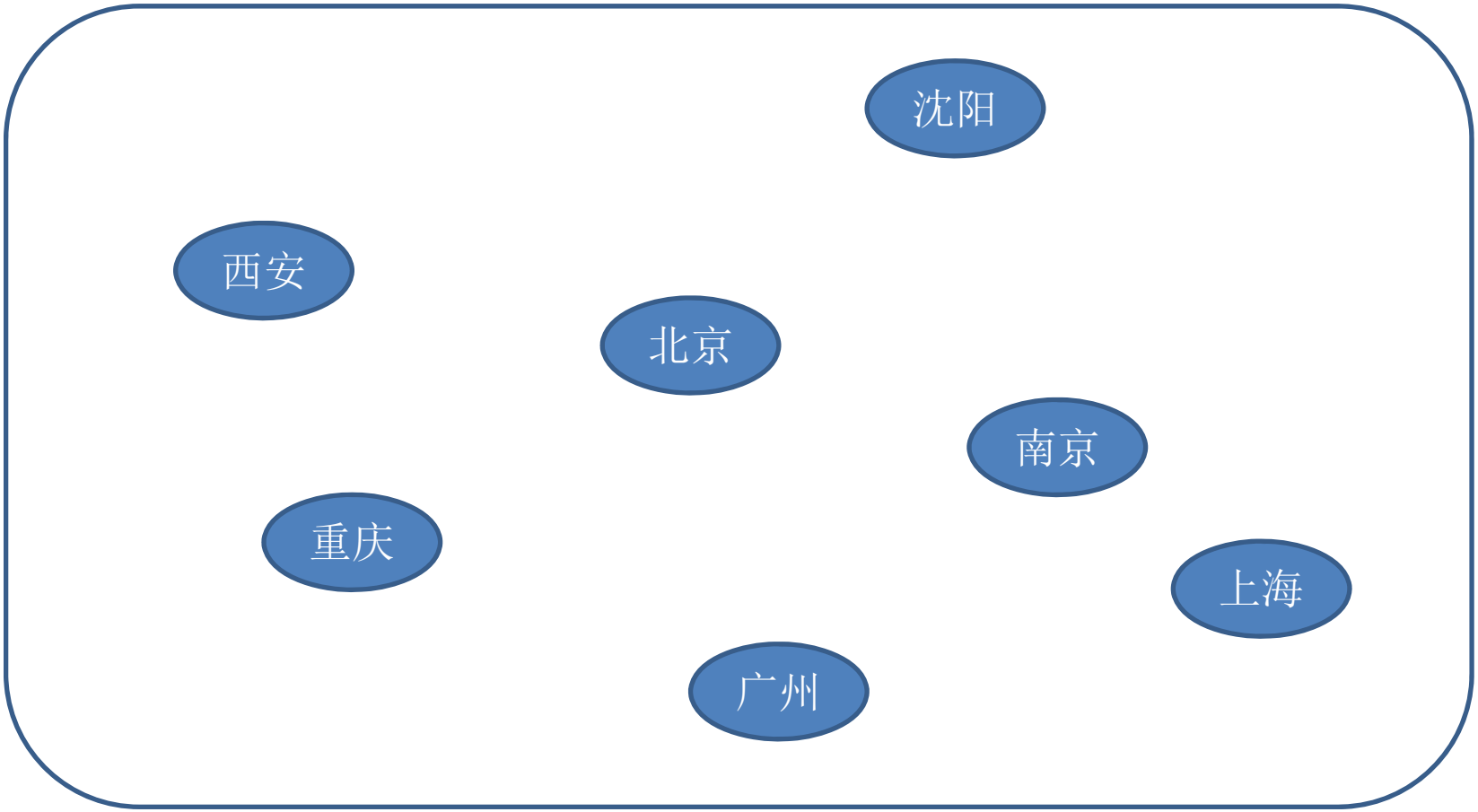
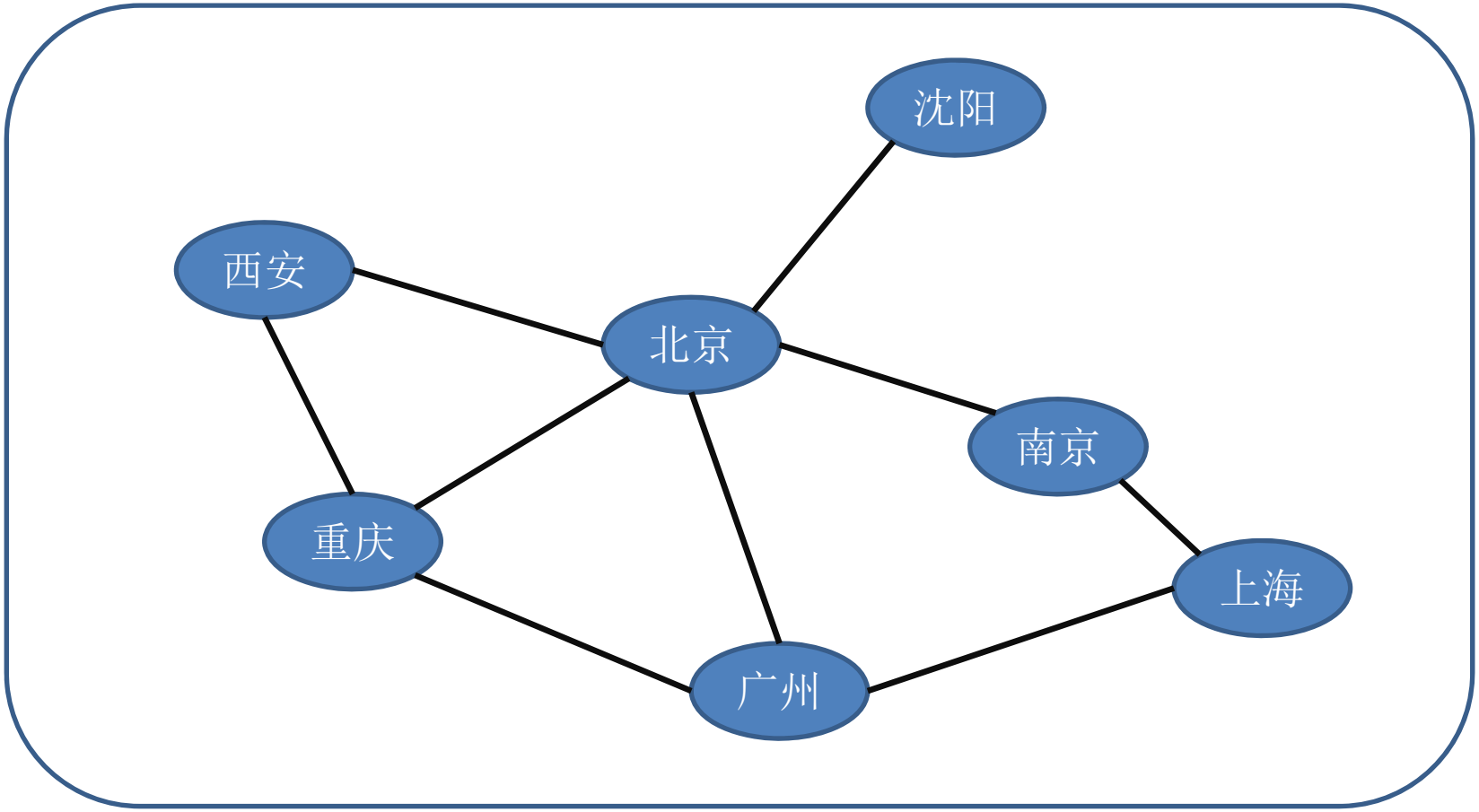
Openflow协议基础入门

