第一章 Openflow1.0

第1.1节 概述

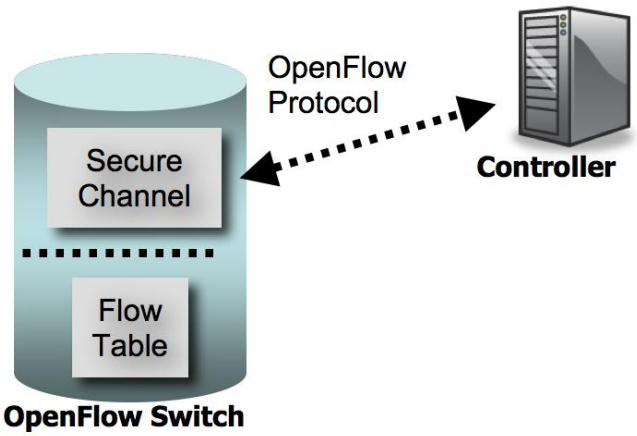
官方网站：http://OpenFlowSwitch.org。

本部分内容按照 Openflow 规范 1.0 版本撰写。1.0 之前版本都是草案，从 1.0 版本开始是正式版本，生产商们理论上应该都参照这个版本。1.0 版本的下载地址为 http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf。

目前最新规范版本为 1.3，但现有实现多以 1.0 版本为主。

第1.2节 交换机组成

每个 of 交换机（switch）都有一张流表，进行包查找和转发。交换机可以通过 of 协议经一个安全通道连接到外部控制器（controller），对流表进行查询和管理。图表 一-1 展示了这一过程。



图表 一-1 of 交换机通过安全通道连接到控制器

流表中包括一些流表项，每个表项包括若干个域：包头域（header fileds，匹配包头多个域）、活动计数器（counters）、0 个或多个执行行动（actions）。交换机对每一个包在流表中进行查找，如果匹配则执行相关行动，否则通过安全通道将包转发到控制器，控制器来决策如何处理无匹配流表的网包，并添加或者删除流表项。

流表项可以将包转发到一个或者多个端口。一般来说，可以指定物理端口，协议并没有预先规定一些抽象集合（包括端口聚合或 vlan 端口）。of 端口状态有限，包括 up、down 或是否生成树洪泛从此端口转发。端口配置可以通过 of 配置协议进行处理。of 虚拟端口包括洪泛和入口等。

第1.3节 流表

流表是交换机进行转发策略控制的核心数据结构。交换芯片通过查找流表表项来决

策对进入交换机的网络流量采取合适的行为。

每个表项包括三个域，包头域（header field），计数器（counters），行动（actions）。如表格 一-1 所示。

表格 一-1 流表项结构

Head Fileds	Counter	Actions

1.3.1 包头域

包头域包括 12 个域，如表格 一-2 所示，包括：进入接口，Ethernet 源地址、目标地址、类型，vlan id，vlan 优先级，IP 源地址、目标地址、协议、IP ToS 位，TCP/UDP 目标端口、源端口。每一个域包括一个确定值或者任意值（ANY）。如果交换机支持 ip 地址子网掩码等，可以实现更为准确的指定。

交换机设计者可以自行提供合适的实现。例如，对于一条拥有多个转发行动（每个指定不同的端口）的流，交换机可以在硬件转发表中通过一个比特掩码来实现。

表格 一-2 流表项的包头域

Ingress Port	Ether Source	Ether Dst	Ether Type	VLAN id	VLAN Priority	IP src	IP dst	IP proto	IP ToS bits	TCP/UDP Src Port	TCP/UDP Dst Port

更具体的各个域的解释参见表格 一-3。

表格 一-3 包头域详细含义

Field	Bits	When applicable	Notes
Ingress Port	(Implementation dependent)	All packets	Numerical representation of incoming port, starting at 1.
Ethernet source address	48	All packets on enabled ports	
Ethernet destination address	48	All packets on enabled ports	
Ethernet type	16	All packets on enabled ports	An OpenFlow switch is required to match the type in both standard Ethernet and 802.2 with a SNAP header and OUI of 0x000000. The special value of0x05FF is used to match all 802.3 packets without SNAP headers.
VLAN id	12	All packets of Ethernet type 0x8100	

VLAN priority	3	All packets of Ethernet type 0x8100	VLAN PCP field
IP source address	32	All IP and ARP packets	Can be subnetmasked
IP destination address	32	All IP and ARPpackets	Can be subnetmasked
IP protocol	8	All IP and IP over Ethernet, ARP packets	Only the lower 8 bits of the ARP opcode are used
IP ToS bits	6	All IP packets	Specify as 8-bitvalue and place ToS in upper 6 bits.
Transport sourceport / ICMP Type	16	All TCP, UDP, andICMP packets	Only lower 8 bitsused for ICMP Type
Transport destination port / ICMP Code	16	All TCP, UDP, andICMP packets	Only lower 8 bits used for ICMP Code

1.3.2 计数器（counter）

计数器可以针对每张表、每个流、每个端口、每个队列来分别维护。计数器可以利用软件实现，通过查询硬件计数器并实现更大的技术范围，计数器没有溢出提示。计数器用来统计流量的一些信息，例如存活时间、错误、活动表项、查找次数、发送包数等。必需的计数器在表格 一-4 中给出。

表格 一-4 统计信息需要的计数器

Counter	Bits
Per Table	
Active Entries	32
Packet Lookups	64
Packet Matches	64
Per Flow	
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32
Per Port	
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64

Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
Receive Frame Alignment Errors	64
Receive Overrun Errors	64
Receive CRC Errors	64
Collisions	64
Per Queue	
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

1.3.3 行动（action）

每个表项对应到 0 个或者多个行动，如果没有转发（forward）行动，则默认丢弃。多个行动的执行需要依照行动列表中优先级顺序依次进行。但在同一个端口上，对包的发送不保证顺序。例如，一个行动列表可能指定在同一个端口上发送两个网包到不同的 VLAN，这两个网包的实际发出顺序可能是任意的，但包的内容必须由顺序执行的行动生成。

对于无法处理的行动列表，交换机可以拒绝流表项，并立刻返回一个不支持的错误（unspported flow error）。在同一个端口上的顺序可能因为生产商实现不同而不同。

行动可以分为两种类型：必备行动（Required Actions）和可选行动（Optional Actions）。必备行动是默认支持的，交换机在连接到控制器时需要通知控制器它支持的可选行动。of 兼容的交换机根据支持行动的不同分为两类：纯 of 交换机（OpenFlow-only）和 of 使能（OpenFlow-enabled）交换机。

纯 of 交换机仅支持必备行动，而 of 使能的交换机、路由器和接入点可能还支持 NORMAL 行动。两者都可以支持 FLOOD 行动。

1.3.3.1 必备行动

必备行动-转发（Forward）

除了给定的合法交换机端口外，还支持如下的特殊端口。

- ALL 转发到所有出口（不包括入口）
- CONTROLLER 封装并转发给控制器
- LOCAL 转发给本地网络栈
- TABLE 对要发出的包执行流表中的行动（仅对 packet-out 消息）
- IN_PORT 从入口发出

必备行动-丢弃（Drop）

没有明确指明处理行动的表项，所匹配的所有网包默认丢弃。

1.3.3.2 可选行动

可选行动-转发

- NORMAL 按照传统交换机的 2 层或 3 层进行转发处理。
- FLOOD 通过最小生成树从出口泛洪发出（注意不包括入口）。

可选行动-入队（Enqueue）

将包转发到绑定到某个端口的队列中。

可选行动-修改域（Modify-field）

修改包头内容。具体的行为见表格 一-5。

表格 一-5 修改域行为

Action	Associated Data	Description
Set VLAN ID	12 bits	If no VLAN is present, a new header is added with the specified VLAN ID and priority of zero. If a VLAN header already exists, the VLAN ID is re- placed with the specified value.
Set VLAN priority	3 bits	If no VLAN is present, a new header is added with the specified priority and a VLAN ID of zero. If a VLAN header already exists, the priority field is replaced with the specified value.
Strip VLAN header	-	StripVLAN header ifpresent.
Modify Ethernet source MAC address	48 bits: Value with which to replace existing source MAC address	Replace the existing Ethernet source MAC address with the new value
Modify Ethernet destination MAC address	48 bits: Value with which to replace existing destination MAC address	Replace the existing Ethernet destination MAC ad- dress with the new value.
Modify IPv4 source address	32 bits: Value with which to replace existing IPv4 source address	Replace the existing IP source address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applica- ble to IPv4 packets.
Modify IPv4 destination address	32 bits: Value with which to replace existing IPv4 desti- nation address	Replace the existing IP destination address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applied to IPv4 packets.
Modify IPv4 ToS bits	6 bits: Value with whichto replace existing IPv4 ToS field	Replace the existing IP ToS field. This action is only ap- plied to IPv4 packets.

Modify transport source port	16 bits: Value with which to replace existing TCP or UDP source port	Replace the existing TCP/UDP source port with new value and update the TCP/UDP checksum. This action is only applica- ble to TCP and UDP pack- ets.
Modify transport destination port	16 bits: Value with which to replace existing TCP or UDP destination port	Replace the existing TCP/UDP destination port with new value and update the TCP/UDP checksum This action is only applied to TCP and UDP packets.

