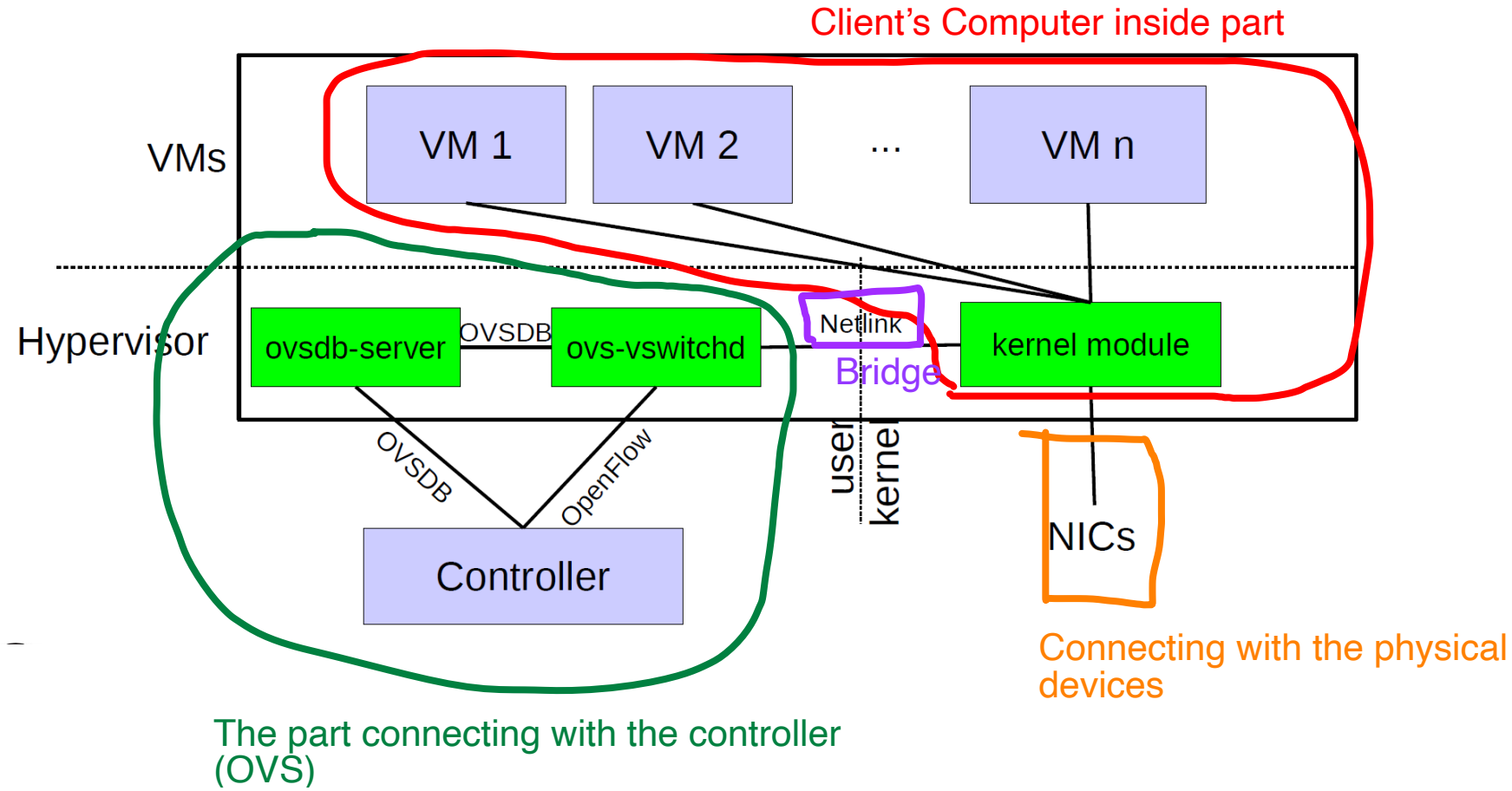
- Broad support
  - Linux, FreeBSD, NetBSD, Windows, ESX
  - KVM, Xen, Docker, VirtualBox, Hyper-V, …
  - OpenStack, CloudStack, OpenNebula, …
- Widely used
  - Most popular OpenStack networking backend
  - Default network stack in XenServer
  - Thousands of subscribers to OVS mailing lists

Open vSwitch: A First Look

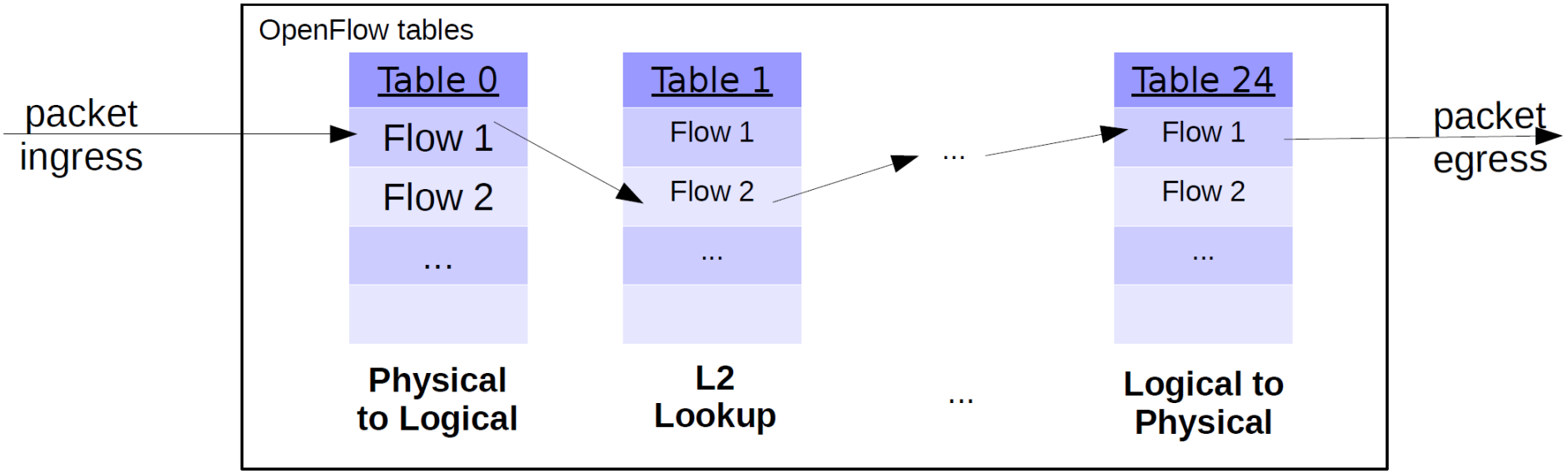


Open vSwitch Architecture

- Both kernel and user space



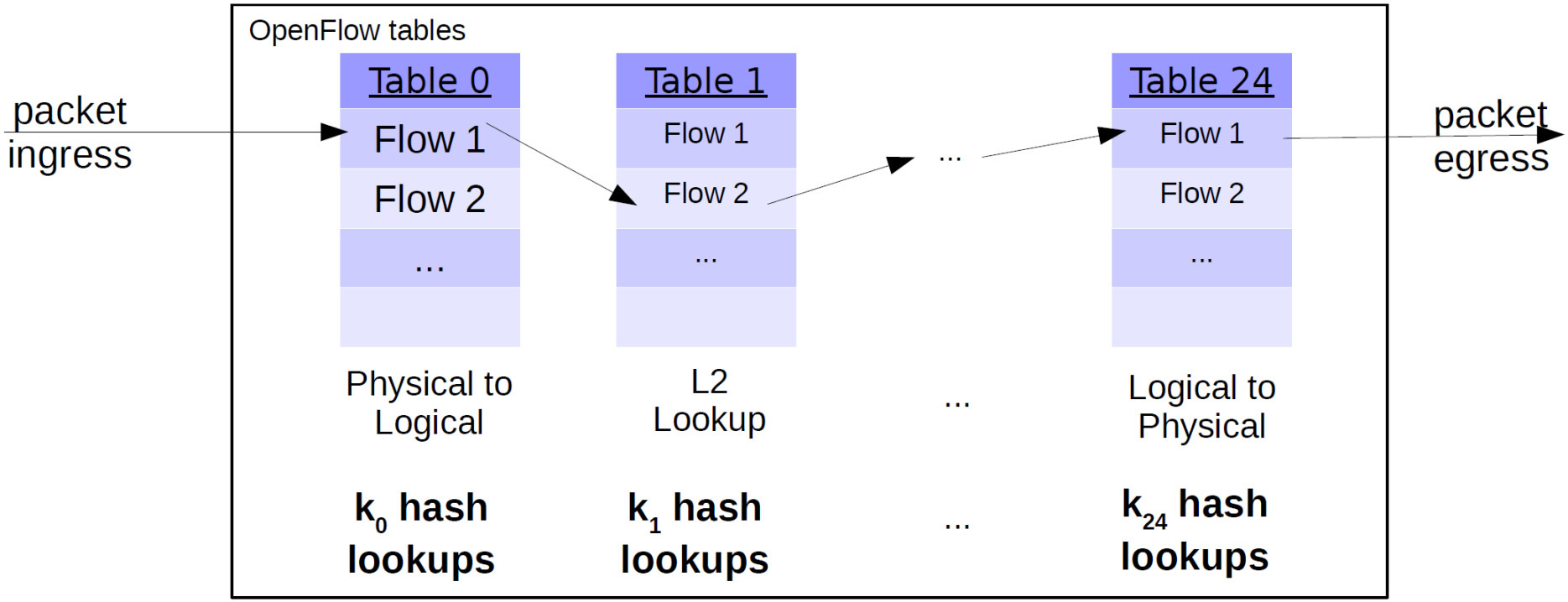
OpenFlow Pipeline



OpenFlow Pipeline

OpenFlow Pipeline

- Classification is expensive on general-purpose CPUs!