北京邮电大学
未来网络实验室
张健男

思考一个问题：
假设有如下N个城市，他们互相不连通，要想实现这几个城市之间的交流我们需要做哪些事情？

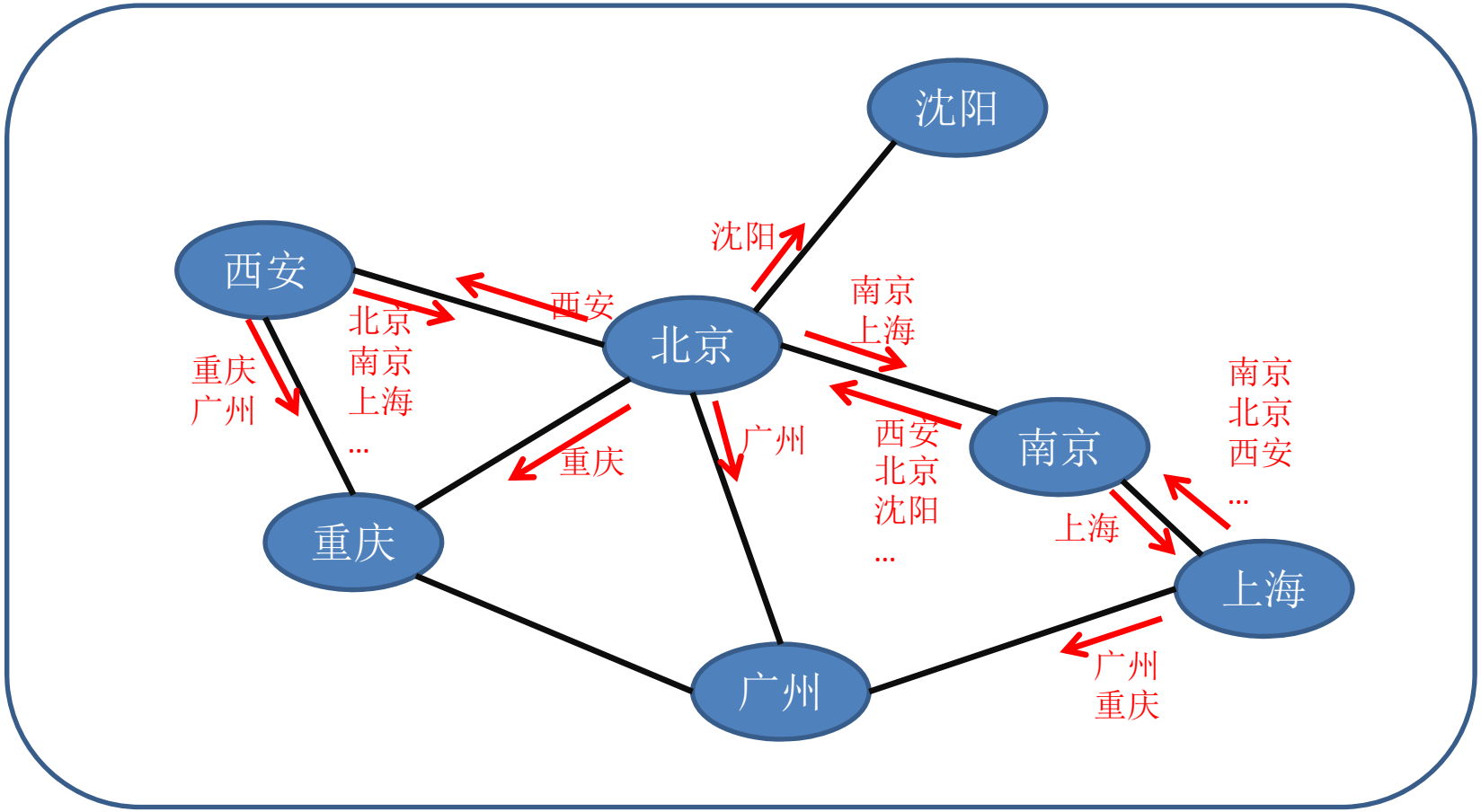


Step1：建设公路网



公路网络包含城市（节点）以及连接城市的公路（边）。
有了公路网，就可以保证任意两个城市之间物理连通。
但是物理上连通就解决问题了吗？

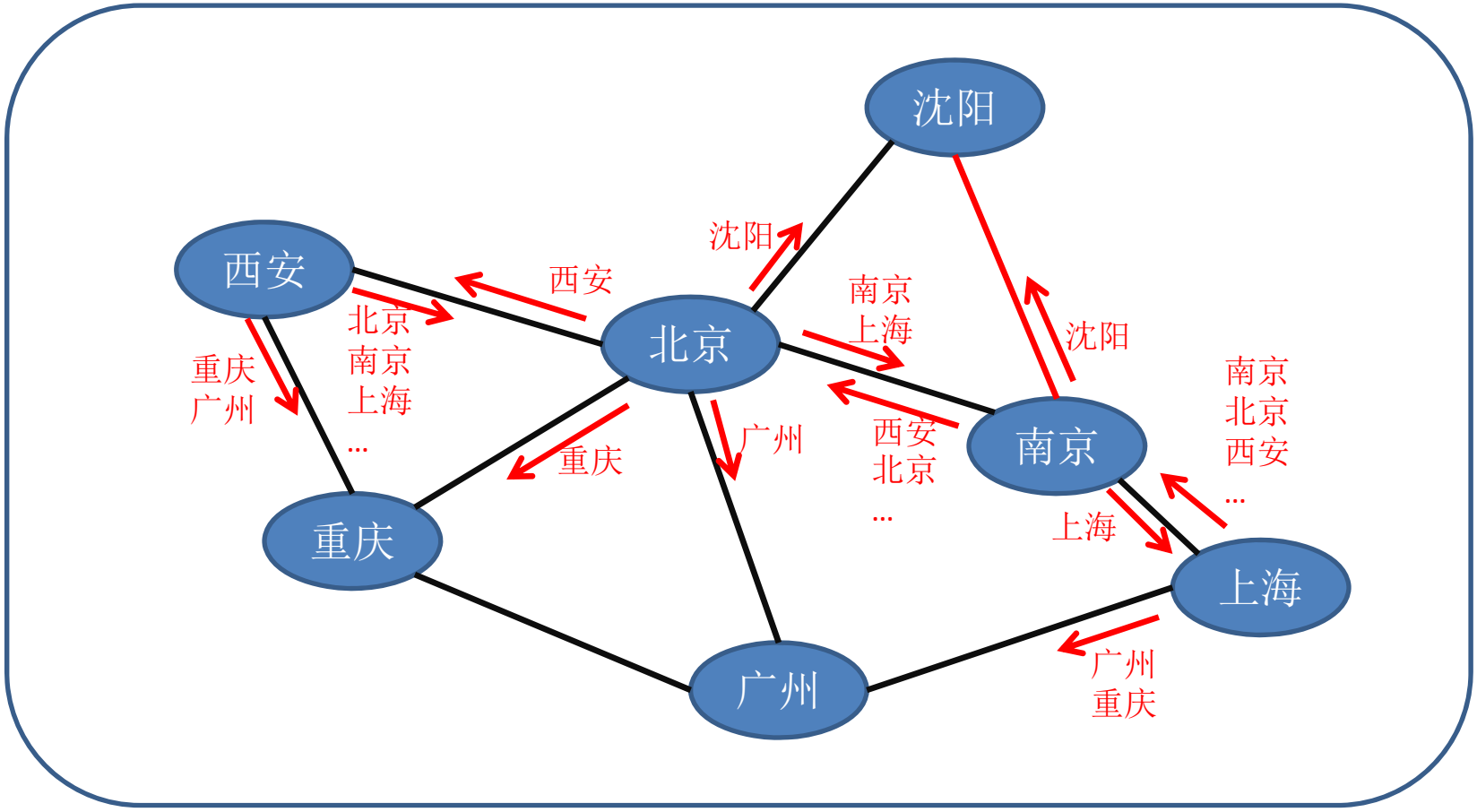
Step2：添加路标



添加路标后，交通流量就知道去往目的地的正确方向。

但是路标就一定是正确的吗？

Step3：建立路标信息维护机制



路标信息维护机制确保公路网络拓扑发生变化时能够更新路标信息，保障路标信息的正确性和最优化。

公路网络正常运行的条件

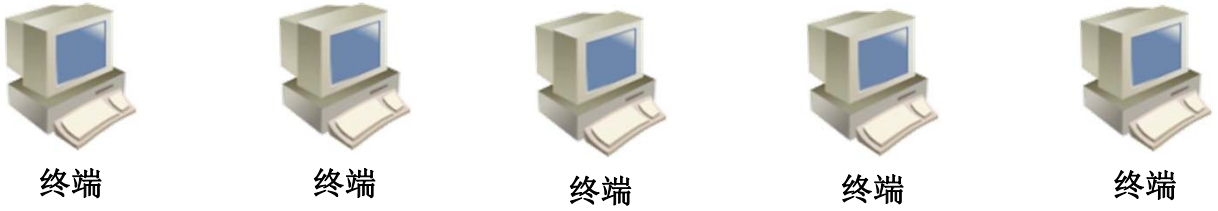