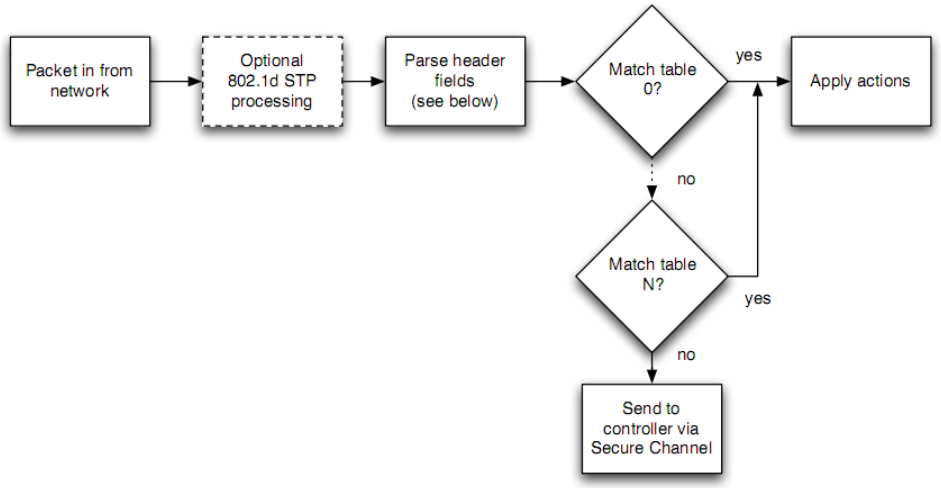
1.3.3.3 交换机类型

通过支持的行为类型不同，兼容 of 的交换机分为两类，一类是“纯 of 交换机”（of-only），一类是“支持 of 交换机”（of-enable）。前者仅需要支持必备行动，后者还可以支持 NORMAL 行动。同时，双方都可以支持泛洪行动（Flood Action）。

表格 一-6 各种类型 of 交换机的支持行动

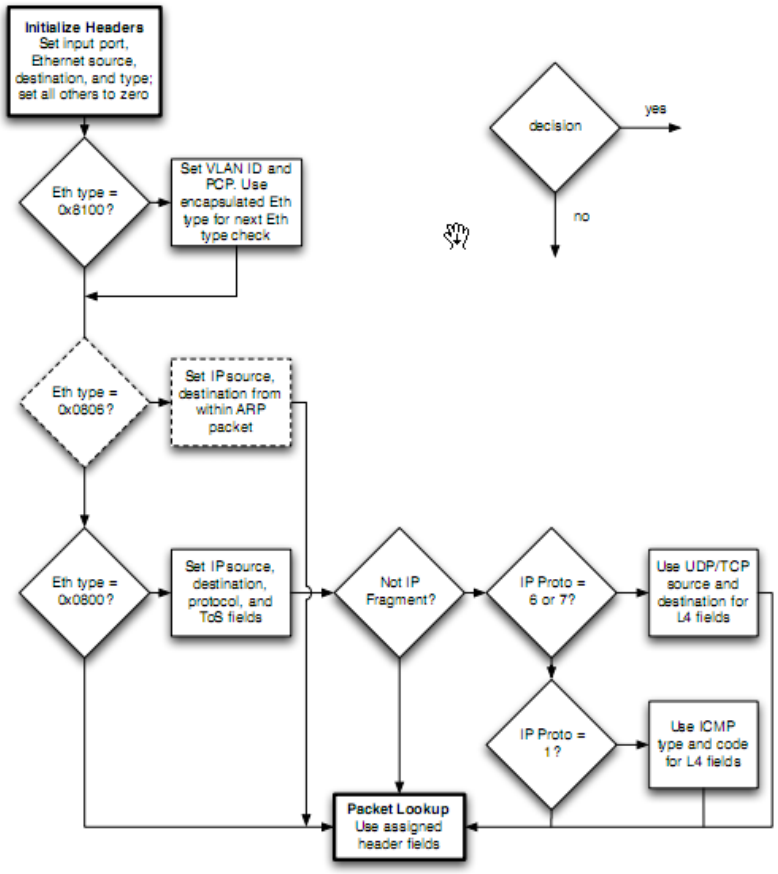
~~ACTION~~	of-only	of-enable
Required Actions	YES	YES
NORMAL	NO	CAN
FLOOD	CAN	CAN

1.3.4 匹配



图表 一-2 整体匹配流程

每个包按照优先级依次去匹配流表中表项，匹配包的优先级最高的表项即为匹配结果。一旦匹配成功，对应的计数器将更新；如果没能找到匹配的表项，则转发给控制器。整体流程参见图表 一-2，具体包头解析匹配过程见图表 一-3。



图表 一-3 包头解析的匹配流程

第1.4节 安全通道

安全通道用来连接交换机和控制器，所有安全通道必须遵守 of 协议。控制器可以配置、管理交换机、接收交换机的事件信息，并通过交换机发出网包等。这一部分是交换机和控制器实现互动的基础。

1.4.1 of 协议

of 协议支持三种消息类型：controller-to-switch，asynchronous（异步）和 symmetric（对称），每一类消息又有多个子消息类型。controller-to-switch 消息由控制器发起，用来管理或获取 switch 状态；asynchronous 消息由 switch 发起，用来将网络事件或交换机状态变化更新到控制器；symmetric 消息可由交换机或控制器发起。

1.4.1.1 controller-to-switch 消息

由控制器（controller）发起，可能需要或不需要来自交换机的应答消息。包括 Features、Configuration、Modify-state、Read-state、Send-packet、Barrier 等。

Features

在建立传输层安全会话（Transport Layer Security Session）的时候，控制器发送 feature 请求消息给交换机，交换机需要应答自身支持的功能。

Configuration

控制器设置或查询交换机上的配置信息。交换机仅需要应答查询消息。

Modify-state

控制器管理交换机流表项和端口状态等。

Read-state

控制器向交换机请求一些诸如流、网包等统计信息。

Send-packet

控制器通过交换机指定端口发出网包。

Barrier

控制器用以确保消息依赖满足，或接收完成操作的通知。

1.4.1.2 asynchronous 消息

asynchronous 消息不需要控制器请求发起，主要用于交换机向控制器通知状态变化等事件信息。主要消息包括 Packet-in、Flow-removed、Port-status、Error 等。

Packet-in

交换机收到一个网包，在流表中没有匹配项，则发送 Packet-in 消息给控制器。如果交换机缓存足够多，网包被临时放在缓存中，网包的部分内容（默认 128 字节）和在交换机缓存中的的序号也一同发给控制器；如果交换机缓存不足以存储网包，则将整个网包作为消息的附带内容发给控制器。

Flow-removed

交换机中的流表项因为超时或修改等原因被删除掉，会触发 Flow-removed 消息。

Port-status

交换机端口状态发生变化时（例如 down 掉），触发 Port-status 消息。

Error

交换机通过 Error 消息来通知控制器发生的问题。

1.4.1.3 symmetric 消息

symmetric 消息也不必通过请求建立，包括 Hello、Echo、Vendor 等。

Hello

交换机和控制器用来建立连接。

Echo

交换机和控制器均可以向对方发出 Echo 消息，接收者则需要回复 Echo reply。该消息用来测量延迟、是否连接保持等。

Vendor

交换机提供额外的附加信息功能。为未来版本预留。

1.4.2 连接建立

通过安全通道建立连接，所有相关流量都不经过交换机流表检查。因此交换机必须将安全通道认为是本地连接。今后版本中将介绍动态发现控制器的协议。

当 of 连接建立起来后，两边必须先发送 OFPT_HELLO 消息给对方，该消息携带支持的最高协议版本号，接受方将采用双方都支持的最低协议版本进行通信。经过协商，一旦发现双方共同支持的协议版本，则连接建立；否则发送 OFPT_ERROR 消息（类型为 OFPET_HELLO_FAILED，代码为 OFPHFC_COMPATIBLE），描述失败原因，并终止连接。

1.4.3 连接中断

当连接发生异常时，交换机应尝试连接备份的控制器。当多次尝试均失败后，交换机将进入紧急模式，并重置所有的 TCP 连接。此时，所有包将匹配指定的紧急模式表项，其他所有正常表项将从流表中删除。此外，当交换机刚启动时，默认进入紧急模式。

1.4.4 加密

安全通道采用 TLS（Transport Layer Security）连接加密。当交换机启动时，尝试连接到控制器的 6633 TCP 端口。双方通过交换证书进行认证。因此，每个交换机至少需配置两个证书，一个是用来认证控制器，一个用来向控制器发出认证。

1.4.5 生成树

交换机可以选择支持 802.1D 生成树协议。如果支持，所有相关包在查找流表之前应该先在本地进行传统处理。支持生成树协议的交换机在 OFPT_FEATURES_REPLY 消息的 compabilities 域需要设置 OFPC_STP 位，并且需要在所有的物理端口均支持生成树协议，但无需在虚拟端口支持。

生成树协议会设置端口状态，来限制发到 OFP_FLOOD 的网包仅被转发到生成树指定的端口。需要注意指定出口的转发或 OFP_ALL 的网包会忽略生成树指定的端口状态，按照规则设置端口转发。

如果交换机不支持 802.1D 生成树协议，则必须允许控制器指定泛洪时的端口状态。

1.4.6 流表修改

流表修改消息可以有以下类型：

```
enum ofp_flow_mod_command {
OFPFC_ADD, /* New flow. */
OFPFC_MODIFY, /* Modify all matching flows. */
OFPFC_MODIFY_STRICT, /* Modify entry strictly matching wildcards */
OFPFC_DELETE, /* Delete all matching flows. */
OFPFC_DELETE_STRICT /* Strictly match wildcards and priority. */
};
```