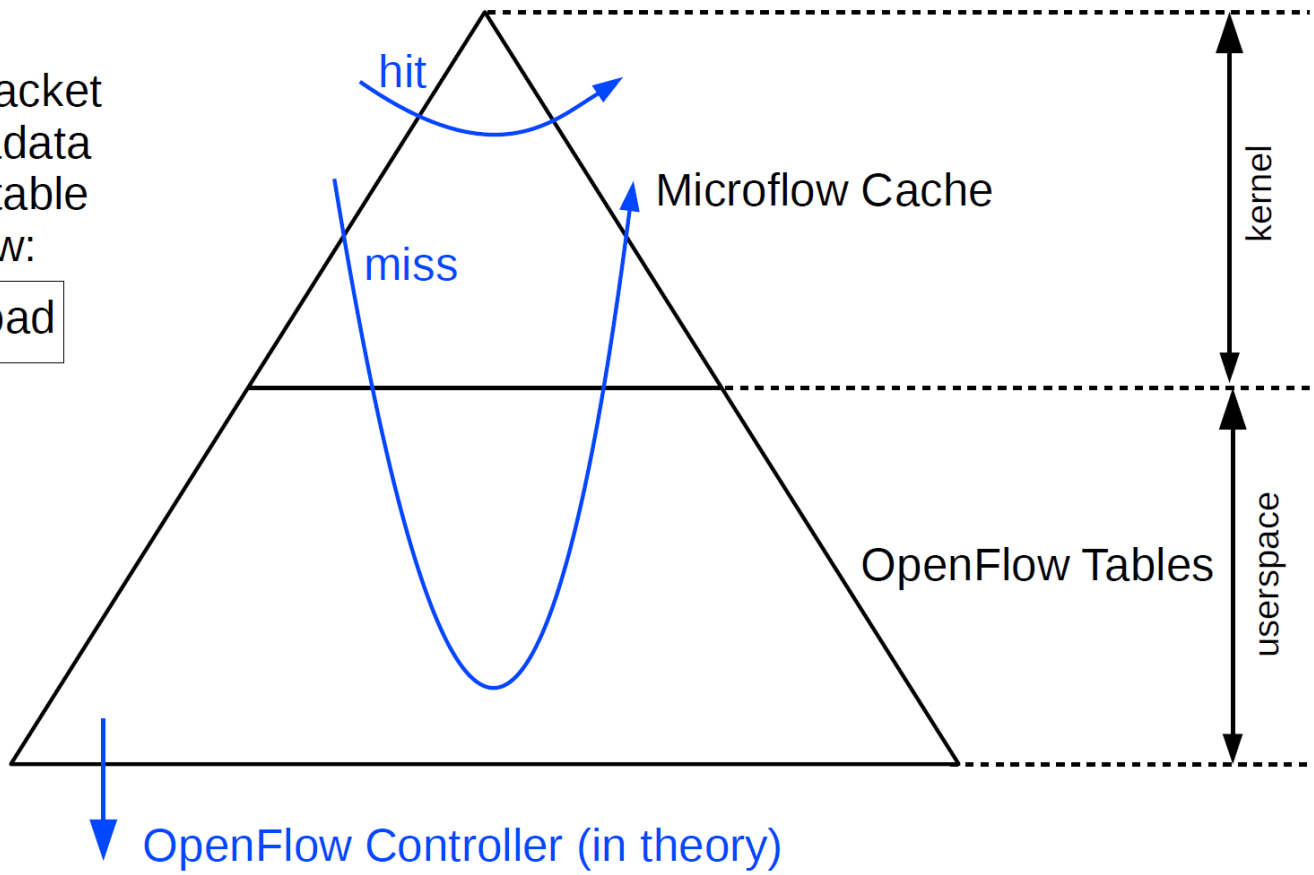
100+ hash lookups per packet for tuple space search?

OVS Cache v1: Microflow Cache

Microflow:

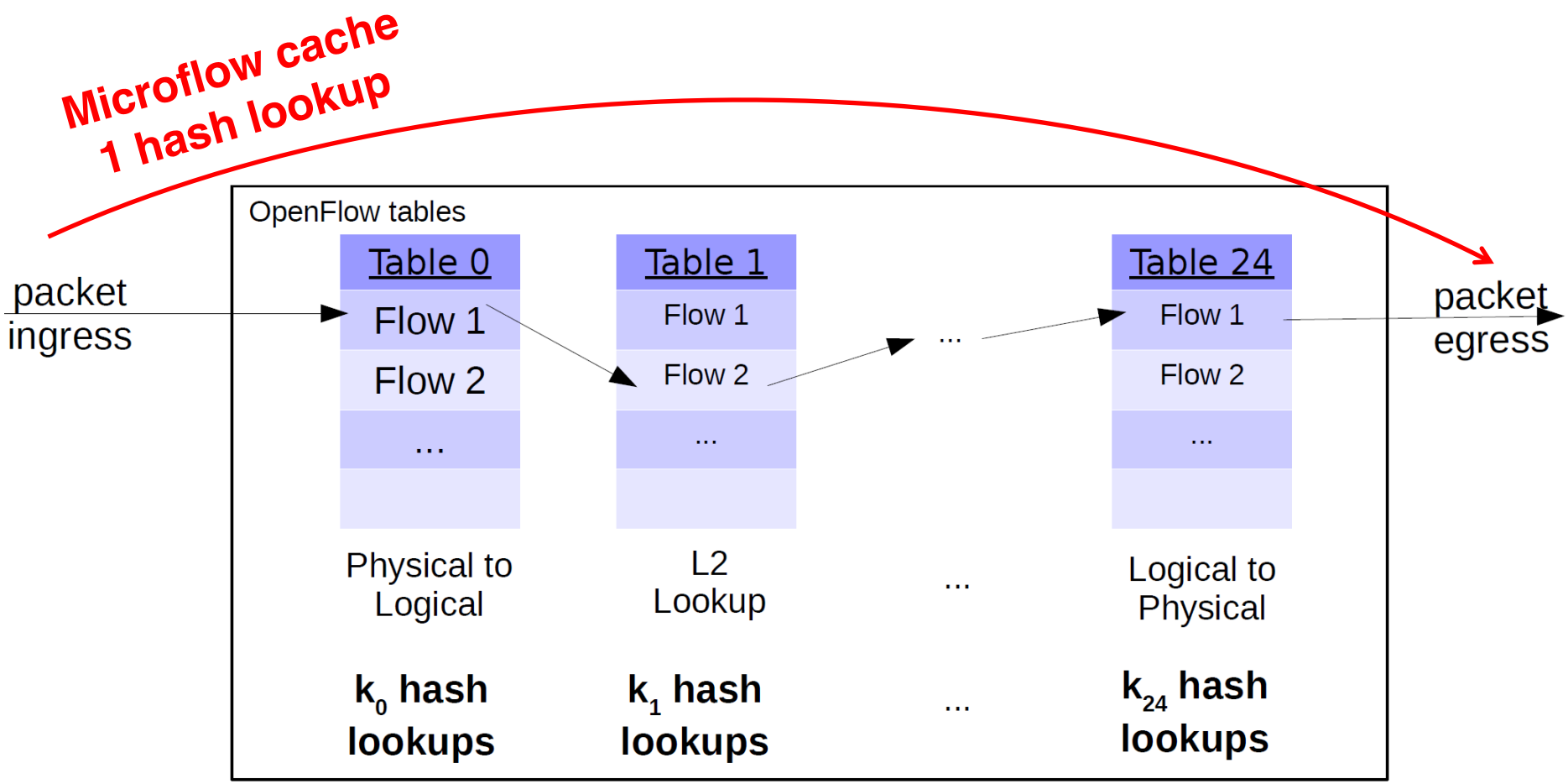
- Complete set of packet headers and metadata
- Suitable for hash table
- Shaded data below:

Eth	IP	TCP	payload



Open vSwitch

Speedup with Microflow Cache



Problems

- Lots of short-lived microflows => Fundamental caching problem: low hit rate

Naive Approach to Populating Cache

- Combine tables 0...24 into one flow table. Easy! Usually, kc << k0 + k1 + … + k24. But:

Table 0
ip_src=*a*
ip_src=*b*
ip_src=*c*
ip_src=*d*

n_1 flows

×

Table 1
ip_dst=*e*
ip_dst=*f*
ip_dst=*g*
ip_dst=*h*

n_2 flows

× ··· ×

Table 24
eth_dst=*i*
eth_dst=*j*
eth_dst=*k*
eth_dst=*m*

n_{24} flows

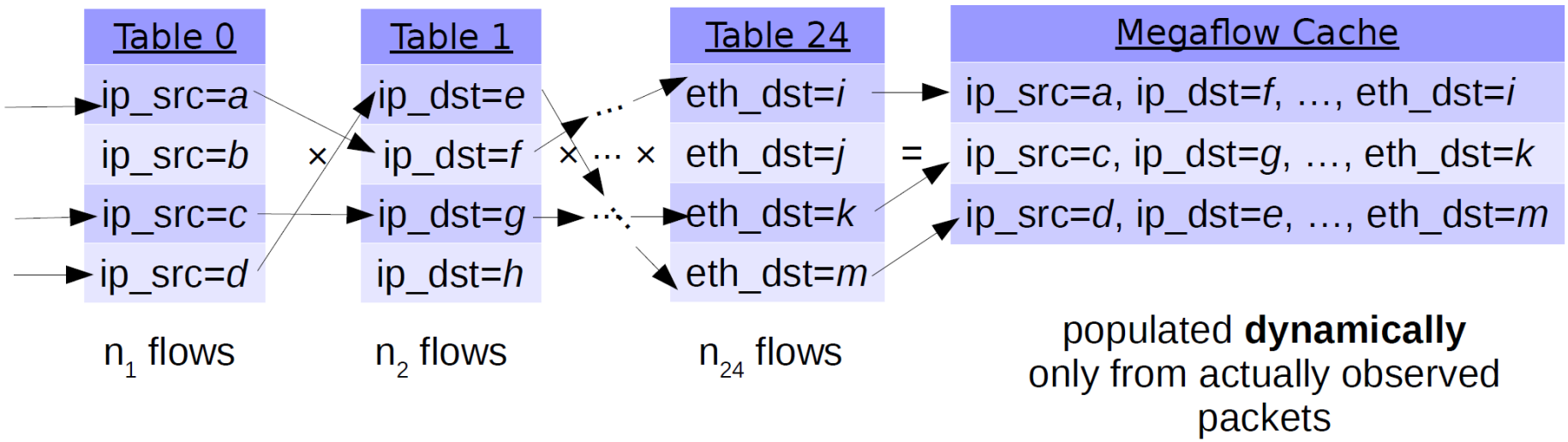
=

Table 0+1+…+24
ip_src=*a*, ip_dst=*e*, …, eth_dst=*i*
ip_src=*a*, ip_dst=e, …, eth_dst=*j*
ip_src=*a*, ip_dst=*e*, …, eth_dst=*k*
…
ip_src=*d*, ip_dst=*h*, …, eth_dst=*k*
ip_src=*d*, ip_dst=*h*, …, eth_dst=*m*

up to $n_1 \times n_2 \times \cdots \times n_{24}$ flows

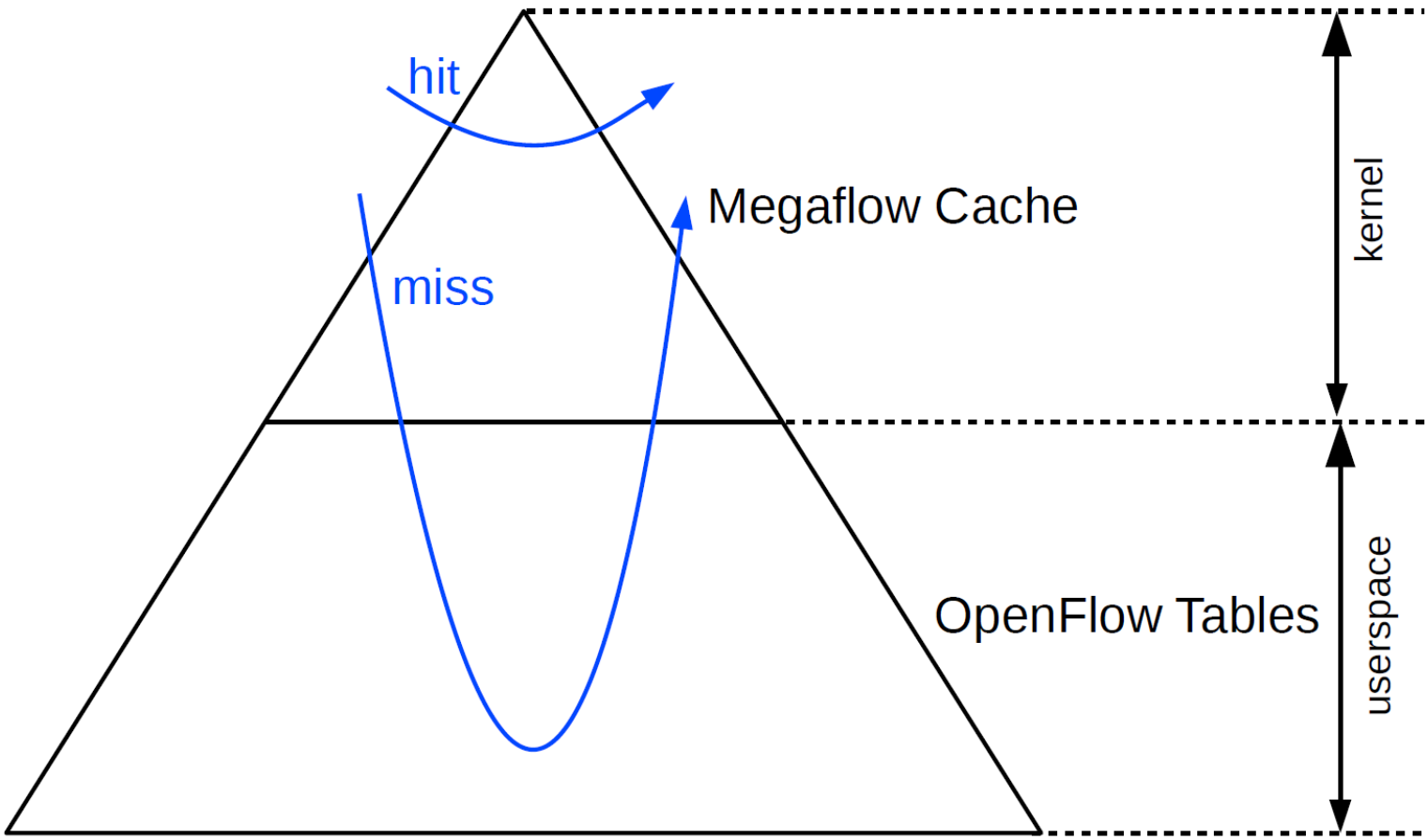
Lazy Approach to Populating Cache

- Solution: Build cache of combined "megaflows" **lazily** as packets arrive.



Same (or better!) table lookups as naive approach.
Traffic locality yields practical cache size.