1、物理连通

2、路标信息

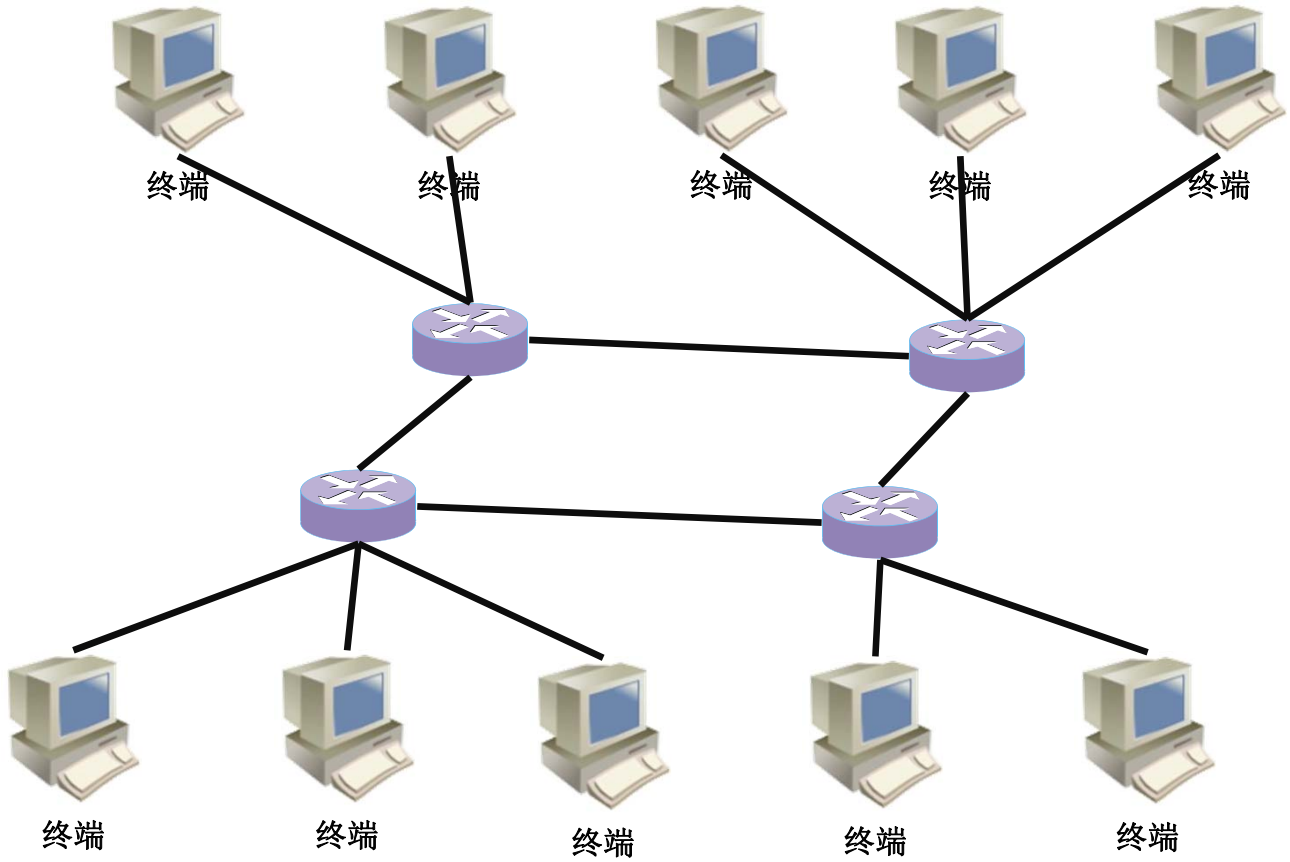
3、路标信息的维护机制

再思考一个问题：

有大量的电脑，现在希望这些电脑互相通信，我们需要做哪些事情？

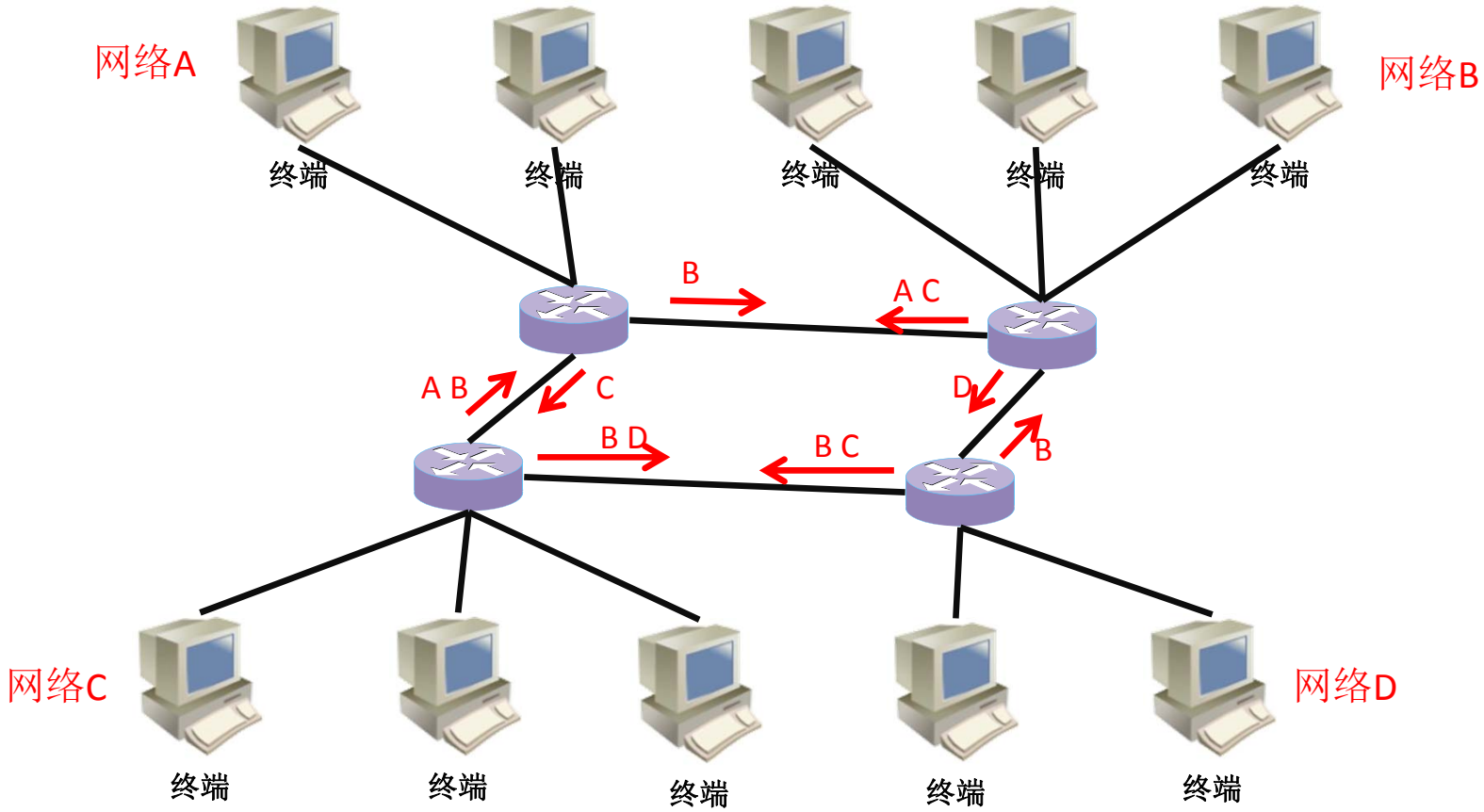


Step1：建设计算机网络

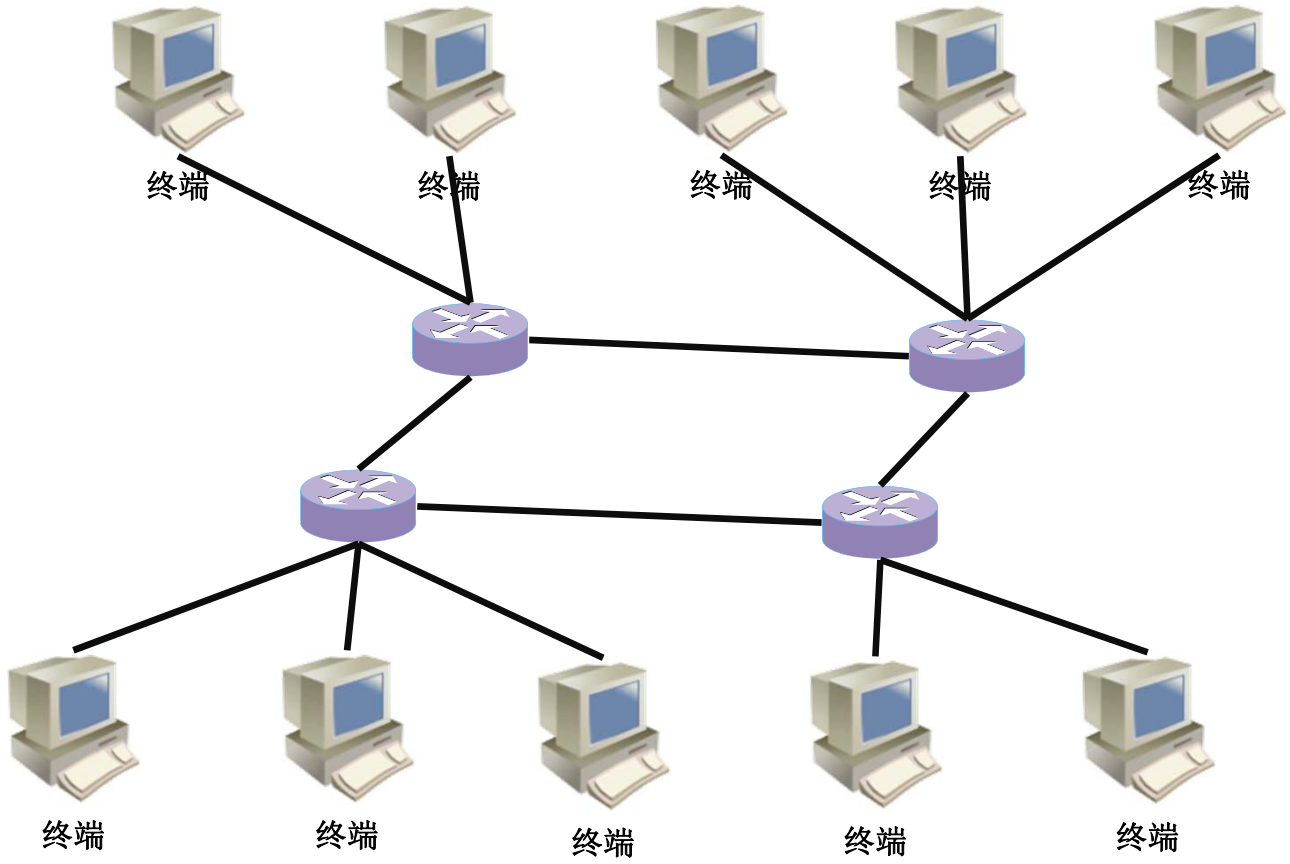


计算机网络包括网络设备（节点）和链路（边）
任意两台计算机之间有信号的物理通路

Step2：创建转发表

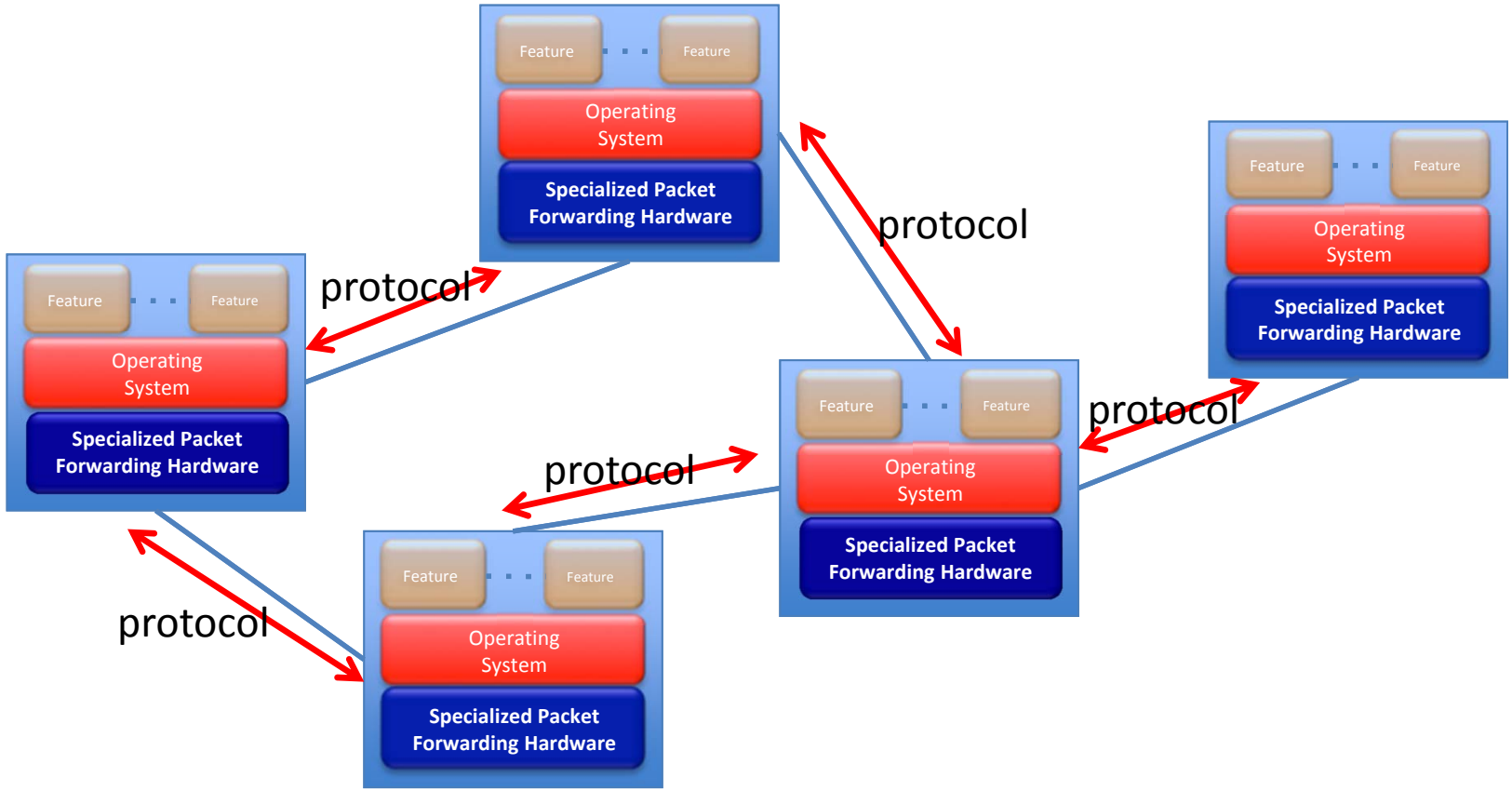


Step3：建立转发表维护机制



Step2中的转发表是由转发表维护机制生成的

传统网络设备工作方式：
所有设备运行相同的协议，通过协议信息交流生成转发表



传统网络设备工作方式

Hub