1.4.6.1 ADD

对于带有 OFPFF_CHECK_OVERLAP 标志的添加（ADD）消息，交换机将先检查新表项是否跟现有表项冲突（包头范围 overlap，且有相同的优先级），如果发现冲突，将拒绝添加，并返回 ofp_error_msg，并且指明 OFPET_FLOW_MOD_FAILED 类型和 OFPFMFC_OVERLAP 代码。

对于合法无冲突的添加，或不带 OFPFF_CHECK_OVERLAP 标志的添加，新表项将被添加到最低编号表中，优先级在匹配过程中获取。如果任何表中已经存在与新表项相同头部域和优先级的旧表项，该项将被新表项替代，同时计数器清零。如果交换机无法找到要添加的表，则返回 ofp_error_msg，并且指明 OFPET_FLOW_MOD_FAILED 类型和 OFPFMFC_ALL_TABLES_FULL 代码。

如果添加表项使用了交换机不合法的端口，则交换机返回 ofp_error_msg 消息，同时带有 OFPET_BAD_ACTION 类型和 OFPBAC_BAD_OUT_PORT 代码。

1.4.6.2 MODIFY

对于修改，如果所有已有表中没有与要修改表项同样头部域的表项，则等同于 ADD 消息，计数器置 0；否则更新现有表项的行为域，同时保持计数器、空闲时间均不变。

1.4.6.3 DELETE

对于删除，如果没有找到要删除表项，不发出任何消息；如果存在，则进行删除操作。如果被删除的表项带有 OFPFF_SEND_FLOW_REM 标志，则触发一条流删除的消息。删除紧急表项不触发消息。

此外，修改和删除还存在另一个_STRICT 版本。对于非_STRICT 版本，通配流表项是激活的，因此，所有匹配消息描述的流表项均受影响（包括包头范围被包含在消息表项中的流表项）。例如，一条所有域都是通配符的非_STRICT 版本删除消息会清空流表，因为所有表项均包含在该表项中。

在_STRICT 版本情况下，表项头跟优先级等都必须严格匹配才执行，即只有同一条表项会受影响。例如，一条所有域都是通配符的 DELETE_STRICT 消息仅删除指定优先级的某条规则。

此外删除消息还支持指定额外的 out_port 域。

如果交换机不能处理流操作消息指定的行为，则返回 OFPET_FLOW_MOD_FAILED ： OFPFMFC_UNSUPPORTED，并拒绝该表项。

1.4.7 流超时

每个表项均有一个 idle_timeout 和一个 hard_timeout，前者计算没有流量匹配的时间（单位都是秒），后者计算被插入表中的时间。一旦到达时间期限，则交换机自动删除该表项，同时发出一个流删除的消息。

第1.5节 of 协议

包括 of 协议相关消息的数据结构等。

1.5.1 of 协议头

数据结构如下。

```
/* Header on all OpenFlow packets. */
struct ofp_header {
uint8_t version; /* OFP_VERSION. */
uint8_t type; /* One of the OFPT_ constants. */
uint16_t length; /* Length including this ofp_header. */
uint32_t xid;
/* Transaction id associated with this packet.
Replies use the same id as was in the request
to facilitate pairing. */
};
OFP_ASSERT(sizeof(struct ofp_header) == 8);
```

Version 用来标明 of 协议版本。当前 of 协议中，version 最重要的位用来标明实验版本，其他位标明修订版本。目前的版本是 0x01，最终的 Type　0 交换机应该是 0x00。

Length 用来标明消息长度。

Type 用来标明消息类型。可能的消息类型包括

```
enum ofp_type {
/* Immutable messages. */
OFPT_HELLO,  /* Symmetric message */
OFPT_ERROR,        /* Symmetric message */
OFPT_ECHO_REQUEST,  /* Symmetric message */
OFPT_ECHO_REPLY,       /* Symmetric message */
OFPT_VENDOR,      /* Symmetric message */

/* Switch configuration messages. */ OFPT_FEATURES_REQUEST,      /*
Controller/switch message */ OFPT_FEATURES_REPLY,       /*           Controller/switch
message  */  OFPT_GET_CONFIG_REQUEST,       /*  Controller/switch  message  */
OFPT_GET_CONFIG_REPLY,   /* Controller/switch message */ OFPT_SET_CONFIG,   /*
Controller/switch message */

/* Asynchronous messages. */
OFPT_PACKET_IN,  /* Async message */
OFPT_FLOW_REMOVED,       /* Async message */
OFPT_PORT_STATUS,     /* Async message */

/* Controller command messages. */
OFPT_PACKET_OUT,       /* Controller/switch message */
OFPT_FLOW_MOD,       /* Controller/switch message */
OFPT_PORT_MOD,       /* Controller/switch message */

/* Statistics messages. */
OFPT_STATS_REQUEST,  /* Controller/switch message */
OFPT_STATS_REPLY,       /* Controller/switch message */

/* Barrier messages. */
OFPT_BARRIER_REQUEST,     /* Controller/switch message */
OFPT_BARRIER_REPLY,   /* Controller/switch message */

/*  Queue  Configuration  messages.  */  OFPT_QUEUE_GET_CONFIG_REQUEST,    /*
Controller/switch message */ OFPT_QUEUE_GET_CONFIG_REPLY      /*
Controller/switch message */
}
```

1.5.2 常用数据结构

包括端口、队列、匹配、行动等。

1.5.2.1 端口

1.5.2.1.1 端口结构

```
/* Description of a physical port */
struct ofp_phy_port {
uint16_t port_no;
uint8_t hw_addr[OFP_ETH_ALEN];
char name[OFP_MAX_PORT_NAME_LEN]; /* Null-terminated */

uint32_t config;     /* Bitmap of OFPPC_* flags. */
uint32_t state;      /* Bitmap of OFPPS_* flags. */

/* Bitmaps of OFPPF_* that describe features.  All bits zeroed if
* unsupported or unavailable. */
uint32_t curr; /* Current features. */
uint32_t advertised;     /* Features being advertised by the port. */
uint32_t supported;      /* Features supported by the port. */
uint32_t peer;       /* Features advertised by peer. */
};
OFP_ASSERT(sizeof(struct ofp_phy_port) == 48);
```

port_no 标明绑定到物理接口的 datapath 值。
hw_addr 是该物理接口的 mac 地址。
OFP_MAX_ETH_ALEN 值为 6。
name 是该接口的名称字符串，以 null 结尾。
OFP_MAX_PORT_NAME_LEN 为 16。

1.5.2.1.2 Config

config 描述了生成树和管理设置，数据结构为

```
/* Flags to indicate behavior of the physical port.  These flags are
* used in ofp_phy_port to describe the current configuration.  They are
* used in the ofp_port_mod message to configure the port's behavior.
*/
enum ofp_port_config {
OFPPC_PORT_DOWN    = 1 << 0,  /* Port is administratively down. */

OFPPC_NO_STP   = 1 << 1,  /* Disable 802.1D spanning tree on port. */
OFPPC_NO_RECV  = 1 << 2,  /* Drop all packets except 802.1D spanning
tree packets. */
OFPPC_NO_RECV_STP  = 1 << 3,  /* Drop received 802.1D STP packets. */
OFPPC_NO_FLOOD     = 1 << 4,  /* Do not include this port when flooding. */
OFPPC_NO_FWD   = 1 << 5,  /* Drop packets forwarded to port. */
OFPPC_NO_PACKET_IN = 1 << 6   /* Do not send packet-in msgs for port. */
};
```

1.5.2.1.3 State

state 描述生成树状态和某个物理接口是否存在，数据结构为

```
/* Current state of the physical port.  These are not configurable from
* the controller.
*/
enum ofp_port_state {
OFPPS_LINK_DOWN    = 1 << 0, /* No physical link present. */

/* The OFPPS_STP_* bits have no effect on switch operation.  The
* controller must adjust OFPPC_NO_RECV, OFPPC_NO_FWD, and
* OFPPC_NO_PACKET_IN appropriately to fully implement an 802.1D spanning
* tree. */
OFPPS_STP_LISTEN   = 0 << 8, /* Not learning or relaying frames. */
OFPPS_STP_LEARN    = 1 << 8, /* Learning but not relaying frames. */
OFPPS_STP_FORWARD  = 2 << 8, /* Learning and relaying frames. */
OFPPS_STP_BLOCK    = 3 << 8, /* Not part of spanning tree. */
OFPPS_STP_MASK  = 3 << 8   /* Bit mask for OFPPS_STP_* values. */
};
```