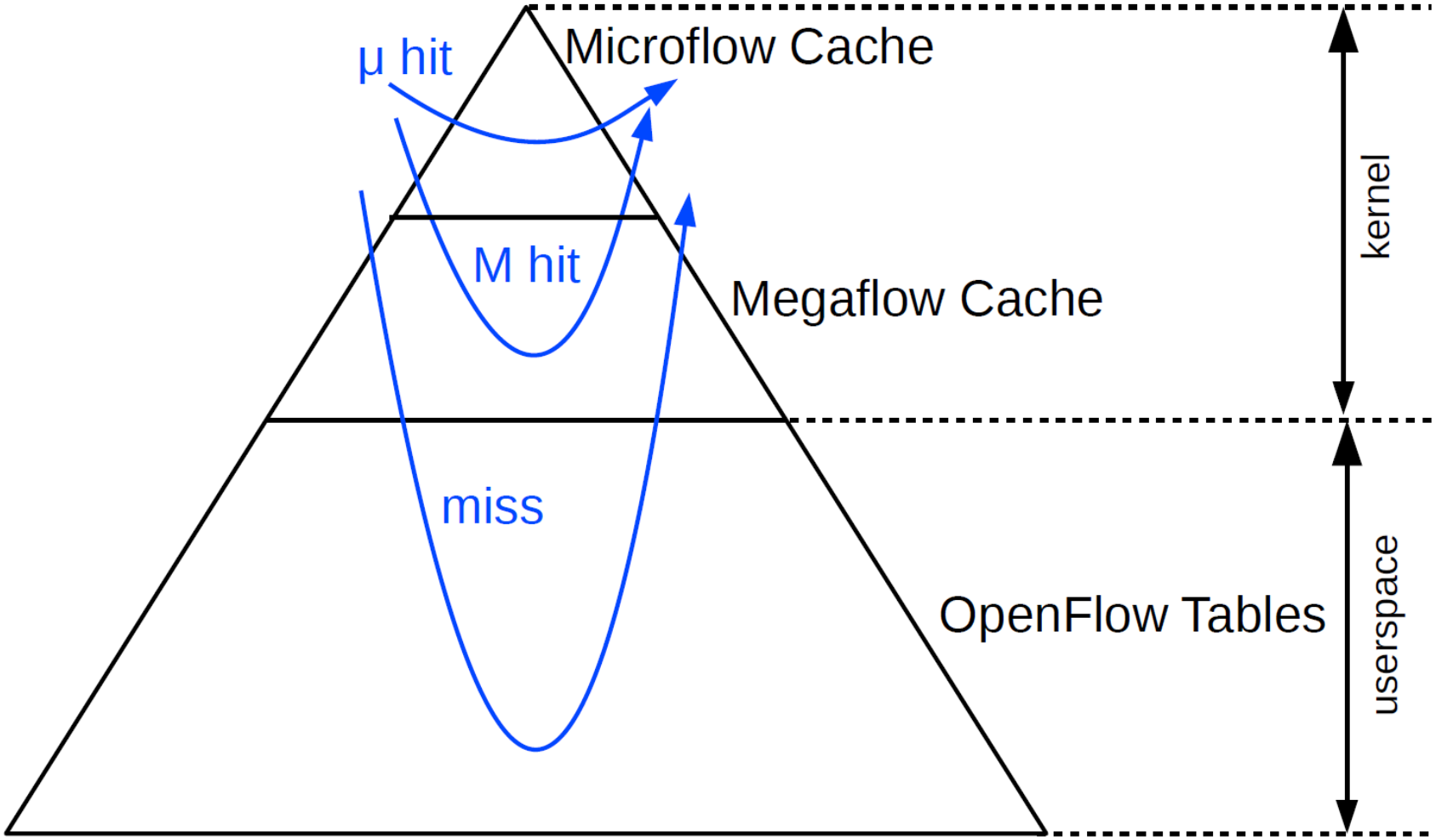
OVS Cache v2: “Megaflow” Cache



Megaflow vs. Microflow Performance

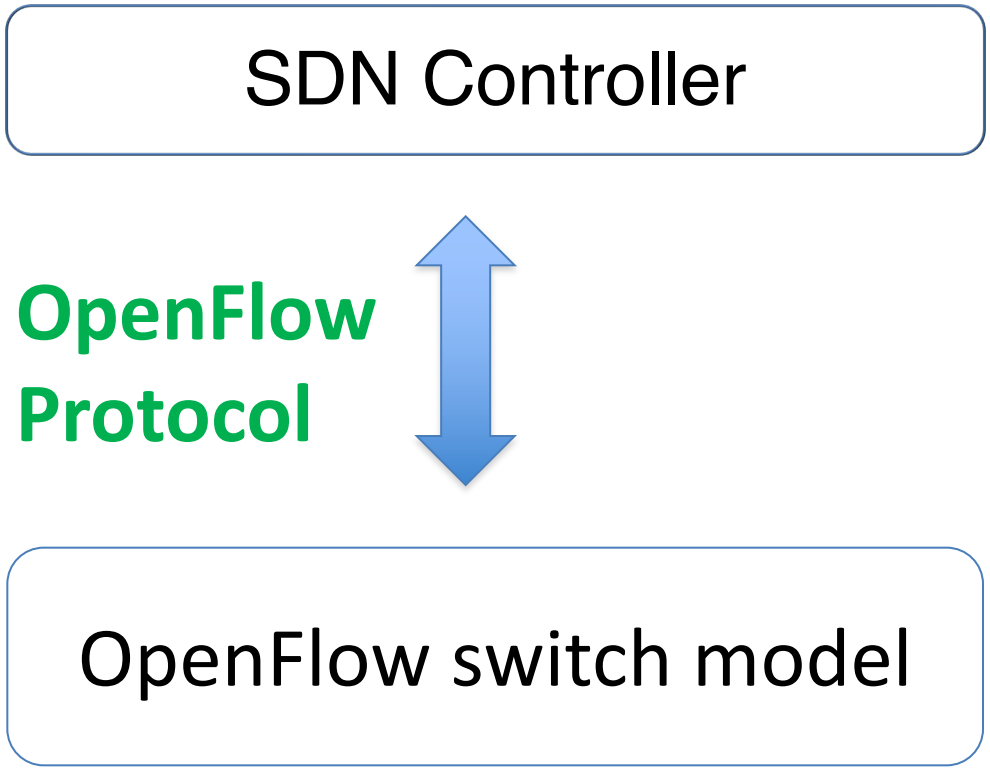
- Microflow cache
  - k0 + k1 + ⋯ + k24 lookups for first packet in microflow
  - 1 lookup for later packets in microflow
- Megaflow cache
  - kc lookups for (almost) every packet
- kc > 1 is normal, so megaflows perform worse in common case!
- Best of both worlds would be:
  - kc lookups for first packet in microflow
  - 1 lookup for later packets in microflow

