SDN

SDN的来龙去脉

网线+网卡+协议栈三要素，是组成最小单元网格的基础，缺一不可。

网线提供物理介质，承载比特流/电信号（类似电话线承载语音流/模拟信号）。

网卡进行数据处理。例如吧磁盘上的数据转化成网线上的比特，或者反过来。

协议栈作为沟通语言，实现通信过程中的数据解析，地址寻址，流控制。

集线器是一种多口中继器，属于物理层设备，对信息进行中继和方法，从任意接口收到的数据回想其他接口泛洪。

集线器不识别数据包的寻址信息和上层内容，无法对终端主机进行隔离，多个主机处于同一个冲突域，采用集线器的网络，带宽利用率低。

网桥是一种链路层设备，记录终端主机的MAC地址并且生成MAC表，网桥根据表转发主机之间的数据流。可以隔离冲突域，提高网络带宽利用率，不同接口之间的数据不会相互冲突。

接口比较有限，默认是两个，对网络隔离比较有限，没有专门的硬件，采用CPU处理，速度较慢。

交换机是一种链路层设备，效果同上。

相对于网桥：

接口数量更加密集（每个主机在独立的冲突域，带宽利用率提高）

采用专用的ASIC硬件芯片进行高速转发。

可以进行VLAN隔离（通过VLAN隔离广播域）

交换机是一种局域网产品，历史上有很多局域网技术，目前以太网称霸，一般用于本地网络，无法实现局域网互联。

路由器是一种网络层产品，基于IP寻址，采用路由表实现数据转发。

路由器主要用于连接不同的局域网，实现广播域隔离，也有可以用于远程通信（广域网连接）。

路由器的诞生是因为网络大爆炸，跨介质跨地理的网络大融合成为现实。

防火墙是一种网络安全产品，用于对网络进行安全访问限制，一般用在互联网边缘。

根据技术特征分为：包过滤，应用代理，状态检测防火墙。

可以看作有安全功能的防火墙。

是否有了防火墙就不用路由器？网络边界的路由器可以直接有防火墙替代，但是防火墙和路由器是可以共存的，路由器侧重地址翻转和路由转发，防火墙侧重安全隔离。

SDN的技术原理

SDN是神什么‘

软件定义的网路

SOW VS 传统网络

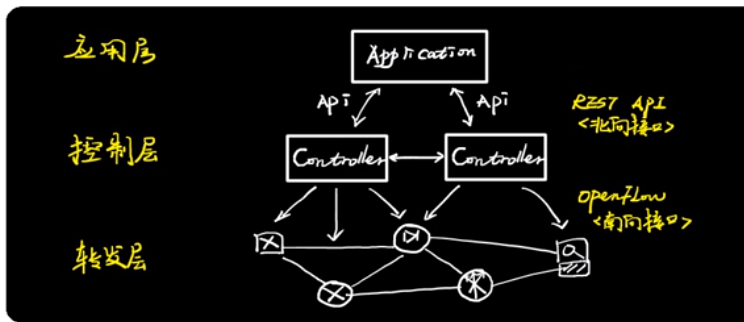
控制转发分离 控制转发耦合

集中式控制 分布式控制

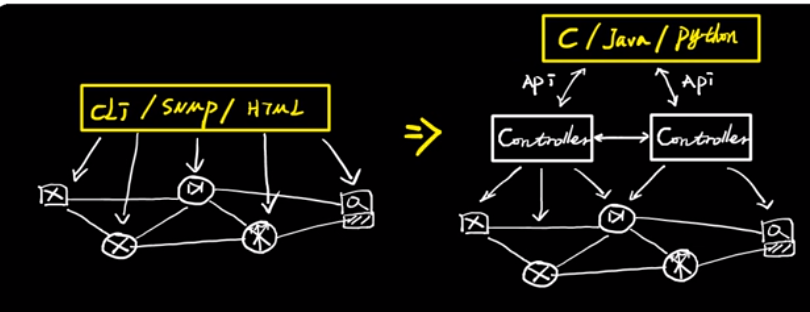
可编程 不可编程

开放接口 不开放

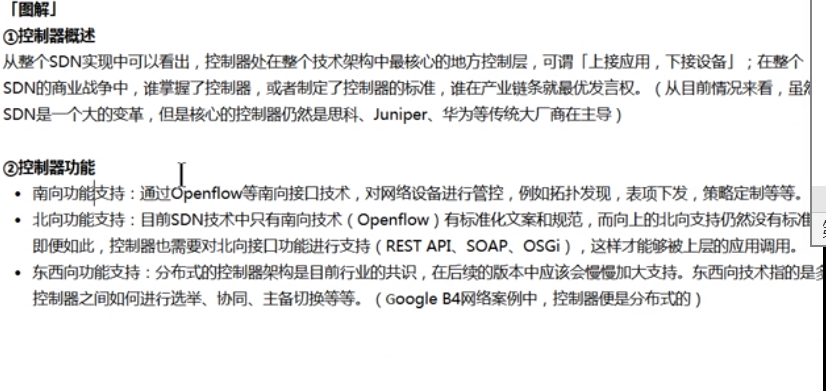
虚拟化 硬件化



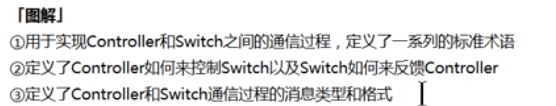