1.5.2.1.4 端口号

一些特定的端口号定义如下（物理端口从 1 开始编号）。

```
/* Port numbering.    Physical ports are numbered starting from 1. */
enum ofp_port {
/* Maximum number of physical switch ports. */
OFPP_MAX = 0xff00,

/* Fake output "ports". */
OFPP_IN_PORT      = 0xfff8,    /* Send the packet out the input port.    This virtual port
must be explicitly usedin order to send back out of the input port. */

OFPP_TABLE    = 0xfff9,    /* Perform actions in flow table.
NB: This can only be the destination
port for packet-out messages. */
OFPP_NORMAL      = 0xfffa,    /* Process with normal L2/L3 switching. */
OFPP_FLOOD        = 0xfffb,    /* All physical ports except input port and
those disabled by STP. */
OFPP_ALL       = 0xfffc,    /* All physical ports except input port. */
OFPP_CONTROLLER = 0xfffd,    /* Send to controller. */
OFPP_LOCAL   = 0xfffe,    /* Local openflow "port". */
OFPP_NONE    = 0xffff     /* Not associated with a physical port. */
};
```

curr，advertised，supported 和 peer 域 标明 链路模式（10M 到 10G， 全双工、半双工），链路类型（铜线/光线）和链路特性 (自动协商和暂停)。链路特性（Port fe atures）数据结构如下，多个标识可以同时设置。

```
/* Features of physical ports available in a datapath. */
enum ofp_port_features {
OFPPF_10MB_HD   = 1 << 0,    /* 10 Mb half-duplex rate support. */
OFPPF_10MB_FD   = 1 << 1,    /* 10 Mb full-duplex rate support. */
OFPPF_100MB_HD = 1 << 2,    /* 100 Mb half-duplex rate support. */
OFPPF_100MB_FD = 1 << 3,    /* 100 Mb full-duplex rate support. */
OFPPF_1GB_HD    = 1 << 4,    /* 1 Gb half-duplex rate support. */
OFPPF_1GB_FD    = 1 << 5,    /* 1 Gb full-duplex rate support. */
OFPPF_10GB_FD   = 1 << 6,    /* 10 Gb full-duplex rate support. */
OFPPF_COPPER    = 1 << 7,    /* Copper medium. */
OFPPF_FIBER  = 1 << 8,    /* Fiber medium. */
OFPPF_AUTONEG   = 1 << 9,    /* Auto-negotiation.  */
OFPPF_PAUSE = 1 << 10, /* Pause. */
OFPPF_PAUSE_ASYM  = 1 << 11    /* Asymmetric pause. */
};
```

1.5.2.2 队列

1.5.2.2.1 队列结构

队列被绑定到某个端口，实现有限的流量控制操作。
数据结构为

```
/* Full description for a queue. */
struct ofp_packet_queue {
uint32_t queue_id;      /* id for the specific queue. */
uint16_t len;  /* Length in bytes of this queue desc. */
uint8_t pad[2];     /* 64-bit alignment. */
struct ofp_queue_prop_header    properties[0];  /* List of properties. */
};
OFP_ASSERT(sizeof(struct    ofp_packet_queue)  == 8);
```

1.5.2.2.2 队列特性

每一个队列都带有一些特性，数据结构为

```
enum ofp_queue_properties {
OFPQT_NONE = 0, /* No property defined for queue (default). */
OFPQT_MIN_RATE, /* Minimum datarate guaranteed. */
/* Other types should be added here
* (i.e. max rate, precedence, etc). */
};
```

特性头的数据结构为

```
/* Common description for a queue. */
struct ofp_queue_prop_header {
uint16_t property; /* One of OFPQT_. */
uint16_t len; /* Length of property, including this header. */
uint8_t pad[4]; /* 64-bit alignemnt. */
};
OFP_ASSERT(sizeof(struct ofp_queue_prop_header) == 8);
```

目前，实现了最小速率队列，数据结构为

```
/* Min-Rate queue property description. */
struct ofp_queue_prop_min_rate {
struct ofp_queue_prop_header prop_header; /* prop: OFPQT_MIN, len: 16. */
uint16_t rate; /* In 1/10 of a percent; >1000 -> disabled. */
uint8_t pad[6]; /* 64-bit alignment */
};
OFP_ASSERT(sizeof(struct ofp_queue_prop_min_rate) == 16);
```

1.5.2.3 流匹配

1.5.2.3.1 流表项结构

描述一个流表项的数据结构为

```
/* Fields to match against flows */
struct ofp_match {
uint32_t wildcards;      /* Wildcard fields. */
uint16_t in_port;  /* Input switch port. */
uint8_t dl_src[OFP_ETH_ALEN];   /* Ethernet source address. */
uint8_t dl_dst[OFP_ETH_ALEN];   /* Ethernet destination address. */
uint16_t dl_vlan;  /* Input VLAN id. */
uint8_t dl_vlan_pcp;    /* Input VLAN priority. */
uint8_t pad1[1];   /* Align to 64-bits */
uint16_t dl_type;  /* Ethernet frame type. */
uint8_t nw_tos;    /* IP ToS (actually DSCP field, 6 bits). */
uint8_t nw_proto; /* IP protocol or lower 8 bits of
* ARP opcode. */
uint8_t pad2[2];   /* Align to 64-bits */
uint32_t nw_src;   /* IP source address. */
uint32_t nw_dst;   /* IP destination address. */
uint16_t tp_src;   /* TCP/UDP source port. */
uint16_t tp_dst;   /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct    ofp_match) == 40);
```

1.5.2.3.2 wildcards

wildcards 可以设置为如下的一些标志。

```
/* Flow wildcards. */
enum ofp_flow_wildcards {
OFPFW_IN_PORT = 1 << 0, /* Switch input port. */ OFPFW_DL_VLAN = 1 <<
1, /* VLAN id. */
OFPFW_DL_SRC = 1 << 2, /* Ethernet source address. */ OFPFW_DL_DST = 1
<< 3, /* Ethernet destination address. */ OFPFW_DL_TYPE = 1 << 4, /*
Ethernet frame type. */ OFPFW_NW_PROTO = 1 << 5, /* IP protocol. */
OFPFW_TP_SRC = 1 << 6, /* TCP/UDP source port. */ OFPFW_TP_DST = 1 <<
7, /* TCP/UDP destination port. */

/* IP source address wildcard bit count. 0 is exact match, 1 ignores the
* LSB, 2 ignores the 2 least-significant bits, ..., 32 and higher wildcard
* the entire field. This is the *opposite* of the usual convention where
* e.g. /24 indicates that 8 bits (not 24 bits) are wildcarded. */
OFPFW_NW_SRC_SHIFT = 8,
OFPFW_NW_SRC_BITS = 6,
OFPFW_NW_SRC_MASK = ((1 << OFPFW_NW_SRC_BITS) - 1) <<
OFPFW_NW_SRC_SHIFT,
OFPFW_NW_SRC_ALL = 32 << OFPFW_NW_SRC_SHIFT,

/* IP destination address wildcard bit count. Same format as source. */
OFPFW_NW_DST_SHIFT = 14,
OFPFW_NW_DST_BITS = 6,
OFPFW_NW_DST_MASK = ((1 << OFPFW_NW_DST_BITS) - 1) <<
OFPFW_NW_DST_SHIFT,
OFPFW_NW_DST_ALL = 32 << OFPFW_NW_DST_SHIFT,

OFPFW_DL_VLAN_PCP = 1 << 20, /* VLAN priority. */ OFPFW_NW_TOS = 1 << 21,
/* IP ToS (DSCP field, 6 bits). */

/* Wildcard all fields. */ OFPFW_ALL = ((1 << 22) - 1)
};
```

如果 wildcards 没有设置，则 ofp_match 精确的标明流表项。注意源和目的的掩码在 wildcards 中用 6 位来表示。其含义与 CIDR 相反，表示低位的多少位被掩掉。例如在 CIDR 中掩码 24 意味着 255.255.255.0，而在 of 中，24 意味着 255.0.0.0。

1.5.2.4 行动

1.5.2.4.1 行动类型

目前，行动类型包括

```
enum ofp_action_type {
OFPAT_OUTPUT,    /* Output to switch port. */
OFPAT_SET_VLAN_VID,  /* Set the 802.1q VLAN id. */
OFPAT_SET_VLAN_PCP,  /* Set the 802.1q priority. */
OFPAT_STRIP_VLAN,    /* Strip the 802.1q header. */
OFPAT_SET_DL_SRC,    /* Ethernet source address. */
OFPAT_SET_DL_DST,    /* Ethernet destination address. */ OFPAT_SET_NW_SRC,
    /* IP source address. */ OFPAT_SET_NW_DST,  /* IP destination address. */
OFPAT_SET_NW_TOS,  /* IP ToS (DSCP field, 6 bits). */ OFPAT_SET_TP_SRC,      /*
TCP/UDP source port. */ OFPAT_SET_TP_DST,  /* TCP/UDP destination port. */
OFPAT_ENQUEUE,  /* Output to queue.    */
OFPAT_VENDOR  = 0xffff
};
```

1.5.2.4.2 行动结构

一个行动应该包括类型、长度和相应的数据。

```
/* Action header that is common to all actions.    The length includes the
* header and any padding used to make the action 64-bit aligned.
* NB: The length of an action *must* always be a multiple of eight. */
struct ofp_action_header  {
uint16_t type;/* One of OFPAT_*. */
uint16_t len;  /* Length of action, including this
header.    This is the length of action,
including any padding to make it
64-bit aligned. */
uint8_t pad[4];
};
OFP_ASSERT(sizeof(struct    ofp_action_header)  == 8);
```

1.5.2.4.3 输出行动

包括如下数据结构

```
/* Action structure for OFPAT_OUTPUT, which sends packets out ’port’.
* When the ’port’ is the OFPP_CONTROLLER, ’max_len’ indicates the max
* number of bytes to send. A ’max_len’ of zero means no bytes of the
* packet should be sent.*/
struct ofp_action_output {
uint16_t type;/* OFPAT_OUTPUT. */
uint16_t len; /* Length is 8. */
uint16_t port;/* Output port. */
uint16_t max_len; /* Max length to send to controller. */
};
OFP_ASSERT(sizeof(struct ofp_action_output) == 8);
```

其中，max_len 指明当端口为 OFPP_CONTROLLER 时，发送到控制器的包内容大小。如果为 0，则发送一个长度为 0 的 packet_in 消息。port 指明发出的物理端口。