OVS Cache v3: Dual Caches



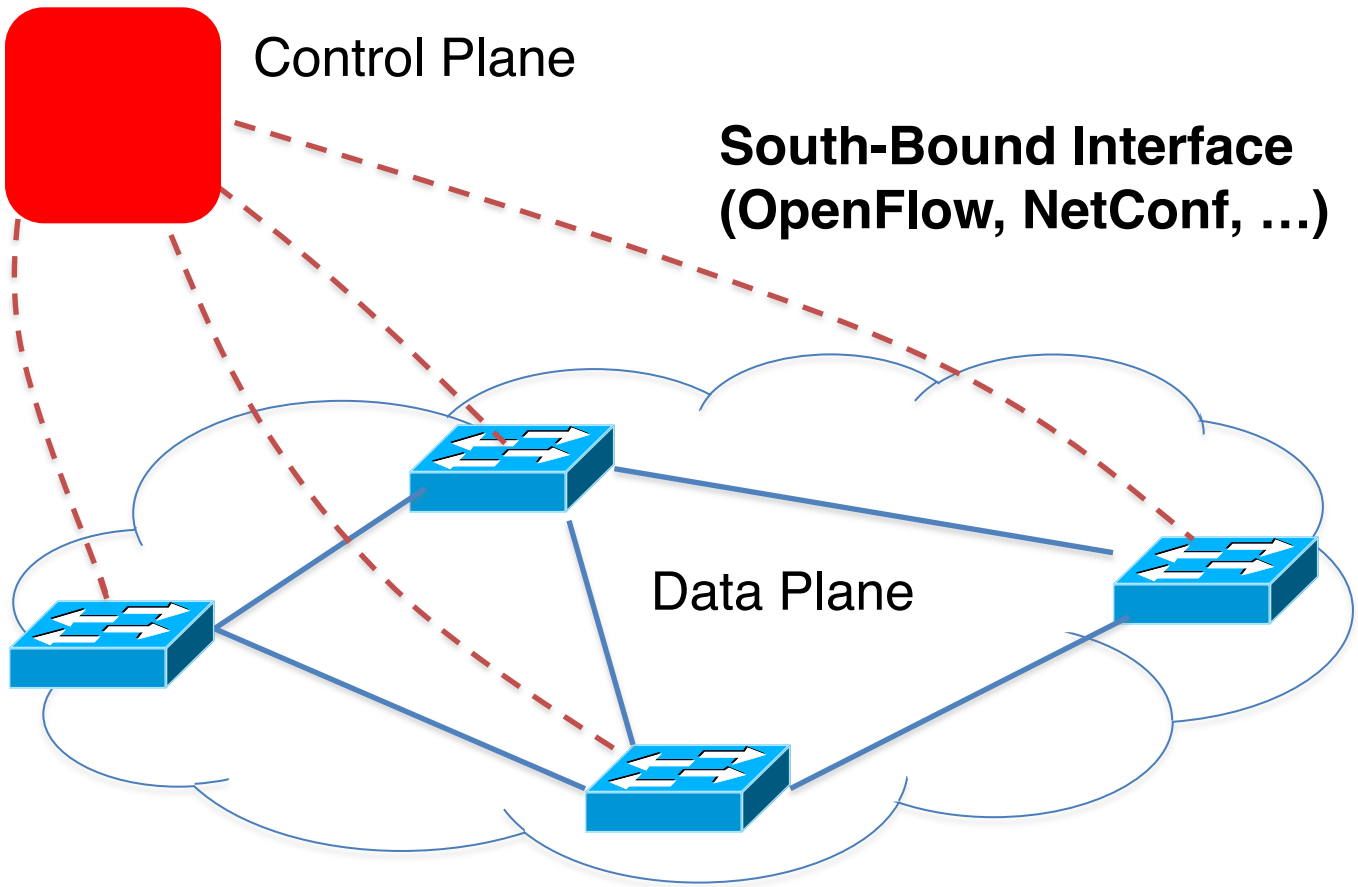
OpenFlow Protocol

- OpenFlow protocol is one south-bound interface for SDN



South-Bound Interface

- The interface between the data plane and control plane



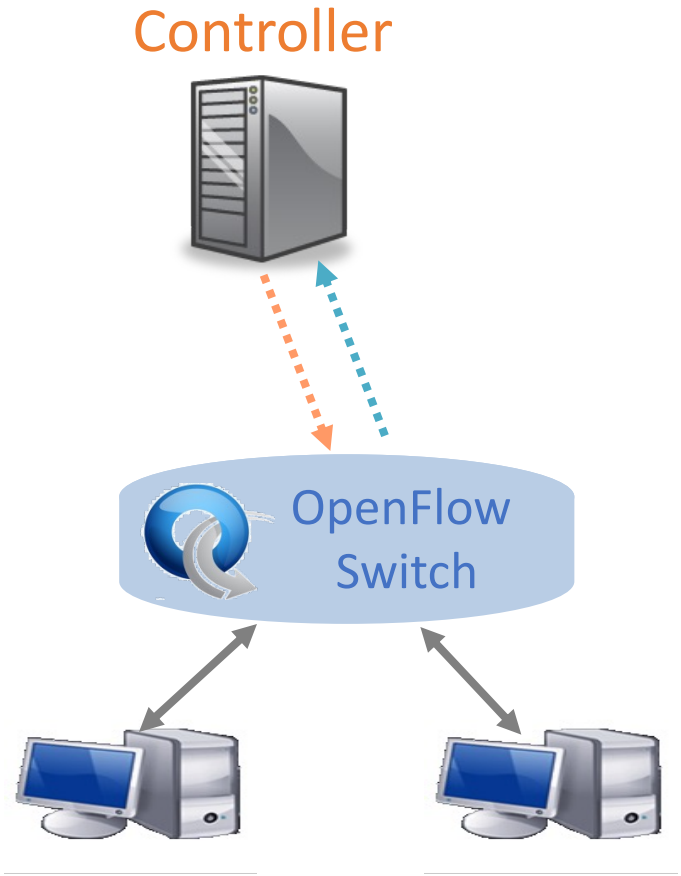
OpenFlow Protocol

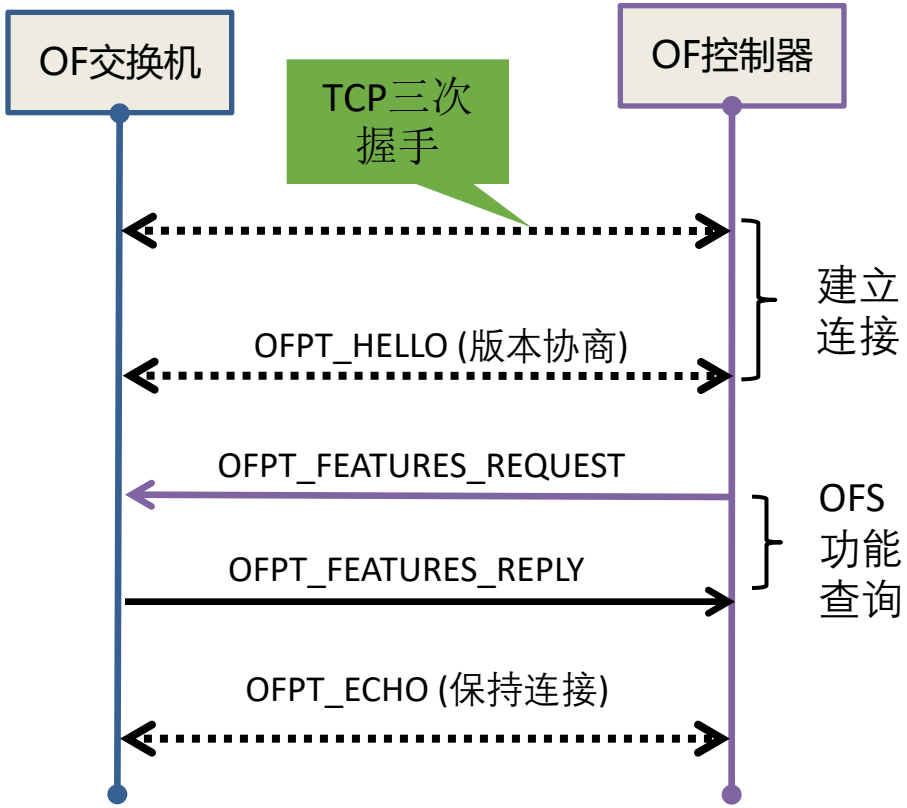
(v1.3.4)

OpenFlow Messages

消息类型	消息例子	描述
Controller-to-Switch	• FlowMod • Packet out • Switch Configuration • Switch Features	• 添加、修改或删除流表项 • 将数据包发送出指定交换机端口 • 配置交换机 • 查询交换机的功能和统计
Asynchronous（异步）	• Packet In • Flow Removed • Port Status	• 没有匹配交换机中任意流表项，通知控制器 • 流表项删除，通知控制器 • 端口状态改变，通知控制器
Symmetric（对称）	• Hello • Echo • Experimenter	• 控制器和交换机建立连接时使用 • 用来确定控制器和交换机连接是否活跃 • 用来消息扩展

OpenFlow Session Setup

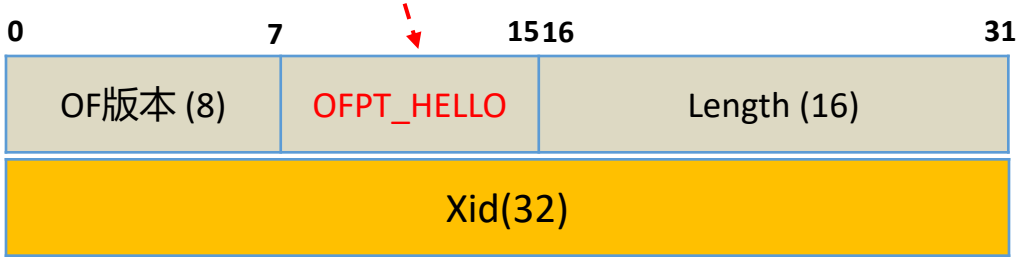


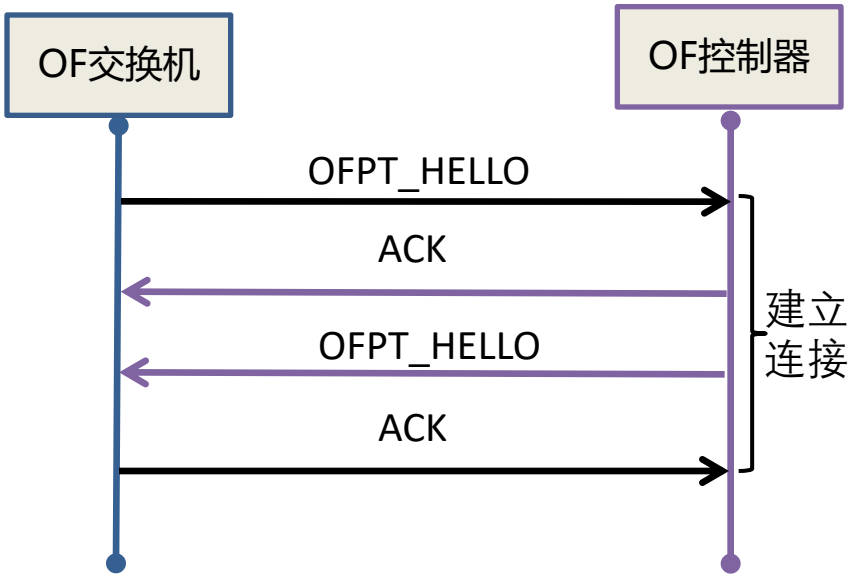


Hello Message

struct ofp_hello

```
struct ofp_header header;
/* Hello element list */
struct ofp_hello_elem_header elements[0]; /*
List of elements - 0 or more */
};OFP_ASSERT(sizeof(struct ofp_hello) == 8);
```

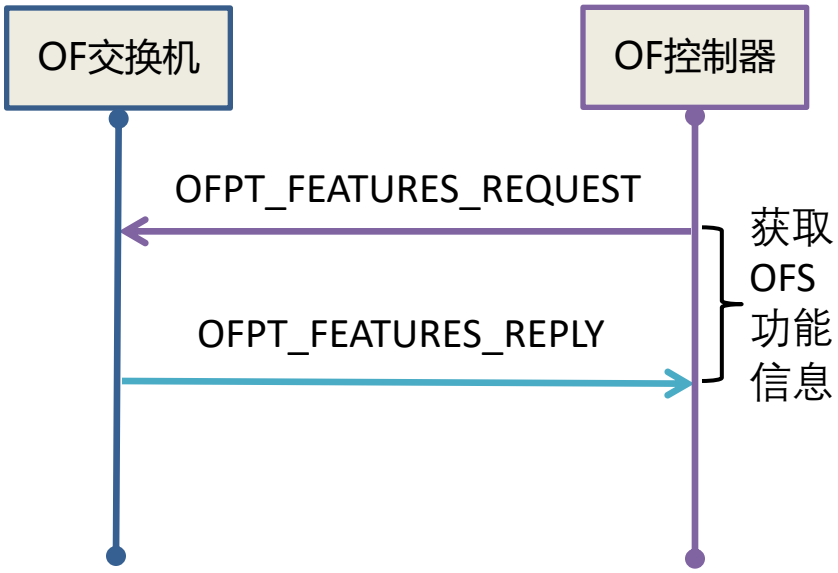




Switch Features Messages

struct ofp_switch_features

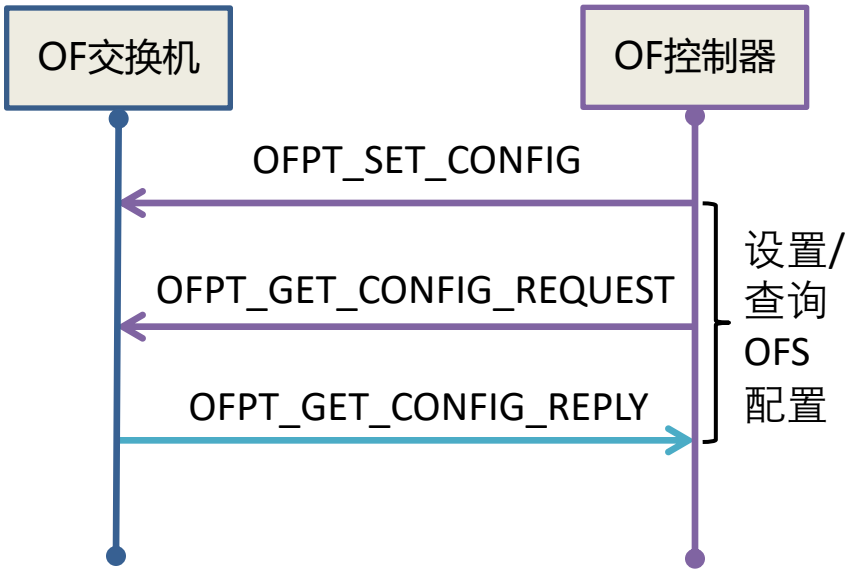
```
struct ofp_header header;
uint64_t datapath_id; /* Datapath unique ID. The lower
48-bits are for a MAC address, while the upper 16-bits
are implementer-defined. */
uint32_t n_buffers;   /* Max packets buffered at once.
*/
uint8_t n_tables;     /* Number of tables supported by
datapath. */
uint8_t auxiliary_id; /* Identify auxiliary connections */
uint8_t pad[2];       /* Align to 64-bits. */
/* Features. */
uint32_t capabilities; /* Bitmap of support
"ofp_capabilities". */
uint32_t reserved;
};OFP_ASSERT(sizeof(struct ofp_switch_features) ==
32);
```