SDN的框架主要有应用层、控制层、转发层（基础设施层）组成，应用层提供应用和服务（网管、安全、流控），控制层提供统一管理和控制（协议计算、策略下发、链路信息收集）、转发曾提供硬件设备（交换机、路由器、防火墙等）进行数据转发。



Controller控制器

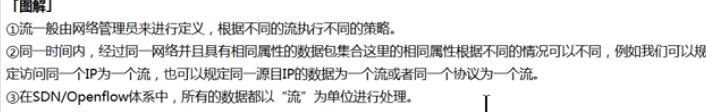


Openflow 协议

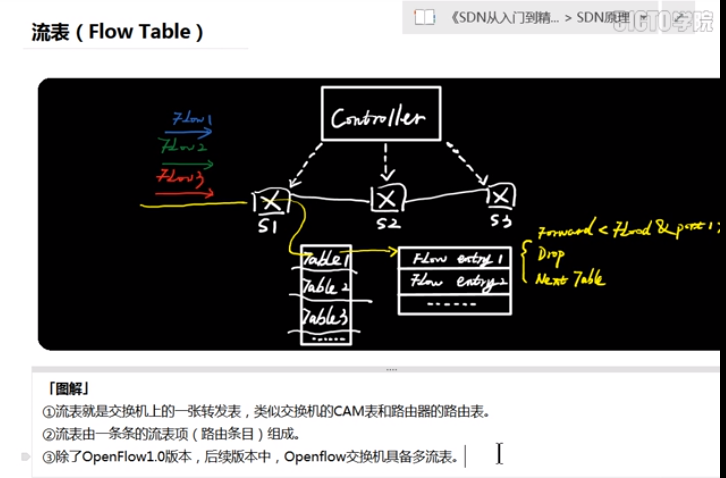


流表（flow table）

什么是流

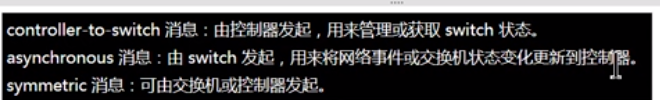


什么是流表



什么是流表项

信道



1. **SDN提出的背景及优势；**

SDN是软件定义网络。

提出的背景：大物移云的时代已经到来，传统的底层网络架构已经无法满足人类的需求，设备繁杂配置麻烦迭代缓慢，各种问题层出不穷。下一代网络，需要可编程按需求定制，集中式统一管理，动态流量监管，自动化部署等。这是SDN的出发点。

优势：与传统网络相比它是控制转发分离的，集中式控制，可编程并且开放接口，虚拟化网络。

a，SDN的设计思想：

* 解耦：将控制平面和管理平面提取出来，在转发设备上只留下数据平面；实现了控制平面和数据平面的解耦，通过控制器管理其所属的转发设备，这些转发设备没有路由器，交换机之分。
* 抽象：将SDN架构按照计算机模型抽象成SDN模型，OpenFlow交换机模型，NOS（网络操作系统）操纵下的全局网络视图。
* 可编程：除了传统的命令行接口和网关协议，还可以通过Shell脚本，Python脚本对整个网络进行管理。

　　控制逻辑和数据转发分离和网络可编程是SDN最为显见的优势。

传统网络的缺点：

a，传统网络及其设备的只可配置，不可编程，只能按照已定义好的协议处理或转发数据，不能适应需求新变化，不能自主开发新功能。

　　　　如购买一个电饭煲，可以煮饭，煲汤。但是我突然想煮玉米，这就不行了。

 b，网络的分布式控制与管理架构带来的制约。如果在网络中新增加一个设备，其流经链路上的网络设备在规划和配置上可能都要发生变化。

　　　　网络的部署，配置与管理需要落到每台设备上去手工完成，每个设备都紧耦合了三个层面，分别是管理平面，控制平面，数据平面。

* 管理平面：配置和管理网络设备提供用户访问界面或窗口（命令行或图形界面）。
* 控制平面：路由表，MAC表和MPLS标签表等控制表是数据平面数据转发的依据。
* 数据平面：根据控制平面相关控制表给出的信息，进行具体的报文处理或转发。