1.5.2.4.4 入队操作

入队（Enqueue）操作将流绑定到某个已配置好的队列中，而不管 TOS 和 VLAN PCP 位。包在入队操作后不应该被修改。如有必要，原始信息可以先备份出来，在网包发出的时候还原信息。如果交换机需要通过 TOS 和 VLAN PCP 位来进行队列操作，则用户需要指定这些位来让流绑定到合适的队列上。

入队操作包括如下的域

```
/* OFPAT_ENQUEUE action struct: send packets to given queue on port. */
struct ofp_action_enqueue {
uint16_t type;/* OFPAT_ENQUEUE. */
uint16_t len; /* Len is 16. */
uint16_t port;/* Port that queue belongs. Should
refer to a valid physical port
(i.e. < OFPP_MAX) or OFPP_IN_PORT. */
uint8_t pad[6]; /* Pad for 64-bit alignment. */
uint32_t queue_id; /* Where to enqueue the packets. */
};
OFP_ASSERT(sizeof(struct ofp_action_enqueue) == 16);
```

action_vlan_vid 包括如下的域

```
/* Action structure for OFPAT_SET_VLAN_VID.    */
struct ofp_action_vlan_vid    {
uint16_t type;        /* OFPAT_SET_VLAN_VID.    */
uint16_t len;   /* Length is 8. */
uint16_t vlan_vid;  /* VLAN id. */
uint8_t pad[2];
};
OFP_ASSERT(sizeof(struct    ofp_action_vlan_vid)    == 8);
```

vlan_vid field 有 16 位，而实际的 VLAN id 仅有 12 位。0xffff 用来标明没有设置 VLAN id。

```
An action_vlan_pcp has the following fields:

/* Action structure for OFPAT_SET_VLAN_PCP.    */
struct ofp_action_vlan_pcp    {
uint16_t type;        /* OFPAT_SET_VLAN_PCP.    */
uint16_t len;   /* Length is 8. */
uint8_t vlan_pcp;   /* VLAN priority. */
uint8_t pad[3];
};
OFP_ASSERT(sizeof(struct    ofp_action_vlan_pcp)    == 8);
```

vlan_pcp field 有 8 位长，但仅有低 3 位有意义。

action_strip_vlan 没有参数，仅包括一个通用的 ofp_action_header. 该行为会剥掉 VLAN tag（如果存在）。

ction_dl_addr 包括如下的域

```
/* Action structure for OFPAT_SET_DL_SRC/DST.    */
struct ofp_action_dl_addr {
uint16_t type;        /* OFPAT_SET_DL_SRC/DST.    */
uint16_t len;   /* Length is 16. */
uint8_t dl_addr[OFP_ETH_ALEN];     /* Ethernet address. */
uint8_t pad[6];
};
OFP_ASSERT(sizeof(struct    ofp_action_dl_addr)    == 16);
```

其中，dl_addr field 是要设置的 mac 地址。

action_nw_addr 包括如下的域

```
/* Action structure for OFPAT_SET_NW_SRC/DST.    */
struct ofp_action_nw_addr {
uint16_t type;        /* OFPAT_SET_TW_SRC/DST.    */
uint16_t len;   /* Length is 8. */
uint32_t nw_addr;         /* IP address. */
};
OFP_ASSERT(sizeof(struct    ofp_action_nw_addr)    == 8);
```

其中，nw_addr field 是要设置的 IP 地址。
action_nw_tos 包括如下的域

```
/* Action structure for OFPAT_SET_NW_TOS. */
struct ofp_action_nw_tos {
uint16_t type;        /* OFPAT_SET_TW_SRC/DST.    */
uint16_t len;   /* Length is 8. */
uint8_t nw_tos;      /* IP ToS (DSCP field, 6 bits). */
uint8_t pad[3];
};
OFP_ASSERT(sizeof(struct    ofp_action_nw_tos) == 8);
```

nw_tos 域是要设置的 ToS 的高 6 位。
action_tp_port 包括如下的域

```
/* Action structure for OFPAT_SET_TP_SRC/DST.    */
struct ofp_action_tp_port {
uint16_t type;        /* OFPAT_SET_TP_SRC/DST.    */
uint16_t len;   /* Length is 8. */
uint16_t tp_port;    /* TCP/UDP port. */
uint8_t pad[2];
};
OFP_ASSERT(sizeof(struct    ofp_action_tp_port)    == 8);
```

tp_port 域是要设置的 TCP/UDP/其它端口。

```
An action_vendor has the following fields:
/* Action header for OFPAT_VENDOR. The rest of the body is vendor-defined. */
struct ofp_action_vendor_header    {
uint16_t type;        /* OFPAT_VENDOR. */
uint16_t len;   /* Length is a multiple of 8. */
uint32_t vendor;    /* Vendor ID, which takes the same form
as in "struct ofp_vendor_header".    */
};
OFP_ASSERT(sizeof(struct    ofp_action_vendor_header) == 8);
```

vendor 域是 Vendor 的 ID，跟结构 ofp_vendor 中一致。

1.5.3 Controller-to-Switch 消息

包括握手、配置交换机、修改状态、配置队列、读取状态、发包、保障等。

1.5.3.1 握手

TLS 连接建立之初，控制器发送一个仅有消息头的 OFPT_FEATURES_REQUEST 消息，交换机返回一个 OFPT_FEATURES_REPLY 消息。

```
/* Switch features. */
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 pad[3]; /* Align to 64-bits. */

/* Features. */
uint32_t capabilities; /* Bitmap of support "ofp_capabilities". */
uint32_t actions; /* Bitmap of supported "ofp_action_type"s. */

/* Port info.*/
struct ofp_phy_port ports[0]; /* Port definitions. The number of ports
is inferred from the length field in
the header. */
};
OFP_ASSERT(sizeof(struct ofp_switch_features) == 32);
```

其中，datapath_id 是 datapath 的唯一标识，其中低 48 位设计为交换机 mac 地址，高 16 位由实现者来定义。

n_tables 域指明交换机支持的流表个数。

capabilities 域使用下面的标志。

```
/* Capabilities supported by the datapath. */
enum ofp_capabilities {
OFPC_FLOW_STATS= 1 << 0, /* Flow statistics. */
OFPC_TABLE_STATS= 1 << 1, /* Table statistics. */
OFPC_PORT_STATS = 1 << 2, /* Port statistics. */
OFPC_STP = 1 << 3, /* 802.1d spanning tree. */
OFPC_RESERVED = 1 << 4, /* Reserved, must be zero. */
OFPC_IP_REASM = 1 << 5, /* Can reassemble IP fragments. */
OFPC_QUEUE_STATS = 1 << 6, /* Queue statistics. */
OFPC_ARP_MATCH_IP = 1 << 7 /* Match IP addresses in ARP pkts. */
};
```

actions 是用来标志支持行动的 bit 串。

ports 是一个 ofp_phy_port 结构数组，来描绘所有支持 of 的交换机端口。

1.5.3.2 配置交换机

控制器通过向交换机发送 OFPT_SET_CONFIG 和 OFPT_GET_CONFIG_REQUEST 消息（该消息仅有头部）来配置和查询交换机配置。对于查询消息，交换机必须回复 OFPT_GET_CONFIG_REPLY 消息。

配置和查询消息结构如下

```
/* Switch configuration. */
struct ofp_switch_config {
struct ofp_header header;
uint16_t flags;      /* OFPC_* flags. */
uint16_t miss_send_len;      /* Max bytes of new flow that datapath should
send to the controller. */
};
OFP_ASSERT(sizeof(struct    ofp_switch_config)  == 12);
```

配置标志包括

```
enum ofp_config_flags {
/* Handling of IP fragments. */
OFPC_FRAG_NORMAL   = 0, /* No special handling for fragments. */
OFPC_FRAG_DROP = 1, /* Drop fragments. */
OFPC_FRAG_REASM      = 2, /* Reassemble (only if OFPC_IP_REASM set). */
OFPC_FRAG_MASK = 3
}
```

OFPC_FRAG_*标志用来指示该如何处理 IP 碎片：正常、丢弃还是重组。

miss_send_len 用来指示当网包在交换机中不匹配或者匹配结果是发到控制器的时候，该发送多少数据给控制器。如果为 0，则发送长度为 0 的 packet_in 消息到控制器。

1.5.3.3 修改状态

控制器修改流表需要通过 OFPT_FLOW_MOD 消息。

```
/* Flow setup and teardown (controller -> datapath). */
struct ofp_flow_mod {
struct ofp_header header;
struct ofp_match match;       /* Fields to match */
uint64_t cookie;     /* Opaque controller-issued identifier. */

/* Flow actions. */
uint16_t command;       /* One of OFPFC_*. */
uint16_t idle_timeout;   /* Idle time before discarding (seconds). */
uint16_t hard_timeout;        /* Max time before discarding (seconds). */
uint16_t priority;    /* Priority level of flow entry. */
uint32_t buffer_id;        /* Buffered packet to apply to (or -1).
Not meaningful for OFPFC_DELETE*. */
uint16_t out_port;        /* For OFPFC_DELETE* commands, require
matching entries to include this as an
output port.    A value of OFPP_NONE
indicates no restriction. */
uint16_t flags;        /* One of OFPFF_*. */
struct ofp_action_header actions[0]; /* The action length is inferred
from the length field in the
header. */
};
OFP_ASSERT(sizeof(struct    ofp_flow_mod) == 72);
```

command 域必须为下面的类型之一。

```
enum ofp_flow_mod_command    {

      OFPFC_ADD,   /* New flow. */
      OFPFC_MODIFY,      /* Modify all matching flows. */
      OFPFC_MODIFY_STRICT, /* Modify entry strictly matching wildcards */
      OFPFC_DELETE,      /* Delete all matching flows. */
      OFPFC_DELETE_STRICT   /* Strictly match wildcards and priority. */
};
```

priority 仅跟设置了通配符的域相关。priority 域指明了流表的优先级，数字越大，则优先级越高。因此，高优先级的带通配符流表项必须要被放到低编号的流表中。交换机负责正确的顺序，避免高优先级规则覆盖掉低优先级的。

buffer_id 标志被 OFPT_PACKET_IN 消息发出的网包在 buffer 中的 id。

out_port 可选的用于进行删除操作时的匹配。

flags 域可能包括如下的标志。

```
enum ofp_flow_mod_flags {
OFPFF_SEND_FLOW_REM = 1 << 0, /* Send flow removed message when flow
* expires or is deleted. */
OFPFF_CHECK_OVERLAP = 1 << 1, /* Check for overlapping entries first. */
OFPFF_EMERG = 1 << 2 /* Remark this is for emergency. */
};
```