Switch Config Messages

struct ofp_switch_config

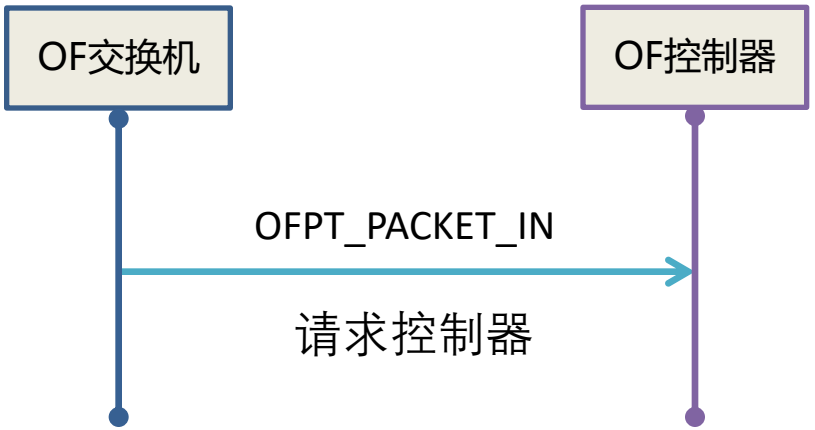
```
struct ofp_header header;
uint16_t flags; /* Bitmap of OFPC_* flags. */
uint16_t miss_send_len; /* Max bytes of packet that
datapath should send to the controller. See
ofp_controller_max_len for valid values.*/
};OFP_ASSERT(sizeof(struct ofp_switch_config) == 12);
```




```
enum ofp_config_flags {
    /* Handling of IP fragments. */
    OFPC_FRAG_NORMAL = 0, /* No special handling for fragments. */
    OFPC_FRAG_DROP = 1 << 0, /* Drop fragments. */
    OFPC_FRAG_REASM = 1 << 1, /* Reassemble (only if OFPC_IP_REASM set).
    */
    OFPC_FRAG_MASK = 3, /* Bitmask of flags dealing with frag. */
};
```

Packet-in Messages

```
struct ofp_packet_in
struct ofp_header header;
uint32_t buffer_id; /* ID assigned by datapath. */
uint16_t total_len; /* Full length of frame. */
uint8_t reason; /* Reason packet is being sent (one of OFPR_*) */
uint8_t table_id; /* ID of the table that was looked up */
uint64_t cookie; /* Cookie of the flow entry that was looked up. */
struct ofp_match match;
 };OFP_ASSERT(sizeof(struct ofp_packet_in) == 32);
```

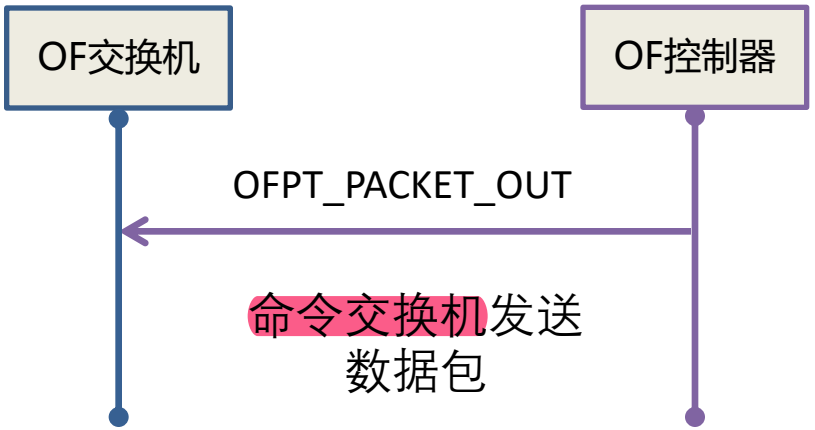



```
enum ofp_packet_in_reason {
OFPR_TABLE_MISS = 0, /* No matching flow (table-miss flow entry). */
OFPR_APPLY_ACTION = 1, /* Output to controller in apply-actions. */
OFPR_INVALID_TTL = 2, /* Packet has invalid TTL */
OFPR_ACTION_SET = 3, /* Output to controller in action set. */
OFPR_GROUP = 4, /* Output to controller in group bucket. */
OFPR_PACKET_OUT = 5, /* Output to controller in packet-out. */};
```

Packet-out Messages

struct ofp_packet_out

```
struct ofp_header header;
uint32_t buffer_id; /* ID assigned by datapath
(OFP_NO_BUFFER if none). */
uint16_t actions_len; /* Size of action array in bytes. */
uint8_t pad[2]; /* actions_lenAlign to 64 bits. */
struct ofp_match match; /* Packet pipeline fields.
Variable size. */
/* The variable size and padded match is followed by
the list of actions. */
/* struct ofp_action_header actions[0]; *//* Action list
- 0 or more. */
/* The variable size action list is optionally followed by
packet data.
*This data is only present and meaningful if buffer_id
== -1. */
/* uint8_t data[0]; */ /* Packet data. The length is
inferred from the length field in the header. */
};OFP_ASSERT(sizeof(struct ofp_packet_out) == 24);
```



FlowMod Messages

struct ofp_flow_mod

```
struct ofp_header header;
… …
uint8_t table_id;
uint8_t command; /* One of OFPFC_*. */
uint16_t idle_timeout;
uint16_t hard_timeout;
uint16_t priority;
uint32_t buffer_id; /* Buffered packet to apply to,
or OFP_NO_BUFFER */
… …
struct ofp_match match /* Fields to match. Variable
size. */
//struct ofp_instruction instructions[0]; /*
Instruction set - 0 or more. The length of the
instruction set is inferred from the
length field in the header. */
};OFP_ASSERT(sizeof(struct ofp_flow_mod) == 56);
```

Apply the actions to the packet buffered @buffer_id, save one packet-out message

FlowMod Messages

struct ofp_flow_mod

```
struct ofp_header header;
… …
uint8_t table_id;
uint8_t command; /* One of OFPFC_*. */
uint16_t idle_timeout;
uint16_t hard_timeout;
uint16_t priority;
uint32_t buffer_id; /* Buffered packet to apply to,
or OFP_NO_BUFFER */
… …
struct ofp_match match /* Fields to match.
Variable size. */
//struct ofp_instruction instructions[0]; /*
Instruction set - 0 or more. The length of the
instruction set is inferred from the
length field in the header. */
};OFP_ASSERT(sizeof(struct ofp_flow_mod) == 56);
```

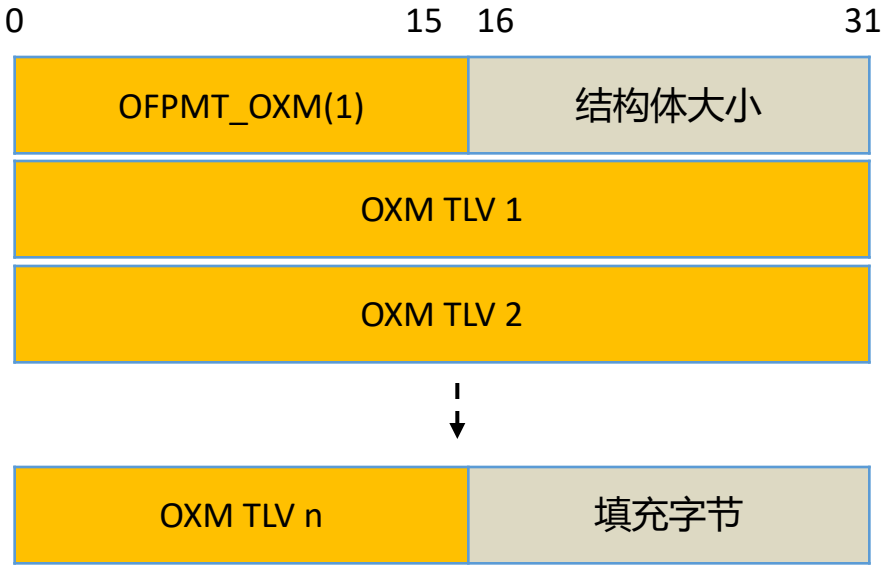
OpenFlow eXtensible Match (OXM)