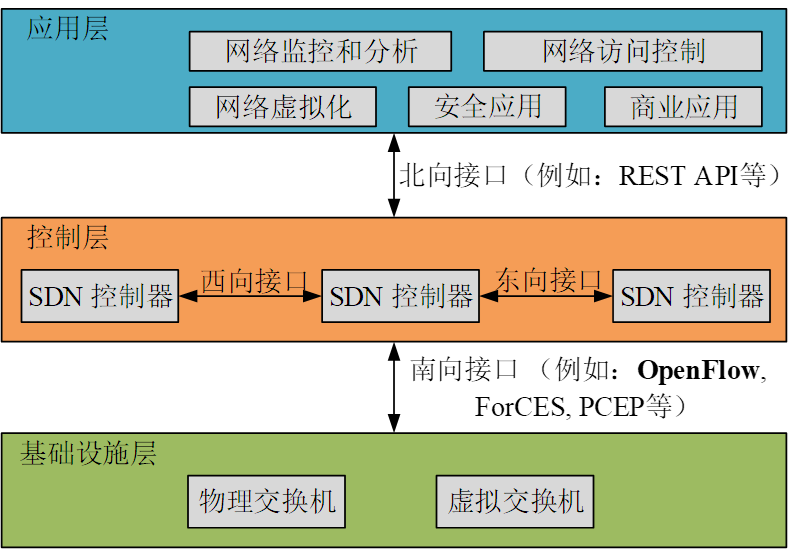
1. **SDN分为哪三层？SDN中的东西南北向接口分别是哪些？**

SDN的框架主要有应用层、控制层、转发层（基础设施层）组成，应用层提供应用和服务（网管、安全、流控），控制层提供统一管理和控制（协议计算、策略下发、链路信息收集）、转发曾提供硬件设备（交换机、路由器、防火墙等）进行数据转发。

SDN三层架构：SDN将传统转发设备中的控制平面提取成控制器，将管理平面提取成各类应用

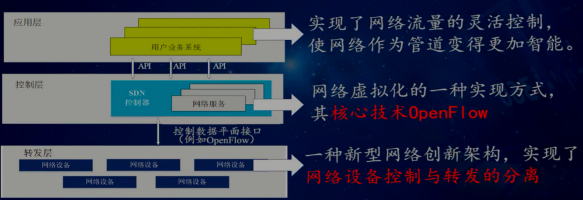
三层架构是如何运作的：如果我只想让基础设施层的设备实现集线器的功能，我可以在应用层编写一个Python脚本，交给控制器，每当基础设备有数据不知道怎么处理时，它会把这个包交给控制器处理，控制器执行脚本后将会回复这个基础设备应该向哪里转发，似乎就是一个数据包经过了一个集线器一样。

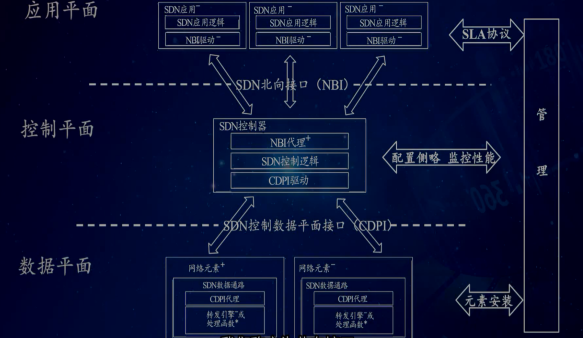
　　所以我们可以在应用层编写不同的脚本去自定义数据报的处理方法。



OpenFlow三层架构是ONF定义的SDN主流技术架构，对转发面进行了标准化，控制层与转发层采用OpenFlow协议。

　　OpenFlow架构如下：





~~四大平面：

* 应用平面：主要处理负载均衡，访问控制，应用加速等问题
* 控制平面：核心是控制器（也叫网络操作系统NOS），主要将SDN应用层请求转换到SDN Datapath（如上面所说的Python脚本）和为SDN应用提供底层网络抽象模型（通过不断与底层交换机的交互保持实时更新）
* 数据平面：网元和它所包含的SND数据通路就像是传统网络中，二级交换级和MAC表。区别在于MAC表是二级交换机自己生成的，而SDN数据通路是控制器下发的
* 管理平面：负责静态工作，如网元初始化配置，指定控制器和定义控制器应用范围等