当 OFPFF_SEND_FLOW_REM 被设置的时候，表项超时删除会触发一条表项删除的信息。

当 OFPFF_CHECK_OVERLAP 被设置的时候，交换机必须检查同优先级的表项之间是否有匹配范围的冲突。

当 OFPFF_EMERG_被设置的时候，交换机将表项当作紧急表项，只有当与控制器连接断开的时候才启用。

控制器使用 OFPT_PORT_MOD 消息来修改交换机物理端口的行为。

```
/* Modify behavior of the physical port */
struct ofp_port_mod {
struct ofp_header header;
uint16_t port_no;
uint8_t hw_addr[OFP_ETH_ALEN]; /* The hardware address is not
configurable. This is used to
sanity-check the request, so it must
be the same as returned in an
ofp_phy_port struct. */

uint32_t config; /* Bitmap of OFPPC_* flags. */
uint32_t mask; /* Bitmap of OFPPC_* flags to be changed. */

uint32_t advertise; /* Bitmap of "ofp_port_features"s. Zero all bits to prevent
any action taking place. */
uint8_t pad[4]; /* Pad to 64-bits. */
};
OFP_ASSERT(sizeof(struct ofp_port_mod) == 32);
```

mask 用来选择 config 中被修改的域。advertise 域没有掩码，所有的特性一起修改。

1.5.3.4 配置队列

队列的配置不在 of 协议考虑内。可以通过命令行或者其他协议来配置。

控制器可以利用下面的结构来查询某个端口已经配置好的队列信息。

```
/* Query for port queue configuration. */
struct ofp_queue_get_config_request {
struct ofp_header header;
uint16_t port;        /* Port to be queried. Should refer
to a valid physical port (i.e. < OFPP_MAX) */
uint8_t pad[2];       /* 32-bit alignment. */
};
OFP_ASSERT(sizeof(struct    ofp_queue_get_config_request) == 12);
```

交换机回复一个 ofp_queue_get_config_reply 命令，其中包括配置好队列的一个列表。

```
/* Queue configuration  for a given port. */
struct  ofp_queue_get_config_reply {
struct  ofp_header  header;
uint16_t  port;
uint8_t  pad[6];
struct  ofp_packet_queue  queues[0];  /* List of configured  queues. */
};
OFP_ASSERT(sizeof(struct     ofp_queue_get_config_reply) == 16);
```

1.5.3.5 获取状态

1.5.3.5.1 基本结构

系统运行时，datapath 可以通过 OFPT_STATS_REQUEST 消息来查询当前状态。

```
struct ofp_stats_request {
struct ofp_header header;
uint16_t type;        /* One of the OFPST_* constants. */
uint16_t flags;       /* OFPSF_REQ_* flags (none yet defined). */
uint8_t body[0];      /* Body of the request. */
};
OFP_ASSERT(sizeof(struct    ofp_stats_request) == 12);
```

交换机则回复一条或多条 OFPT_STATS_REPLY 消息。

```
struct ofp_stats_reply {
struct ofp_header header;
uint16_t type;        /* One of the OFPST_* constants. */
uint16_t flags;       /* OFPSF_REPLY_* flags. */
uint8_t body[0];      /* Body of the reply. */
};
OFP_ASSERT(sizeof(struct    ofp_stats_reply) == 12);
```

其中，flags 用来标明是否有多条附加的回复，还是仅有一条。type 则定义信息的类型，以决定 body 中信息该如何解析。

```
enum ofp_stats_types {
/* Description of this OpenFlow switch.
* The request body is empty.
* The reply body is struct ofp_desc_stats. */
OFPST_DESC,

/* Individual flow statistics.
* The request body is struct ofp_flow_stats_request.
* The reply body is an array of struct ofp_flow_stats. */
OFPST_FLOW,

/* Aggregate flow statistics.
* The request body is struct ofp_aggregate_stats_request.
* The reply body is struct ofp_aggregate_stats_reply. */
OFPST_AGGREGATE,

/* Flow table statistics.
* The request body is empty.
* The reply body is an array of struct ofp_table_stats. */
OFPST_TABLE,

/* Physical port statistics.
* The request body is struct ofp_port_stats_request.
* The reply body is an array of struct ofp_port_stats. */
OFPST_PORT,

/* Queue statistics for a port
* The request body defines the port
* The reply body is an array of struct ofp_queue_stats */
OFPST_QUEUE,

/* Vendor extension.
* The request and reply bodies begin with a 32-bit vendor ID, which takes
* the same form as in "struct ofp_vendor_header".      The request and reply
* bodies are otherwise vendor-defined. */
OFPST_VENDOR = 0xffff
};
```

1.5.3.5.2 整体信息

OFPST_DESC 请求类型提供了制造商、软件、硬件版本、序列号等整体信息。

```
/* Body of reply to OFPST_DESC request.  Each entry is a NULL-terminated
* ASCII string. */
struct ofp_desc_stats {
char mfr_desc[DESC_STR_LEN];   /* Manufacturer description. */
char hw_desc[DESC_STR_LEN];    /* Hardware description. */
char sw_desc[DESC_STR_LEN];    /* Software description. */
char serial_num[SERIAL_NUM_LEN];   /* Serial number. */
char dp_desc[DESC_STR_LEN];    /* Human readable description of datapath. */
};
OFP_ASSERT(sizeof(struct  ofp_desc_stats)  == 1056);
```

每一项都是 ASCII 格式，以 null 结尾。DESC_STR_LEN 是 256，而 SERIAL_NUM_LEN 是 32。dp_desc 作为调试目的，可以自定义一些交换机标识信息。

```
/* Body for ofp_stats_request of type OFPST_FLOW. */
struct ofp_flow_stats_request  {
struct ofp_match match;      /* Fields to match. */
uint8_t table_id;   /* ID of table to read (from ofp_table_stats),
0xff for all tables or 0xfe for emergency. */
uint8_t pad;   /* Align to 32 bits. */
uint16_t out_port;       /* Require matching entries to include this
as an output port.   A value of OFPP_NONE
indicates no restriction. */
};
OFP_ASSERT(sizeof(struct  ofp_flow_stats_request)  == 44);
```

1.5.3.5.3 单流请求信息

OFPST_FLOW 请求类型则可以获取针对某个流的单独信息。

```
/* Body for ofp_stats_request of type OFPST_FLOW. */
struct ofp_flow_stats_request  {
struct ofp_match match;      /* Fields to match. */
uint8_t table_id;   /* ID of table to read (from ofp_table_stats), 0xff for all tables or
0xfe for emergency. */
uint8_t pad;   /* Align to 32 bits. */
uint16_t out_port;       /* Require matching entries to include this
as an output port.   A value of OFPP_NONE
indicates no restriction. */
};
OFP_ASSERT(sizeof(struct  ofp_flow_stats_request)  == 44);
```

match 域包括对于匹配流的描述。

table_id 描述要读取的流表的 id，0xff 则表示所有的流表。

out_port 是可选的对出口进行过滤的域。如果不是 OFPP_NONE，那么可以进行匹配过滤。注意如果不需要进行出口过滤，则需要被设置为 OFPP_NONE。

1.5.3.5.4 单流回复消息

回复信息包括下列的任意数组。

```
/* Body of reply to OFPST_FLOW request. */
struct ofp_flow_stats {
uint16_t length;     /* Length of this entry. */
uint8_t table_id;    /* ID of table flow came from. */
uint8_t pad;
struct ofp_match match;       /* Description of fields. */
uint32_t duration_sec;   /* Time flow has been alive in seconds. */
uint32_t duration_nsec;       /* Time flow has been alive in nanoseconds beyond
duration_sec. */
uint16_t priority;    /* Priority of the entry. Only meaningful
when this is not an exact-match entry. */
uint16_t idle_timeout;   /* Number of seconds idle before expiration. */
uint16_t hard_timeout;        /* Number of seconds before expiration. */
uint8_t pad2[6];     /* Align to 64-bits. */
uint64_t cookie;     /* Opaque controller-issued identifier. */
uint64_t packet_count;  /* Number of packets in flow. */
uint64_t byte_count;     /* Number of bytes in flow. */
struct ofp_action_header actions[0]; /* Actions. */
};
OFP_ASSERT(sizeof(struct    ofp_flow_stats) == 88);
```