- OXM = compact TLV (type-length-value) format

匹配域
(Match Fields)

struct ofp_match

```
uint16_t type;          /* One of OFPMT_* */
uint16_t length;        /* Length of ofp_match
(excluding padding) */
uint8_t oxm_fields[0];  /* 0 or more OXM match
fields */
uint8_t pad[4];    /* Zero bytes - see above for
sizing */
```




```
enum ofp_match_type {
OFPMT_STANDARD = 0,   /* Deprecated. */
OFPMT_OXM = 1,        /* OpenFlow Extensible Match */};
```

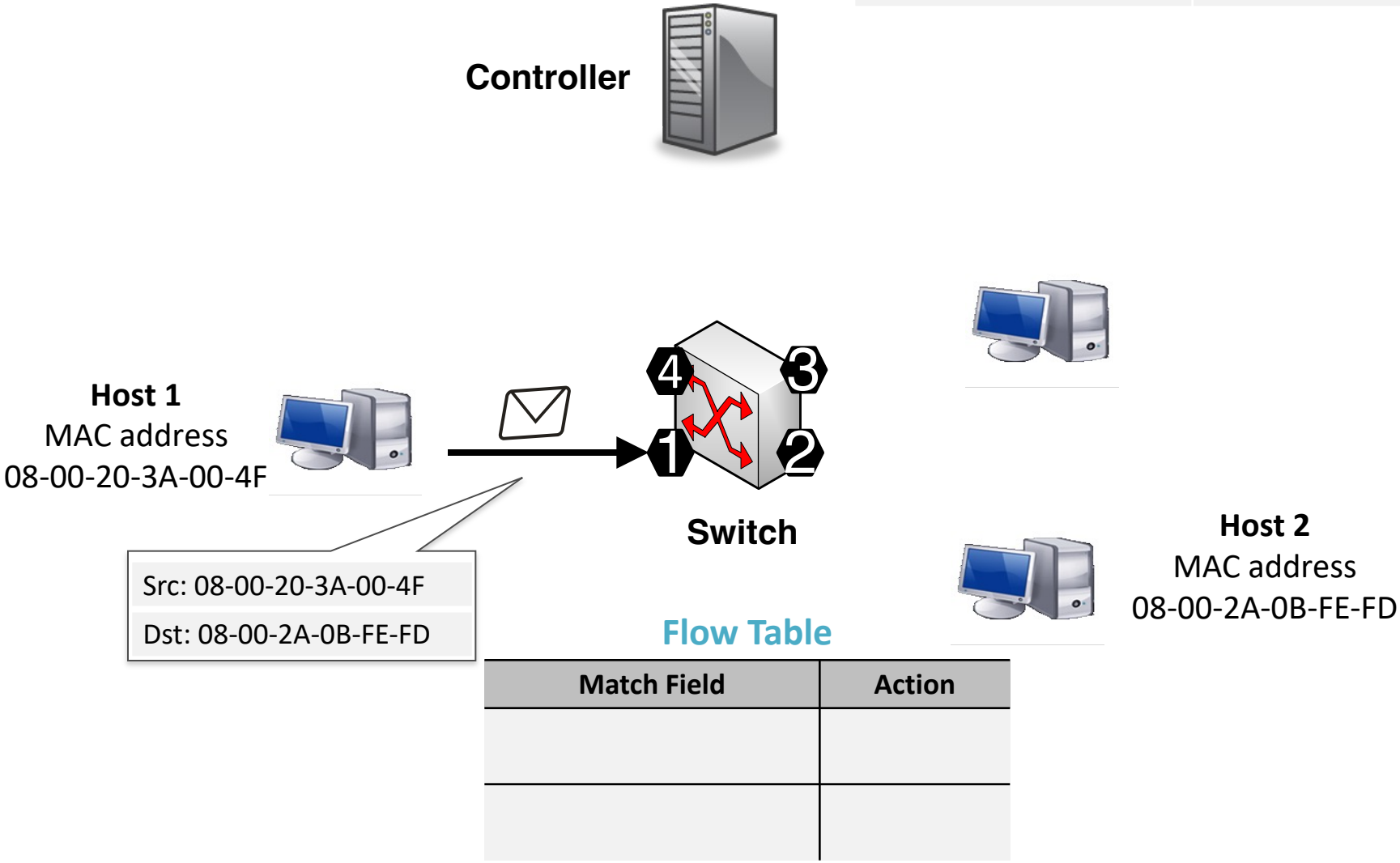
FlowMod Messages

struct ofp_flow_mod

```
struct ofp_header header;
… …
uint8_t table_id;
uint8_t command; /* One of OFPFC_*. */
uint16_t idle_timeout;
uint16_t hard_timeout;
uint16_t priority;
uint32_t buffer_id; /* Buffered packet to apply to,
or OFP_NO_BUFFER */
… …
struct ofp_match match /* Fields to match. Variable
size. */
//struct ofp_instruction instructions[0]; /*
Instruction set - 0 or more. The length of the
instruction set is inferred from the
length field in the header. */
};OFP_ASSERT(sizeof(struct ofp_flow_mod) == 56);
```