- 工作原理：基于物理端口转发
- 策略：Flood

L2Switch

- 工作原理：基于MAC地址表转发
- 策略：STP+MAC地址学习

Router

- 工作原理：基于路由表转发
- 策略：静态路由+动态路由协议

共同点：分布式策略(策略的制定者为设备本身)

转发表的结构：

编号	数据包匹配特征	数据包处理方法
1	特征1	处理方法1
2	特征2	处理方法2
……	……	……
N	特征N	处理方法N

HUB的转发表结构

编号	入端口	数据包处理方法
1	入端口1	从其他所有端口转发
2	入端口2	从其他所有端口转发
……	……	……
N	入端口N	从其他所有端口转发

L2Switch的转发表结构

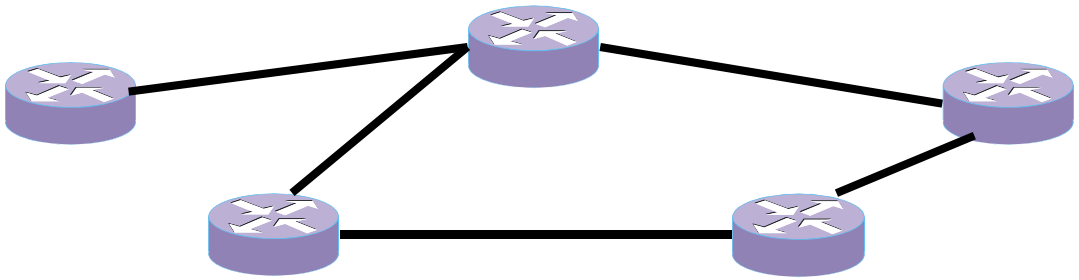
编号	目的MAC	数据包处理方法
1	目的MAC1	出口1
2	目的MAC2	出口2
……	……	……
N	目的MACN	出口N