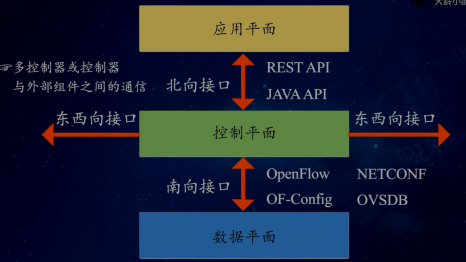
 　　~~两大接口：

* 北向接口：向应用层提供抽象的网络视图，使应用能直接控制网络的行为；

　　　　　　　　控制器将网络能力封装后开放接口，供上层应用调用。

　　　　　　　　直到目前，还没有一个标准的北向接口协议，REST API成为SDN北向接口的主流设计。

* 南向接口：实现转发行为控制，设备性能查询，同级报告，事件通知等功能。最流行的南向接口协议是OpenFlow协议。下图是控制器通过OpenFlow1.0协议和交换机交互



因为控制器为了提高处理效率和减轻控制器的压力，类似于DNS服务器，采取的分布式布局（如Google B4网络架构）。借助东西接口，可以选举和协同控制器，切换主备控制器等。

 　　~~ OpenFlow架构需要理解的三个组成部分：

* 流表：指示交换机如和进行流的处理。
* 安全通道：负责控制器与交换机之间的交互，也就是走一些相关的OpenFlow协议。
* OpenFlow协议：定义了一种南向接口标准。

（1）开放控制层北向接口的SDN（可以简称为SDN-N）

这种思路研究数据网络开放控制面与业务应用之间的北向接口，向上提供资源抽象，实现软件可编程控制的网络架构。数据网络中控制层北向接口的开放有利于互联网应用服务感知数据网络状态、优化业务应用设计、改善用户业务体验，因此得到了互联网服务提供商的支持。北向接口开放性研究发端于5年前的P2P研究热潮，为了实现P2P流量优化与数据网络流量调度之间的协调，IETF启动了ALTO、DECADE等多个工作组，随着P2P热度的消退，这些工作的研究进展缓慢，但是SDN的升温为这个研究方向注入了新的活力。但是研究北向接口的开放性，主要是要抽象不同业务应用的共性特征，及其对数据网络的承载需求，但是业务应用的多样性使得这项工作目前进展并不顺利。

（2）开放控制层南向接口的SDN（可以简称为SDN-S）

这种思路就是大家通常理解的SDN，即数据网络中控制平面与数据平面的分离。目前比较热的ONF的Openflow协议和IETF的Forces协议都是工作在这个层面的，都是定义控制平面与数据平面分离后，两者之间的通信协议。Openflow与Forces协议的不同点在于：Openflow所面对的转发设备硬件假设只支持十元组，Openflow可以针对十元组做各种转发规则的配置；而Forces假定所面对的转发设备硬件是协议无关的，Forces可以以XML语言的格式来任意定义底层转发设备的处理逻辑。协议无关的转发设备目前也成为了研究的热点，要做到协议无关，需要硬件具备众多的功能，看似十分困难的工作，但是一些芯片厂商和设备厂商已经研发出了协议无关的转发产品，是一个值得关注的方向。

（3）开放控制层东西南北向接口的SDN（可以简称为SDN-SE）

在开放了南北向接口以后，SDN发展中面临的一个问题就是控制平面的扩展性问题，也就是多个设备的控制平面之间如何协同工作，这涉及到SDN中控制平面的东西向接口的定义问题。如果没有定义东西向接口，那么SDN充其量只是一个数据设备内部的优化技术，不同SDN设备之间还是要还原为IP路由协议进行互联，其对网络架构创新的影响力就十分有限。如果能够定义标准的控制平面的东西向接口，就可以实现SDN设备“组大网”，使得SDN技术走出IDC内部和数据设备内部，成为一种有革命性影响的网络架构。目前对于SDN东西向接口的研究还刚刚起步，IETF和ITU均未涉及这个研究领域。

从网络开放性的发展趋势来看，SDN概念对于网络设备和网络架构设计的影响还处于初级阶段，以后随着SDN中控制平面北向和东西向接口的标准化，以及SDN技术与网络虚拟化技术的融合，将使SDN技术释放出更大活力和更为深远的影响力。