MAC table

MAC address	Ingress port

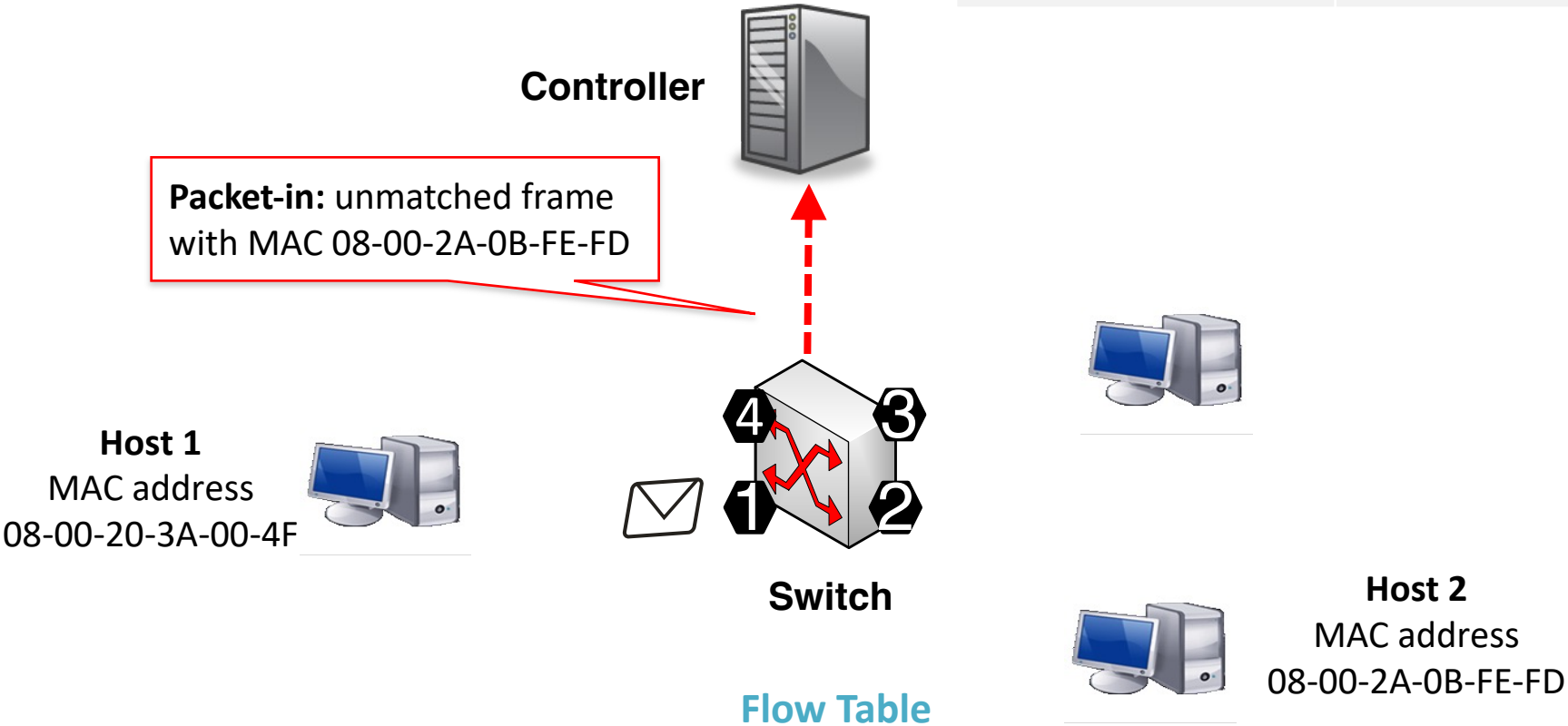


Flow Table

Match Field	Action

MAC table

MAC address	Ingress port

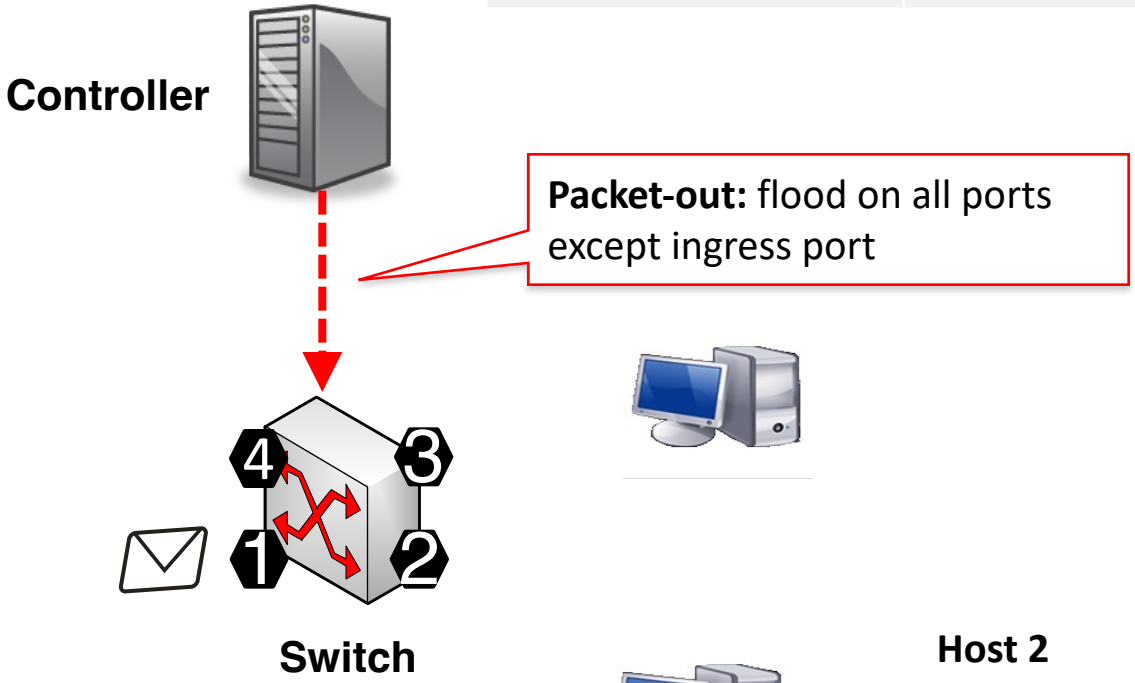


Flow Table

Match Field	Action

MAC table

MAC address	Ingress port
08-00-20-3A-00-4F	1



Flow Table

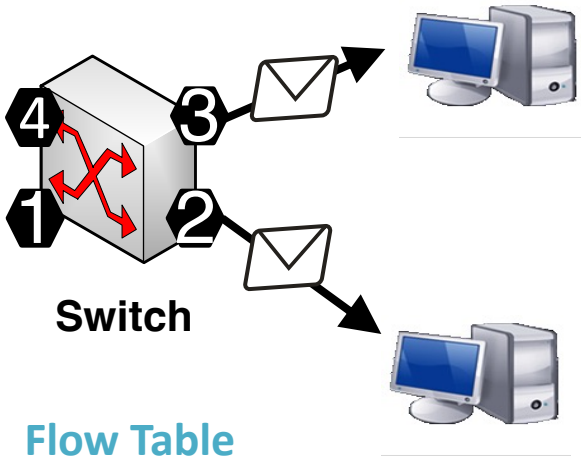
Match Field	Action

MAC table

MAC address	Ingress port
08-00-20-3A-00-4F	1

Controller

Host 1
MAC address
08-00-20-3A-00-4F



Host 2
MAC address
08-00-2A-0B-FE-FD

Flow Table

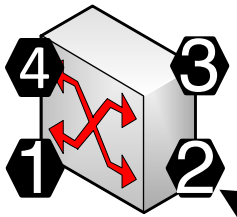
Match Field	Action

MAC table

MAC address	Ingress port
08-00-20-3A-00-4F	1

Controller

Host 1
MAC address
08-00-20-3A-00-4F



Switch

Src: 08-00-2A-0B-FE-FD
Dst: 08-00-20-3A-00-4F

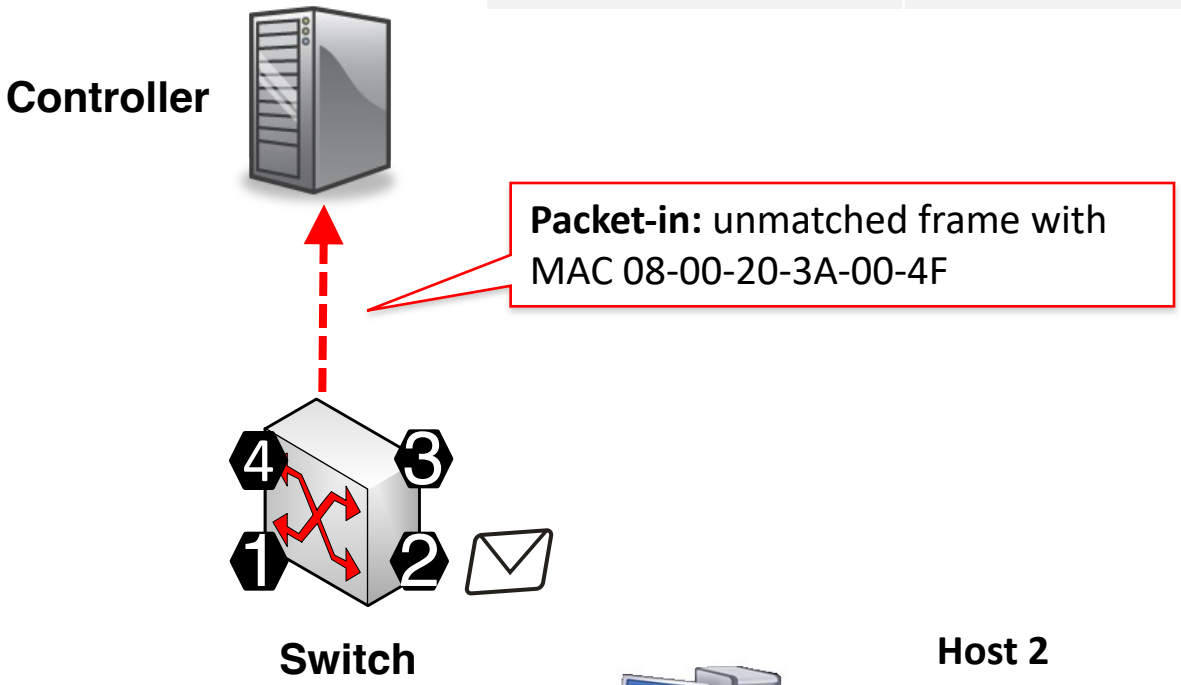
Host 2
MAC address
08-00-2A-0B-FE-FD

Flow Table

Match Field	Action

MAC table

MAC address	Ingress port
08-00-20-3A-00-4F	1
08-00-2A-0B-FE-FD	2



Flow Table

Match Field	Action

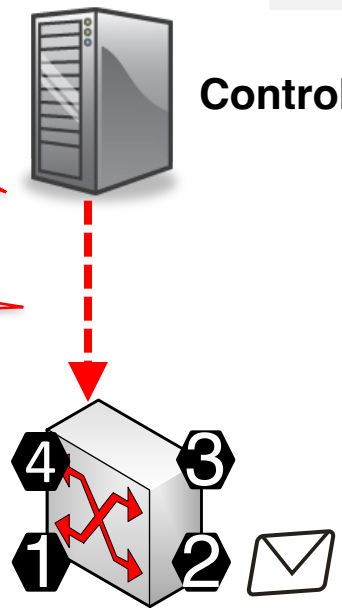
MAC table

MAC address	Ingress port
08-00-20-3A-00-4F	1
08-00-2A-0B-FE-FD	2

Flow-mod messages

Match	Action
Src: 08-00-2A-0B-FE-FD Dst: 08-00-20-3A-00-4F	Forward on port 1

Match	Action
Src: 08-00-20-3A-00-4F Dst: 08-00-2A-0B-FE-FD	Forward on port 2



Host 2
MAC address
08-00-2A-0B-FE-FD

struct ofp_flow_mod

```
uint64_t cookie;
uint64_t cookie_mask;
uint8_t table_id;
uint8_t command;
uint16_t idle_timeout;
uint16_t hard_timeout;
uint16_t priority;
uint32_t buffer_id;
uint32_t out_port;
uint32_t out_group;
uint16_t flags;
uint16_t importance;
struct ofp_match match;
```

Flow Table

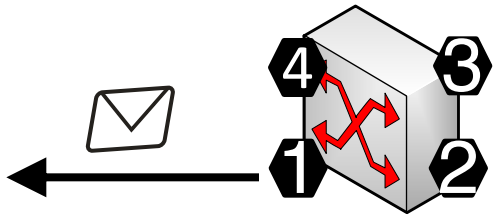
Match Field	Action

MAC table

MAC address	Ingress port
08-00-20-3A-00-4F	1
08-00-2A-0B-FE-FD	2

Controller

Host 1
MAC address
08-00-20-3A-00-4F



Switch

Host 2
MAC address
08-00-2A-0B-FE-FD

Flow Table

Match Field	Action
Src: 08-00-2A-0B-FE-FD Dst: 08-00-20-3A-00-4F	Forward on port 1
Src: 08-00-20-3A-00-4F Dst: 08-00-2A-0B-FE-FD	Forward on port 2

Lab 2: OpenFlow Learning Switch

- Warm-up
  - Construct a network using **Mininet**
  - Use **wireshark** to capture OpenFlow messages
  - Ping and use **ovs-ofctl** to check the flow tables

- Task
  - Construct the ARPANET-1969 (4 nodes) using Mininet
  - Run Ryu as the remote controller
  - Let UCLA ping UTAH
  - If not reachable, try to solve it by programming with Ryu