Router的转发表结构

编号	目的IP网段	数据包处理方法
1	目的IP网段1	出口1
2	目的IP网段2	出口2
……	……	……
N	目的IP网段N	出口N

软件定义网络

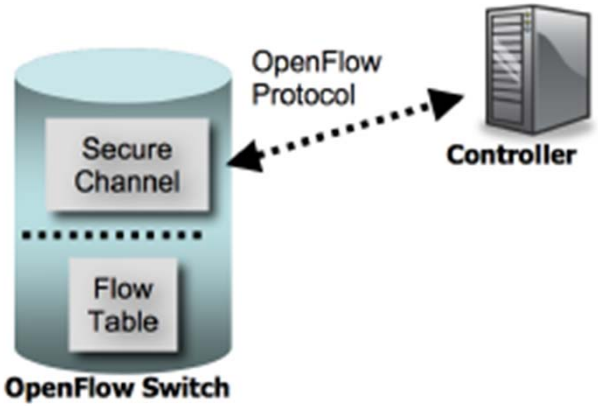
控制器：控制器知道所有网络信息，负责指挥设备如何工作



交换机：交换机不知道任何网络信息，只会按照控制器的指挥工作

软件定义网络的网络设备之间不运行任何协议，网络设备的转发表由控制器配置生成。
控制器与网络设备之间需要一种协议来互相通信

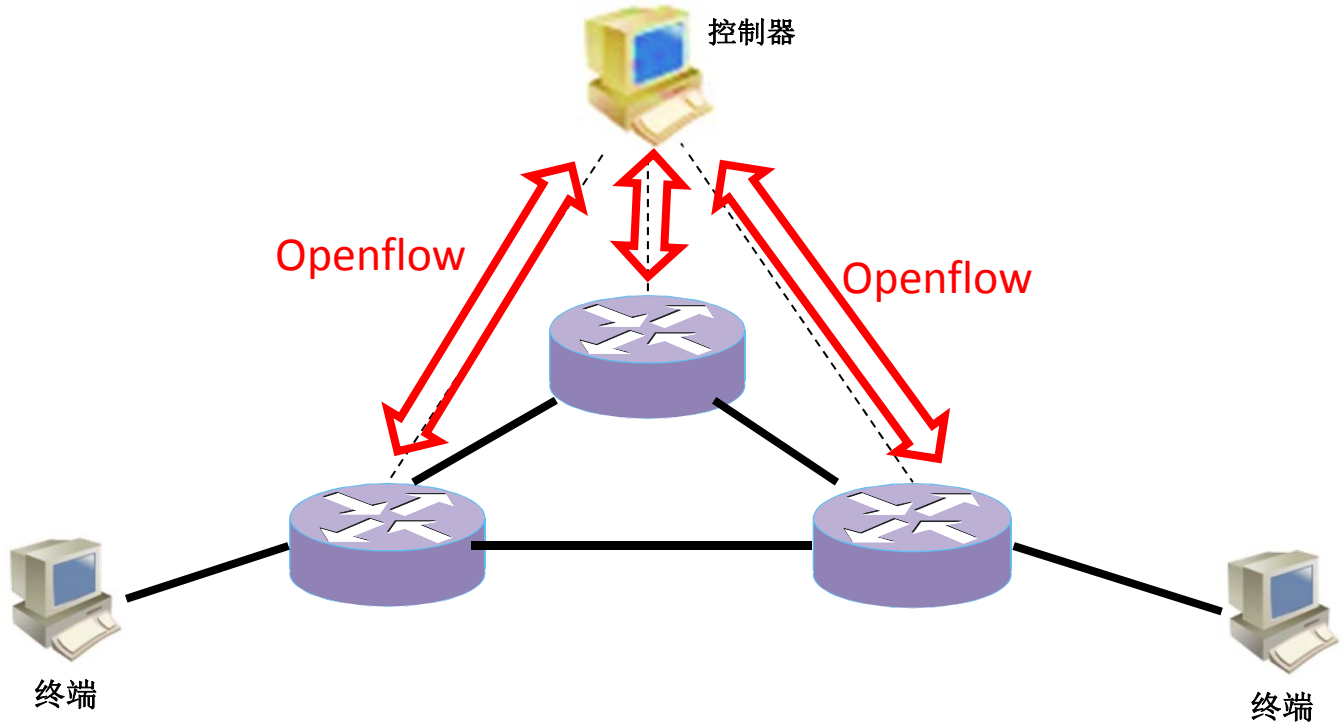
基于Openflow协议的软件定义网络



SDN网络分为两张网

1、数据网(数据平面)：Traffic

2、信令网(控制平面)：Openflow Protocol



Openflow交换机（Openflow1.0）

Openflow交换机中的转发表称为流表（Flow table）
流表包含数据包匹配特征和数据包处理方法

数据包匹配特征：

- 一层：交换机入端口（Ingress Port）
- 二层：源MAC地址（Ether source）、目的MAC地址（Ether dst）、以太网类型（EtherType）、VLAN标签（VLAN id）、VLAN优先级（VLAN priority）
- 三层：源IP（IP src）、目的IP（IP dst）、IP协议字段（IP proto）、IP服务类型（IP ToS bits）
- 四层：TCP/UDP源端口号（TCP/UDP src port）、 TCP/UDP目的端口号（TCP/UDP dst port）

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN pri-or-ity	IP src	IP dst	IP proto	IP ToS bits	TCP/ UDP src port	TCP/ UDP dst port

数据包处理方法：

- 转发
- 修改包头

Openflow1.0对数据包匹配特征的描述方法

```
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);
```

ofp_match的wildcard

```
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). */
```

ofp_match的wildcard

22-31	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	IP服务类型	VLAN优先级	目的IP						源IP						目的端口	源端口	IP协议字段	以太网类型	目的MAC	源MAC	VLAN标签	入端口

除源IP和目的IP以外，掩码位为0表示对应匹配项需要精确匹配，掩码为为1表示忽略匹配项。
源IP和目的IP字段的掩码表示32bitIP地址可以忽略匹配的长度。
如果源IP的掩码为8（wildcard的8-13bit为001000），表示源IP字段的高24bit需要精确匹配，源IP字段的低8bit可以忽略。

Openflow1.0对数据包处理方法的描述方法

Openflow1.0提供两种数据包的处理方法：

- 转发（Forward）
- 修改包头（Modify field）
  - SET_VLAN_VID 修改VLAN标签
  - SET_VLAN_PCP 修改VLAN优先级
  - STRIP_VLAN 弹出VLAN标签
  - SET_DL_SRC 修改源MAC地址
  - SET_DL_DST 修改目的MAC地址
  - SET_NW_SRC 修改源IP地址
  - SET_NW_DST 修改目的IP地址
  - SET_NW_TOS 修改IP服务类型字段
  - SET_TP_SRC 修改源端口号
  - SET_TP_DST 修改目的端口号

以上每一种操作称为一个动作（Action），流表中的数据包处理方法是一个动作列表（Action List），动作列表由以上各种动作组合合成。

Action头，包括Type和len字段

```
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];
};
```

Type为一下类型之一

```
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
};
```

其中OFPAT_OUTPUT和OFPAT_ENQUEUE为转发操作，其他类型为修改包头操作

Action——OUTPUT类型

```
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. */
};
```

Output类型Action的结构包含一个port参数和一个max_len参数
Port参数指定了数据包的输出端口，输出端口可以是交换机的一个实际物理端口，也可以是一下虚拟端口

- ALL：将数据包从除入端口以外其他所有端口发出
- CONTROLLER：将数据包发送给控制器
- LOCAL：将数据包发送给交换机本地端口
- TABLE：将数据包按照流表匹配条目处理
- IN_PORT：将数据包从入端口发出
- NORMAL：按照普通二层交换机流程处理数据包
- FLOOD：将数据包从最小生成树使能端口转发（不包括入端口）

当port为CONTROLLER时，max_len指定了发给CONTROLLER的数据包最大长度。当port为其他参数时，max_len无意义。

Action——ENQUEUE类型

```
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. */
};
```

Actions——修改VLANID

```
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];
};
```

Action——修改VLAN优先级

```
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];
};
```

Action——修改MAC地址

```
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];
};
```

Action——修改IP地址

```
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. */
};
```

Action——修改IP服务类型

```
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];
};
```

Action——修改传输层端口号

```
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];
};
```

Openflow交换机流表

编号	Openflow Match	Action List
1	match1	actions1
2	match2	actions2
…	…	…

每条流表条目还包括Counter字段，用来保存与条目相关的统计信息

Header Fields	Counters	Actions

控制器如何写流表？

Openflow消息

Openflow消息总共分为三大类：

1、Controller-to-Switch

控制器至交换机消息此类消息由控制器主动发出

- Features 用来获取交换机特性
- Configuration 用来配置Openflow交换机
- Modify-State 用来修改交换机状态(修改流表)
- Read-Stats 用来读取交换机状态
- Send-Packet 用来发送数据包
- Barrier 阻塞消息

2、Asynchronous

异步消息此类消息由交换机主动发出

- Packet-in 用来告知控制器交换机接收到数据包
- Flow-Removed 用来告知控制器交换机流表被删除
- Port-Status 用来告知控制器交换机端口状态更新
- Error 用来告知控制器交换机发生错误

3、Symmetric

对称消息，可以由控制器或交换机主动发起

- Hello 用来建立Openflow连接
- Echo 用来确认交换机与控制器之间的连接状态
- Vendor 厂商自定义消息

Openflow消息格式

Openflow协议数据包由Openflow Header和Openflow Message两部分组成

Openflow Header的结构：

```
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. */
};
```

Openflow Message结构与具体消息类型有关

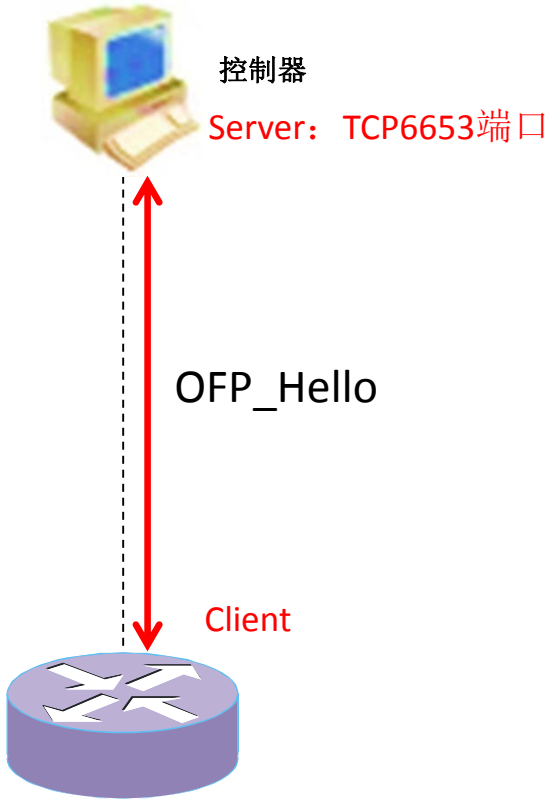
Openflow消息类型

```
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 */

};
```

建立Openflow连接

控制器与交换机互相发送Hello消息
Hello消息中只包含Openflow Header
Openflow Header中的version字段为发送方所支持的最高版本Openflow协议
双方选取Hello消息中最低版本的协议作为通信协议
如果有一方不支持Openflow协议版本，应发送Error消息后断开连接

如果双方Openflow版本可以兼容，则Openflow连接建立成功。

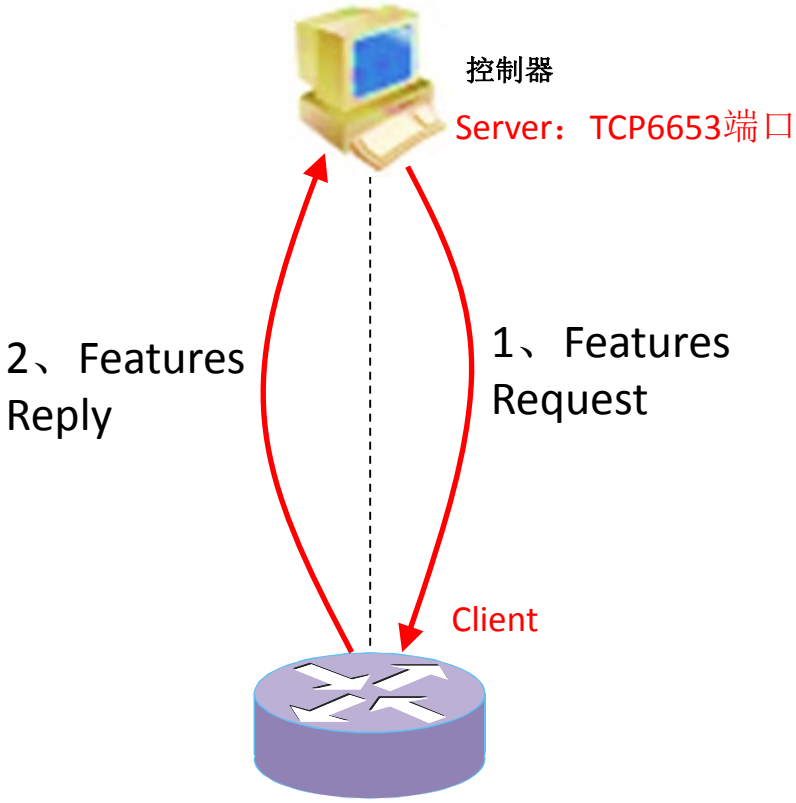


Openflow连接建立后，控制器最关心的事情是什么？

获取交换机特性（Features）信息

Openflow连接建立后，控制器最需要获得交换机的特性信息，交换机的特性信息包括交换机的ID(DPID)，交换机缓冲区数量，交换机端口及端口属性等等。

控制器向交换机发送Features Request消息查询交换机特性，Features Request消息只包含Openflow Header。
交换机在收到Features Request消息后返回Features Reply消息，Features Reply消息包括Openflow Header 和Features Reply Message