这些域包括 flow_mod 消息中提供的项、要插入的流表、包计数、流量计数等。

duration_sec 和 duration_nsec 包括了流表项自创建起的时间。单位分别是秒和纳秒。

1.5.3.5.5 多流请求信息

OFPST_AGGREGATE 请求类型可以获取多流信息。

```
/* Body for ofp_stats_request of type OFPST_AGGREGATE. */
struct ofp_aggregate_stats_request {
struct ofp_match match;       /* Fields to match. */
uint8_t table_id;    /* ID of table to read (from ofp_table_stats)
0xff for all tables or 0xfe for emergency. */
uint8_t pad;   /* Align to 32 bits. */
uint16_t out_port;        /* Require matching entries to include this
as an output port.   A value of OFPP_NONE
indicates no restriction. */
};
OFP_ASSERT(sizeof(struct   ofp_aggregate_stats_request) == 44);
```

match、table_id 跟 out_port 作用跟单流信息类似。

1.5.3.5.6 多流回复信息

多流回复信息体位如下结构。

```
/* Body of reply to OFPST_AGGREGATE  request. */
struct ofp_aggregate_stats_reply {
uint64_t packet_count; /* Number of packets in flows. */
uint64_t byte_count;    /* Number of bytes in flows. */
uint32_t flow_count;    /* Number of flows. */
uint8_t pad[4];      /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct    ofp_aggregate_stats_reply) == 24);
```

1.5.3.5.7 表统计信息

表统计信息请求是通过 OFPST_TABLE 请求类型，该请求消息仅有消息头，不包括消息体。

表统计回复信息包括下面结构的一个数组。

```
/* Body of reply to OFPST_TABLE request. */
struct ofp_table_stats {
uint8_t table_id; /* Identifier of table. Lower numbered tables
are consulted first. */
uint8_t pad[3]; /* Align to 32-bits. */
char name[OFP_MAX_TABLE_NAME_LEN];
uint32_t wildcards; /* Bitmap of OFPFW_* wildcards that aresupported by the
table. */
uint32_t max_entries; /* Max number of entries supported. */
uint32_t active_count; /* Number of active entries. */
uint64_t lookup_count; /* Number of packets looked up in table. */
uint64_t matched_count; /* Number of packets that hit table. */
};
OFP_ASSERT(sizeof(struct ofp_table_stats) == 64);
```

body 域中包括通配符，来指明表中支持通配符的域。例如，直接 hash 表该域为 0，而顺序匹配表则为 OFPFW_ALL。各项以包经过表的顺序排序。

OFP_MAX_TABLE_NAME_LEN 是 32。

1.5.3.5.8 端口统计请求信息

端口状态的请求消息类型为 OFPST_PORT。

```
/* Body for ofp_stats_request of type OFPST_PORT. */
struct ofp_port_stats_request {
uint16_t port_no; /* OFPST_PORT message must request statistics
* either for a single port (specified in
* port_no) or for all ports (if port_no ==
* OFPP_NONE). */
uint8_t pad[6];
};
OFP_ASSERT(sizeof(struct ofp_port_stats_request) == 8);
```

port_no 用来过滤获取信息的端口，如果获取所有端口信息，则设置为 OFPP_NONE。

1.5.3.5.9 端口统计回复信息

回复消息体中包括下面结构的一个数组。

```
/* Body of reply to OFPST_PORT request. If a counter is unsupported, set
* the field to all ones. */
struct ofp_port_stats {
uint16_t port_no;
uint8_t pad[6];       /* Align to 64-bits. */
uint64_t rx_packets;      /* Number of received packets. */
uint64_t tx_packets;      /* Number of transmitted packets. */
uint64_t rx_bytes;  /* Number of received bytes. */
uint64_t tx_bytes;  /* Number of transmitted bytes. */
uint64_t rx_dropped;     /* Number of packets dropped by RX. */
uint64_t tx_dropped;     /* Number of packets dropped by TX. */
uint64_t rx_errors;       /* Number of receive errors.    This is a super-set
of more specific receive errors and should be
greater than or equal to the sum of all
rx_*_err values. */
uint64_t tx_errors;       /* Number of transmit errors.    This is a super-set
of more specific transmit errors and should be
greater than or equal to the sum of all
tx_*_err values (none currently defined.) */
uint64_t rx_frame_err;  /* Number of frame alignment errors. */

uint64_t rx_over_err;     /* Number of packets with RX overrun. */
uint64_t rx_crc_err;      /* Number of CRC errors. */
uint64_t collisions;      /* Number of collisions. */
};
OFP_ASSERT(sizeof(struct    ofp_port_stats) == 104);
```

对于不可用的计数器，交换机将返回-1。

1.5.3.5.10 队列信息

请求消息类型为 OFPST_QUEUE。消息体中 port_no 和 queue_id 分别指明要查询的端口和对列。如果查询全部，则分别置为 OFPP_ALL 和 OFPQ_ALL。

```
struct ofp_queue_stats_request    {
uint16_t port_no;  /* All ports if OFPT_ALL. */
uint8_t pad[2];      /* Align to 32-bits. */
uint32_t queue_id;       /* All queues if OFPQ_ALL. */
};
OFP_ASSERT(sizeof(struct    ofp_queue_stats_request)    == 8);
```

回复消息体中包括下面结构的一个数组。

```
struct ofp_queue_stats {
uint16_t port_no;
uint8_t pad[2];      /* Align to 32-bits. */
uint32_t queue_id;       /* Queue i.d */
uint64_t tx_bytes; /* Number of transmitted bytes. */
uint64_t tx_packets;     /* Number of transmitted packets. */
uint64_t tx_errors;/* Number of packets dropped due to overrun. */
};
OFP_ASSERT(sizeof(struct     ofp_queue_stats)  == 32);
```

1.5.3.5.11 生产商信息

生产商信息通过 OFPST_VENDOR 消息类型来请求获取，消息首 4 个字节为生产商的标志号。

vendor 域为 32 位，每个生产商是唯一的。如果首字节为 0，则剩下 3 个字节为生产商对应的 IEEE　OUI。

1.5.3.6　发包

如果控制器希望通过交换机发包，需要利用 OFPT_PACKET_OUT 消息。

```
/* Send packet (controller -> datapath). */
struct ofp_packet_out {
struct ofp_header header;
uint32_t buffer_id;        /* ID assigned by datapath (-1 if none). */
uint16_t in_port;    /* Packet's input port (OFPP_NONE if none). */
uint16_t actions_len;     /* Size of action array in bytes. */

struct ofp_action_header actions[0]; /* Actions. */
/* uint8_t data[0]; */     /* Packet data.    The length is inferred
from the length field in the header.
(Only meaningful if buffer_id == -1.) */
};
OFP_ASSERT(sizeof(struct     ofp_packet_out) == 16);
```

buffer_id 跟 ofp_packet_in 中给出的一致。如果 buffer_id 是-1，则网包内容被包括在 data 域中。如果 OFPP_TABLE 被指定为发出行为的出口，则 packet_out 中的 in_port 将被用来进行流表查询。

1.5.3.7　保障消息

控制器如果仅仅想确保消息依赖完整或指令完成，可以使用 OFPT_BARRIER_RE

QUEST 消息。该消息类型没有消息体。

交换机一旦收到该消息，则需要先执行完该消息前到达的所有指令，然后再执行其后的。之前指令处理完成后，交换机要回复 OFPT_BARRIER_REPLY 消息，且携带有原请求信息的 xid 信息。

1.5.4 Asynchronous 消息

包括包进入（Packet_In）、流删除（Flow Removed）、端口状态（Port Status）、错误（Error）等。

1.5.4.1 包进入

当网包被 datapath 获取并发给控制器的时候，使用 OFPT_PACKET_IN 消息。

```
/* Packet received on port (datapath -> controller). */
struct ofp_packet_in {
struct ofp_header header;
uint32_t buffer_id;       /* ID assigned by datapath. */
uint16_t total_len; /* Full length of frame. */
uint16_t in_port;   /* Port on which frame was received. */
uint8_t reason;     /* Reason packet is being sent (one of OFPR_*) */
uint8_t pad;
uint8_t data[0];    /* Ethernet frame, halfway through 32-bit word,
so the IP header is 32-bit aligned.    The
amount of data is inferred from the length
field in the header.    Because of padding,
offsetof(struct ofp_packet_in, data) ==
sizeof(struct ofp_packet_in) - 2. */
};
OFP_ASSERT(sizeof(struct    ofp_packet_in) == 20);
```

其中 buffer_id 是 datapath 用来标志一个在缓存中的网包。当包进入消息发给控制器的时候，部分网包信息也会被一并发给控制器。

如果网包是因为流表项行为指定（发给控制器）而发给控制器，则有 action_output 中的 max_len 字节数据发给控制器；如果是因为查不到匹配表项而发给控制器，则至少 OFPT_SET_CONFIG 消息中的 miss_send_len 字节发给控制器。默认 miss_send_len 是 128（字节）。如果网包没有放入缓存，则整个网包内容都需要发给控制器，此时 buffer_id 被设置为-1。

交换机需要负责在控制器接手前保护相关的 buffer 信息不被复用。

reason 域可以为如下类型。

```
/* Why is this packet being sent to the controller? */
enum ofp_packet_in_reason {
OFPR_NO_MATCH,      /* No matching flow. */
OFPR_ACTION      /* Action explicitly output to controller. */
};
```