Features Reply Message结构

```
/* 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. */
};
```

datapath_id为交换机独一无二的ID号
n_buffers为交换机可以同时缓存的最大数据包个数
n_tables为交换机的流表数量
Capabilities表示交换机支持的特殊功能
Actions表示交换机支持的动作（见ofp_action_type）
ofp_phy_ports为交换机的物理端口描述列表

```
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. */
};
```

物理端口描述

```
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. */
};
```

port_no为物理端口的编号
hw_addr为端口的MAC地址
name为端口的名称
config为端口的配置
State为端口状态
curr, advertised supported,peer为端口物理属性

```
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. */
};
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. */
};
```

配置交换机Openflow属性

```
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. */
};

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
};
```

Openflow交换机只有两个属性需要控制器配置

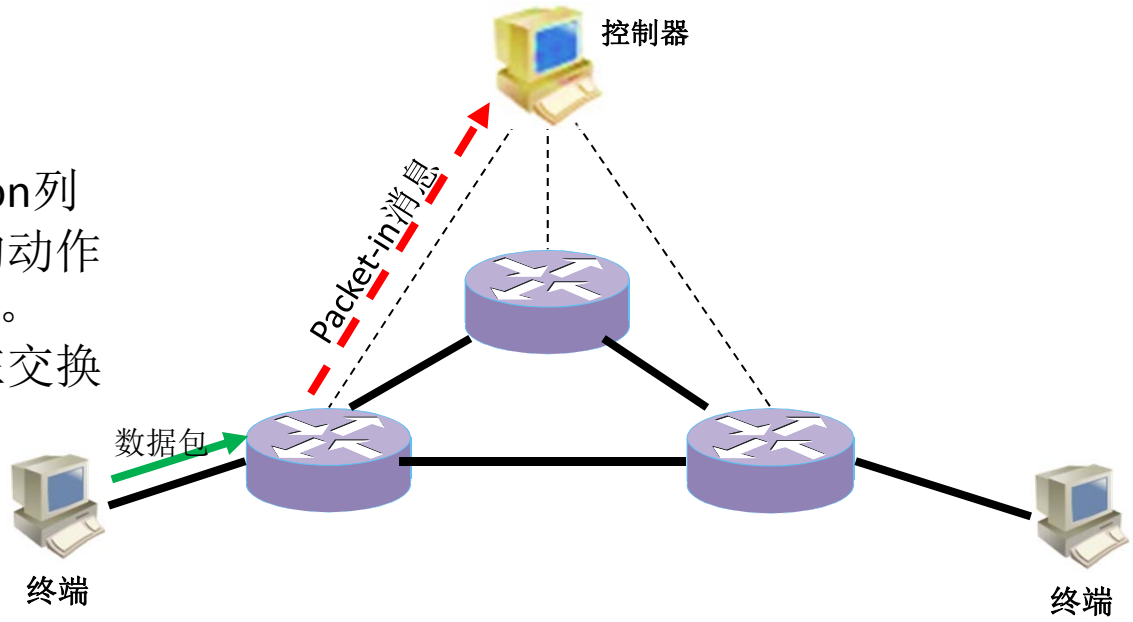
第一个属性为flags，用来指示交换机如何处理IP分片数据包

第二个属性为miss_send_len，用来指示当一个交换机无法处理的数据包到达时，将数据包的发给控制器的最大字节数。

Packet-in事件（交换机接收数据包）

Packet-in消息触发情况1：
当交换机收到一个数据包后，会查找流表，找出与数据包包头相匹配的条目。
如果流表中有匹配条目，则交换机按照流表所指示的action列表处理数据包。
如果流表中没有匹配条目，则交换机会将数据包封装在Packet-in消息中发送给控制器处理。此时数据包会被缓存在交换机中等待处理。

Packet-in消息触发情况2：
交换机流表所指示的action列表中包含转发给控制器的动作（Output=CONTROLLER）。
此时数据包不会被缓存在交换机中。



Packet-in消息格式

```
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. */
};
```

buffer_id为packet-in事件所携带的数据包在交换机中的缓存区ID

total_len为data段的长度

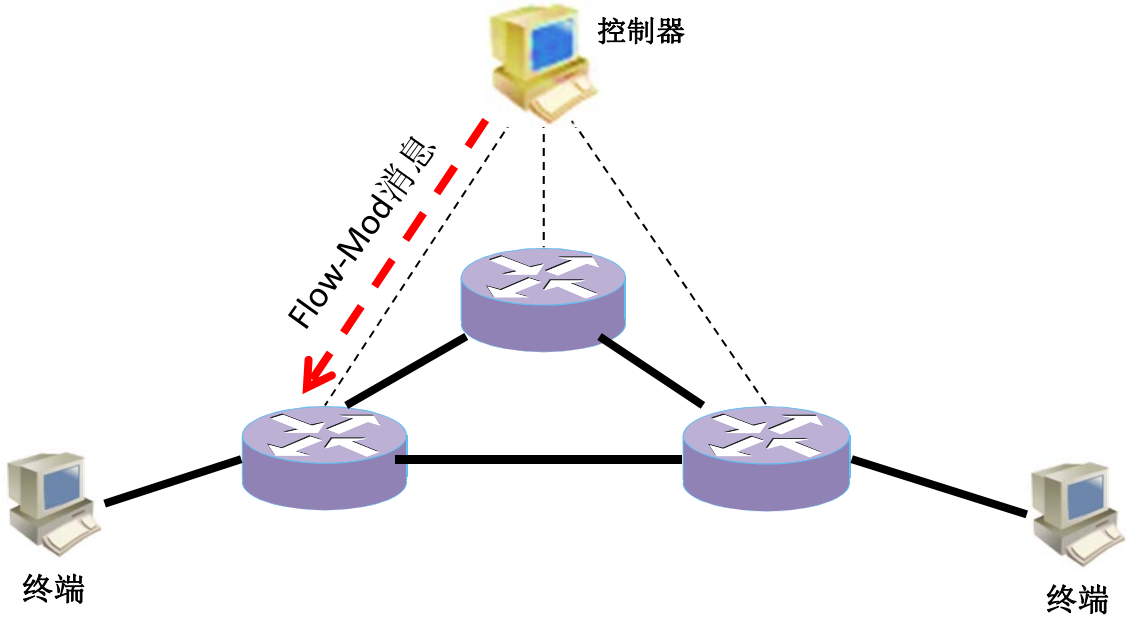
in_port数据包进入交换机的入接口号

Reason为packet-in事件产生的原因

```
enum ofp_packet_in_reason {
    OFPR_NO_MATCH,          /* No matching flow. */
    OFPR_ACTION             /* Action explicitly output to controller. */
};
```

控制器配置流表（Flow-Mod消息）

Flow-Mod消息用来添加、删除、修改Openflow交换机的流表信息
Flow-Mod消息共有五种类型：
ADD、DELETE、DELETE-STRICT、MODIFY、MODIFY-STRICT
ADD类型的flow-mod消息用来添加一条新的流表项
DELETE类型的flow-mod消息用来删除所有符合一定条件的流表项
DELETE-STRICT类型的flow-mod消息用来删除某一条指定的流表项
MODIFY类型的flow-mod消息用来修改所有符合一定条件的流表项
MODIFY-STRICT类型的flow-mod消息用来修改某一条指定的流表项



Flow-Mod消息格式

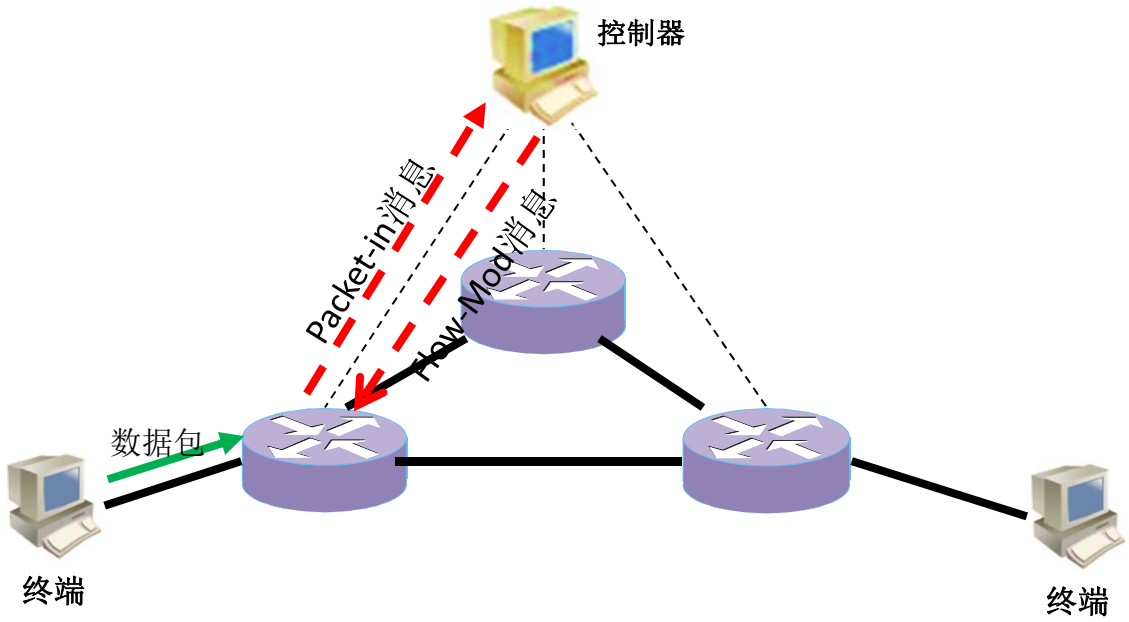
```
struct ofp_flow_mod {
    struct ofp_header header;
    struct ofp_match match;      /* Fields to match */
    uint64_t cookie;             /* Opaque controller-issued identifier. */

    /* 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. */
};
```

Match为流表的match域
Cookie为控制器定义的流表项标识符
Command是flow-mod的类型，可以是ADD、DELETE、DELETE-STRICT、MODIFY、MODIFY-STRICT
Idle_timeout为流表项的空闲超时时间
Hard_timeout为流表项的最大生存时间
Priority为流表项的优先级，交换机优先匹配高优先级的流表项
Buffer_id为交换机中的缓冲区ID，flow-mod消息可以指定一个缓冲区ID，该缓冲区中的数据包会按照此flow-mod消息的action列表处理
Out_port为删除流表的flow_mod消息提供额外的匹配参数
Flags为flow-mod命令的一些标志位，可以用来指示流表删除后是否发送flow-removed消息，添加流表时是否检查流表重复项，添加的流表项是否为应急流表项。
Actions为Action列表

用Flow-Mod消息响应Packet-in消息

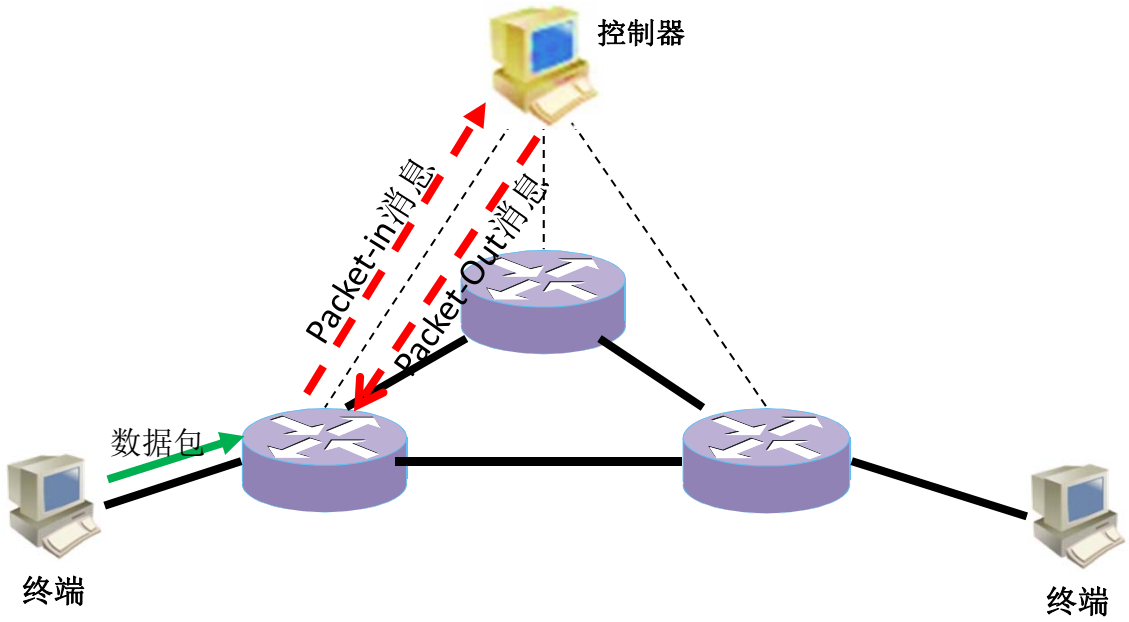
当交换机收到一个数据包并且交换机中没有与该数据包匹配的流表项时，交换机将此数据包封装到Packet-in消息中发送给控制器，并且交换机会将该数据包缓存。

控制器收到Packet-in消息后，可以发送flow-mod消向交换机写一个流表项。并且将flow-mod消息中的buffer_id字段设置为packet-in消息中的buffer_id值。从而控制器向交换机写入了一条与数据包相关的流表项，并且指定该数据包按照此流表项的aciton列表处理。



交换机转发数据包（Packet-Out）

并不是所有的数据包都需要向交换机中添加一条流表项来匹配处理，网络中还存在多种数据包，它出现的数量很少（如ARP、IGMP等），以至于没有必要通过流表项来指定这一类数据包的处理方法。
此时，控制器可以使用PacketOut消息，告诉交换机某一个数据包如何处理。



Packet-Out消息格式

```
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.) */
};
```

Buffer_id为交换机中缓冲区ID号，当buffer_id为-1时，指定的缓冲区为packet-out消息的data段

In_port为Packet-Out消息提供额外的匹配信息，当Packet-Out的buffer_id为-1，并且action列表中指定了Output=TABLE的动作，in_port将作为data段数据包的额外匹配信息进行流表查询

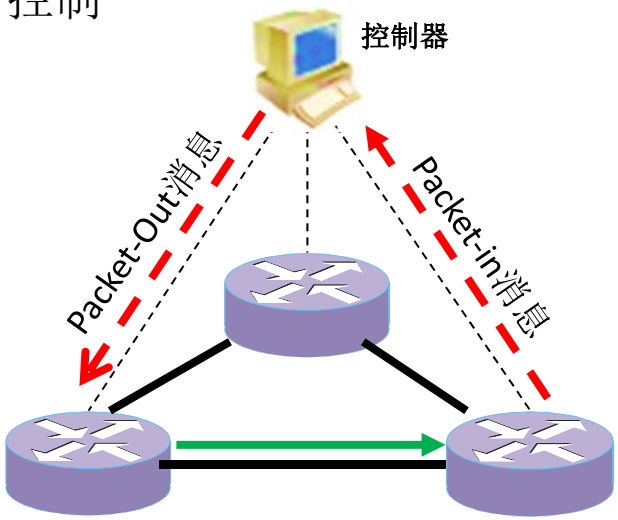
Actions_len指定了action列表的长度，用来区分actions和data段

Data为一个缓冲区，可以存储一个以太网帧

Packet-Out消息的应用场景

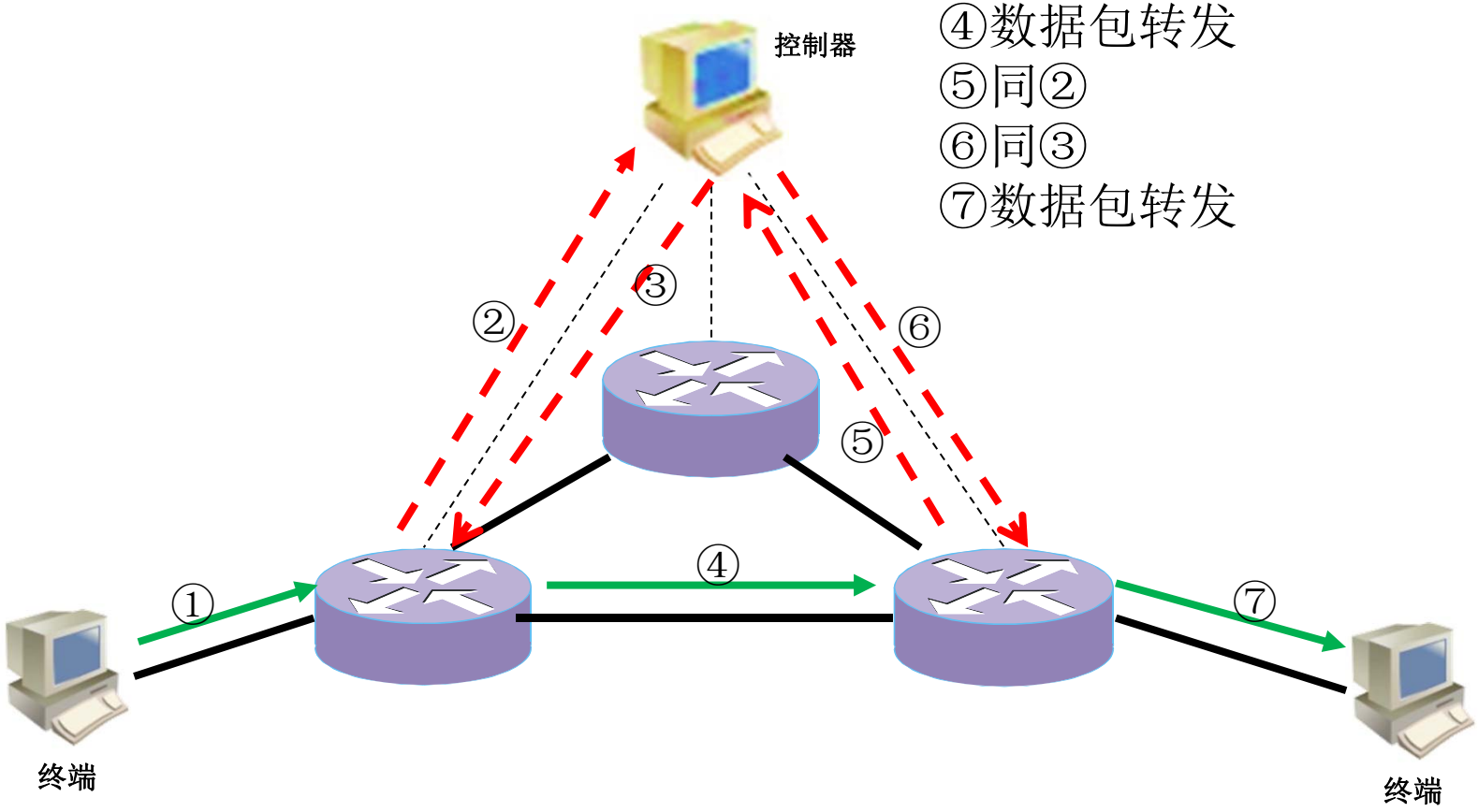
1、指定某一个数据包的处理方法
2、让交换机产生一个数据包并按照action列表处理