1.5.4.2 流删除

如果控制器希望在流表项超时删除的时候得到通知，则 datapath 通过 OFPT_FLOW_REMOVED 消息类型来通知控制器。

```
/* Flow removed (datapath -> controller). */
struct ofp_flow_removed {
struct ofp_header header;
struct ofp_match match;    /* Description of fields. */
uint64_t cookie;   /* Opaque controller-issued identifier. */

uint16_t priority;  /* Priority level of flow entry. */
uint8_t reason;    /* One of OFPRR_*. */
uint8_t pad[1];    /* Align to 32-bits. */

uint32_t duration_sec; /* Time flow was alive in seconds. */
uint32_t duration_nsec;      /* Time flow was alive in nanoseconds beyond
duration_sec. */
uint16_t idle_timeout; /* Idle timeout from original flow mod. */
uint8_t pad2[2];   /* Align to 64-bits. */
uint64_t packet_count;
uint64_t byte_count;
};
OFP_ASSERT(sizeof(struct ofp_flow_removed) == 88);
```

其中，match、cookie、priority 等域的定义跟建立流表项时候一致。
reason 域可以为如下类型之一。

```
/* Why was this flow removed? */
enum ofp_flow_removed_reason {
OFPRR_IDLE_TIMEOUT,      /* Flow idle time exceeded idle_timeout. */
OFPRR_HARD_TIMEOUT,      /* Time exceeded hard_timeout. */
OFPRR_DELETE     /* Evicted by a DELETE flow mod. */
};
```

idle_timeout 则为添加表项时候指定。
packet_count 和 byte_count 分别用来计数与该流表项相关的包和流量信息。

1.5.4.3 端口状态

当物理端口被添加、删除或修改的时候，datapath 需要使用 OFPT_PORT_STATUS 消息来通知控制器。

```
/* A physical port has changed in the datapath */
struct ofp_port_status {
struct ofp_header header;
uint8_t reason;     /* One of OFPPR_*. */
uint8_t pad[7];     /* Align to 64-bits. */
struct ofp_phy_port desc;
};
OFP_ASSERT(sizeof(struct    ofp_port_status) == 64);
```

status 可以为如下的值之一。

```
/* What changed about the physical port */
enum ofp_port_reason {
OFPPR_ADD,  /* The port was added. */
OFPPR_DELETE,     /* The port was removed. */
OFPPR_MODIFY    /* Some attribute of the port has changed. */
};
```

1.5.4.4 错误

1.5.4.4.1 基本结构

当交换机需要通知控制器发生问题时，可以使用 OFPT_ERROR_MSG 消息。

```
/* OFPT_ERROR: Error message (datapath -> controller). */
struct ofp_error_msg {
struct ofp_header header;

uint16_t type;
uint16_t code;
uint8_t data[0];    /* Variable-length data.    Interpreted based
on the type and code. */
};
OFP_ASSERT(sizeof(struct    ofp_error_msg) == 12);
```

type 用来标明高层的错误类型，code 为对应类型的错误代码。data 则为错误相关的其他信息。

以_EPERM 结尾的 code 对应为权限问题。
目前错误类型有

```
/* Values for 'type' in ofp_error_message.  These values are immutable: they
* will not change in future versions of the protocol (although new values may
* be added). */
enum ofp_error_type {

    OFPET_HELLO_FAILED,  /* Hello protocol failed. */
    OFPET_BAD_REQUEST,  /* Request was not understood. */
    OFPET_BAD_ACTION,  /* Error in action description. */
    OFPET_FLOW_MOD_FAILED, /* Problem modifying flow entry. */
    OFPET_PORT_MOD_FAILED,  /* Port mod request failed. */
    OFPET_QUEUE_OP_FAILED  /* Queue operation failed. */
};
```

1.5.4.4.2 错误类型对应代码

对 OFPET_HELLO_FAILED 错误类型，错误代码有

```
/* ofp_error_msg 'code' values for OFPET_HELLO_FAILED.  'data' contains an
* ASCII text string that may give failure details. */
enum ofp_hello_failed_code {
OFPHFC_INCOMPATIBLE,  /* No compatible version. */
OFPHFC_EPERM  /* Permissions error. */
};
```

data 域包括 ASCII 字符串来描述错误的一些细节。
对于 OFPET_BAD_REQUEST 错误类型，错误代码有

```
/* ofp_error_msg 'code' values for OFPET_BAD_REQUEST.  'data' contains at least
* the first 64 bytes of the failed request. */
enum ofp_bad_request_code  {
OFPBRC_BAD_VERSION,  /* ofp_header.version not supported. */
OFPBRC_BAD_TYPE,  /* ofp_header.type not supported. */
OFPBRC_BAD_STAT,  /* ofp_stats_request.type  not supported. */
OFPBRC_BAD_VENDOR,  /* Vendor not supported (in ofp_vendor_header
* or ofp_stats_request or ofp_stats_reply). */

    OFPBRC_BAD_SUBTYPE, /* Vendor subtype not supported. */
    OFPBRC_EPERM,  /* Permissions error. */
    OFPBRC_BAD_LEN, /* Wrong request length for type. */
    OFPBRC_BUFFER_EMPTY,  /* Specified buffer has already been used. */
    OFPBRC_BUFFER_UNKNOWN  /* Specified buffer does not exist. */
};
```

data 域至少包括 64 字节的失败请求。
对于 OFPET_BAD_ACTION 错误类型，错误代码有

```
/* ofp_error_msg 'code' values for OFPET_BAD_ACTION.  'data' contains at least
* the first 64 bytes of the failed request. */
enum ofp_bad_action_code  {

    OFPBAC_BAD_TYPE,  /* Unknown action type. */
    OFPBAC_BAD_LEN, /* Length problem in actions. */
    OFPBAC_BAD_VENDOR, /* Unknown vendor id specified. */
    OFPBAC_BAD_VENDOR_TYPE,  /* Unknown action type for vendor id. */
    OFPBAC_BAD_OUT_PORT,  /* Problem validating output action. */
    OFPBAC_BAD_ARGUMENT,  /* Bad action argument. */
    OFPBAC_EPERM,  /* Permissions error. */
    OFPBAC_TOO_MANY,  /* Can't handle this many actions. */
    OFPBAC_BAD_QUEUE  /* Problem validating output queue. */
};
```

data 域至少包括 64 字节的失败请求。
对于 OFPET_FLOW_MOD_FAILED 错误类型，错误代码有

```
/* ofp_error_msg 'code' values for OFPET_FLOW_MOD_FAILED. 'data' contains
* at least the first 64 bytes of the failed request. */
enum ofp_flow_mod_failed_code {
OFPFMFC_ALL_TABLES_FULL, /* Flow not added because of full tables. */
OFPFMFC_OVERLAP, /* Attempted to add overlapping flow with
* CHECK_OVERLAP flag set. */
OFPFMFC_EPERM, /* Permissions error. */
OFPFMFC_BAD_EMERG_TIMEOUT, /* Flow not added because of non-zero
idle/hard
* timeout. */
OFPFMFC_BAD_COMMAND, /* Unknown command. */
OFPFMFC_UNSUPPORTED /* Unsupported action list - cannot process in
* the order specified. */
};
```

data 域至少包括 64 字节的失败请求。

对于 OFPET_PORT_MOD_FAILED 错误类型，错误代码有

```
/* ofp_error_msg 'code' values for OFPET_PORT_MOD_FAILED. 'data' contains
* at least the first 64 bytes of the failed request. */
enum ofp_port_mod_failed_code {
OFPPMFC_BAD_PORT, /* Specified port does not exist. */
OFPPMFC_BAD_HW_ADDR, /* Specified hardware address is wrong. */
};
```

data 域至少包括 64 字节的失败请求。

对于 OFPET_QUEUE_OP_FAILED 错误类型，错误代码有

```
/* ofp_error msg 'code' values for OFPET_QUEUE_OP_FAILED. 'data' contains
* at least the first 64 bytes of the failed request */
enum ofp_queue_op_failed_code {
OFPQOFC_BAD_PORT, /* Invalid port (or port does not exist). */

OFPQOFC_BAD_QUEUE, /* Queue does not exist. */ OFPQOFC_EPERM /*
Permissions error. */
};
```

data 域至少包括 64 字节的失败请求。

如果错误消息是回复给控制器的指定请求，如 OFPET_BAD_REQUEST，OFPET_BAD_ACTION 或 OFPET_FLOW_MOD_FAILED，则消息头中的 xid 需要跟对应请求消息一致。

1.5.5 Symmetric 消息

包括 Hello、响应请求、回复、生产商信息等。

1.5.5.1 Hello

OFPT_HELLO 消息没有消息体，仅有 of 消息头。扩展版本应该能解释带有消息体的 hello 消息，并将其消息体内容忽略。

1.5.5.2 响应请求

响应请求消息由一个 of 消息头加上任意的消息体组成，用来协助测量延迟、带宽、控制器跟交换机之间是否保持连接等信息。

1.5.5.3 响应回复

响应回复消息由一个 of 消息头加上对应请求的无修改消息体组成，用来协助测量延迟、带宽、控制器跟交换机之间是否保持连接等信息。

1.5.5.4 生产商信息

生产商消息结构如下

```
/* Vendor extension. */
struct ofp_vendor_header {
struct ofp_header header;  /* Type OFPT_VENDOR */
uint32_t vendor;
/* Vendor ID:
* - MSB 0: low-order bytes are IEEE OUI.
* - MSB != 0: defined by OpenFlow  consortium. */
/* Vendor-defined arbitrary additional data. */
};
OFP_ASSERT(sizeof(struct  ofp_vendor_header) == 12);
```

vendor 域为 32 位长，如果首字节为 0，则其他 3 个字节定义为 IEEE OUI。

如果交换机无法理解一条生产商消息，它需要发送 OFPT_ERROR 消息带有 OFPBRC_BAD_VENDOR 错误代码和 OFPET_BAD_REQUEST 错误代码。