典型应用：链路发现
控制器向一个交换机发送Packet-Out消息，buffer_id=-1，data段为某种特殊数据包，actions为从交换机的某个端口转发
如果发出这个数据包的端口另一端也连接一个Openflow交换机，对端的交换机会产生一个Packet-In消息将这个特殊的数据包上交给控制器，从而控制器探测到一条链路的存在



基于Openflow的SDN工作流程

①主机向网络发送数据包
②OF交换机流表无匹配项，通过PacketIn事件将数据包上报给控制器
③控制器下发流表(或PacketOut)
④数据包转发
⑤同②
⑥同③
⑦数据包转发

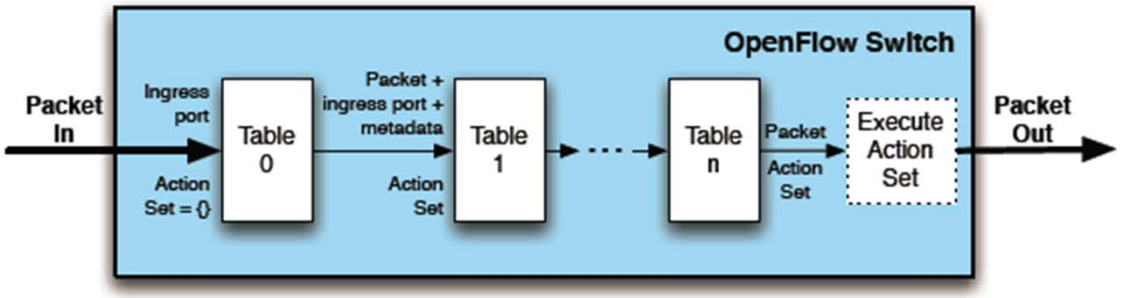


Openflow1.3介绍

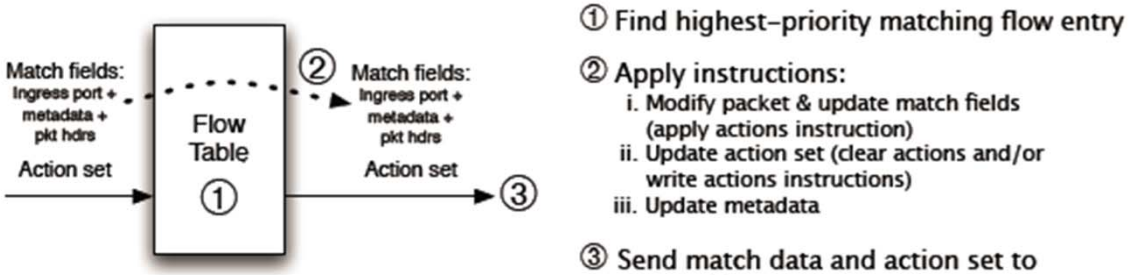
Openflow1.3与Openflow1.0的主要不同：

- 增加多级流表
- 增加了组表
- 增加了Meter
- 修改了数据包特征匹配的描述方法（match方法）
- 增加了数据包处理的动作类型

多级流表：



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



(b) Per-table packet processing

流表项：

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

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

Openflow1.0中的Actions变成了Instructrions

组表：

Group Identifier	Group Type	Counters	Action Buckets

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

Group Identifier：一个32bit标识符
Group Type：组表条目类型
Counters：组表条目的使用计数
Action Buckets：一个Action列表的有序表（一系列Action列表的集合）

Group Type：
1、all：执行action buckets中的所有动作，可以用于组播
2、select：随机执行action buckets中的一个动作，可以用于多径
3、indirect：只包含一个Action列表的组表，效率更高，可以用于路由聚合
4、fast failover：执行action buckets中的第一条生存条目，可以用于快速故障恢复

Meter：

Meter用来定义Openflow交换机对数据包转发的性能参数

Meter Identifier	Meter Bands	Counters

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

Band Type	Rate	Counters	Type specific arguments

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

- *Optional:* **drop**: drop (discard) the packet. Can be used to define a rate limiter band.
- *Optional:* **dscp remark**: increase the drop precedence of the DSCP field in the IP header of the packet. Can be used to define a simple DiffServ policer.

Openflow1.3数据包匹配方法：

```
struct ofp_match {
    uint16_t type;              /* One of OFPMT_* */
    uint16_t length;            /* Length of ofp_match (excluding padding) */
    /* Followed by:
     *   - Exactly (length - 4) (possibly 0) bytes containing OXM TLVs, then
     *   - Exactly ((length + 7)/8*8 - length) (between 0 and 7) bytes of
     *     all-zero bytes
     * In summary, ofp_match is padded as needed, to make its overall size
     * a multiple of 8, to preserve alignement in structures using it.
     */
    uint8_t oxm_fields[0];      /* 0 or more OXM match fields */
    uint8_t pad[4];             /* Zero bytes - see above for sizing */
};
```

Match结构由Openflow1.0的定长结构变为Openflow1.3的变长结构

Type有以下两种，OFPMT_STANDARD和OFPMT_OXM。其中OFPMT_STANDARD被废弃了

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

OXM_FIELDS

OXM=Openflow eXtensible Match

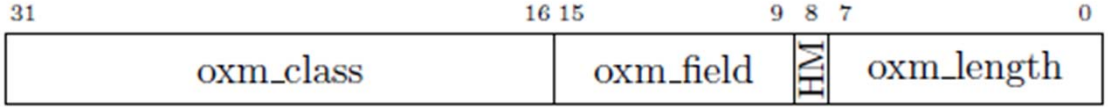


Figure 4: OXM TLV header layout.

Name		Width	Usage
oxm_type	oxm_class	16	Match class: member class or reserved class
	oxm_field	7	Match field within the class
	oxm_hasmask	1	Set if OXM include a bitmask in payload
	oxm_length	8	Length of OXM payload

Table 9: OXM TLV header fields.

```
enum ofp_oxm_class {
    OFPXMC_NXM_0          = 0x0000,    /* Backward compatibility with NXM */
    OFPXMC_NXM_1          = 0x0001,    /* Backward compatibility with NXM */
    OFPXMC_OPENFLOW_BASIC = 0x8000,    /* Basic class for OpenFlow */
    OFPXMC_EXPERIMENTER   = 0xFFFF,    /* Experimenter class */
};
```

HM表示OXM TLV中是否含有掩码项

openflow_basic_class定义的OXM_fields

```
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 */
};
```

Openflow1.3定义的必备匹配项

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

Table 11: Required match fields.

Instructions：

Meter meter_id:将数据包交给指定的meter限制
Apply-Actions actions：立即执行指定的动作，但不清除Action Set
Clear-Actions：清除Action Set中的所有动作
Write-Actions actions：向Action Set写入操作
Write-Metadata metadata/mask：写metadata
Goto-Table next-table-id：跳转到下一个流表

Action Set：

1. **copy TTL inwards**: apply copy TTL inward actions to the packet
2. **pop**: apply all tag pop actions to the packet
3. **push-MPLS**: apply MPLS tag push action to the packet
4. **push-PBB**: apply PBB tag push action to the packet
5. **push-VLAN**: apply VLAN tag push action to the packet
6. **copy TTL outwards**: apply copy TTL outwards action to the packet
7. **decrement TTL**: apply decrement TTL action to the packet
8. **set**: apply all set-field actions to the packet
9. **qos**: apply all QoS actions, such as set_queue to the packet
10. **group**: if a group action is specified, apply the actions of the relevant group bucket(s) in the order specified by this list
11. **output**: if no group action is specified, forward the packet on the port specified by the output action

谢谢