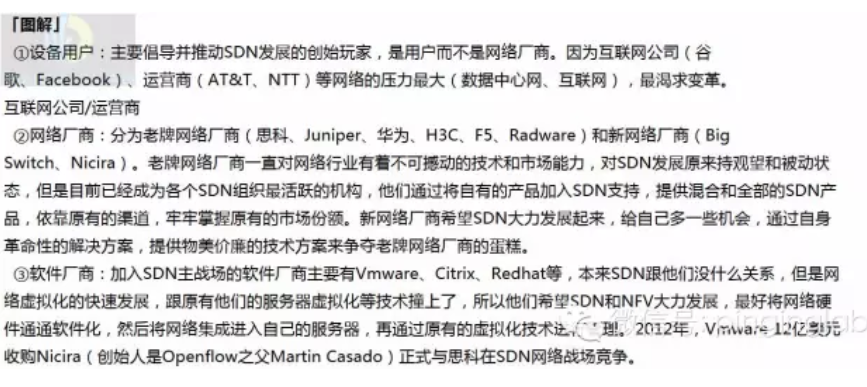
**3. 推动SDN的组织有哪些？ 与早期计算机网络的组织相比，有哪些变化？**

**ONF：**开放网络联盟，是最早的SDN组织，创始成员是Goole,Facebook，微软，雅虎等互联网成员，旨在推动SDN标准化，制定并且发布Openflow技术的标准。

**ODL：**成员主要是思科 华为等网络厂商，旨在提供开源的网络操作系统

**ONOS**：开放网络操作系统。创始成员主要来自运营商，旨在为运营商用户提供一个开源的网络控制器。

**NFV：**网络功能虚拟化，成员主要来自运营商

  
**4. SDN协议与TCP/IP协议的关系是？端口号在SDN中如何使用？**

1 TCP/IP是Internet环境下的五层标准协议栈，

2 SDN是一种思想，即网络的自动化运维，表现为数据和控制分离，所以它不是一种协议或者别的什么东西

3 openflow是实现SDN一种实现形式，就openflow本身，是一种网络控制器和网络设备间交换信息和下发命令的协议，它主要是为了实现二层交换和三层路由，它是一种集中式的路由控制协议。与之可比较的分布式的BGP或OSPF等协议

4 即便openflow在控制层面改变了以往分布式的路由协议，但是数据层面网络设备上经过的还是普通的TCP/IP协议栈的数据包，如ICMP、TCP等

端口：

OpenFlow Ports是OpenFlow Switch与剩余网络之间传递Packet的网络接口。OpenFlow Switches之间通过OpenFlow Ports彼此相互逻辑连接。一个OpenFlow Switch准备了若干个可用的OpenFlow Ports用于OpenFlow processing，不过这里所说的OpenFlow Ports与传统的硬件交换机的网络接口并不是一样的概念，有些传统硬件交换机的网络接口在OpenFlow Switch里并不适用，OpenFlow Switch也可以定义自己额外附加的OpenFlow Ports。

　　Packets通过 **ingress port** 被接收到OpenFlow Switch中，被Pipeline处理后，其或许会被转发到一个 **output port** 上。在Packet被Pipeline处理的整个流程中，这个 **ingress port** 会被作为Packet的一个属性存在，它表示了该Packet是从哪个OpenFlow Port进入到OpenFlow Switch里的，并可被用于后续的Flow Entry Matching过程。OpenFlow Pipeline可以通过使用 output action决定将该Packet转发到一个**output port**上，进而将该Packet重新送回到网络中。

　　一个OpenFlow Switch必须支持三种类型的OpenFlow Ports：physical ports（物理端口）、logical ports（逻辑端口）和reserved ports（预留端口）。具体OpenFlow Ports 描述如下：

1、Standard Ports（标准端口）：

 　　OpenFlow Standard Ports通常被定义为是Physical ports、Logical ports和支持的LOCAL Reserved ports（不包括其他的Reserved ports）。它们可被用于 ingress port，也可被用于 output port，同样可以用于 Group 里，它们都有 port counters。

2、Physical Ports（物理端口）：

　　OpenFlow Physical Ports是OpenFlow Switch定义的、对应于物理交换机的一个硬件接口的port。比如在一个以太网交换机上，Physical Port会唯一对应于一个以太网接口。但是在一些特殊的环境中，比如在基于物理机硬件交换机上虚拟出来的OpenFlow Switch，它的Physical Port可能仅仅只是对应于这个物理硬件交换机的硬件接口上的一个虚拟切片。

3、Logical Ports（逻辑端口）：

　　OpenFlow Logical Ports是OpenFlow Switch定义的端口，它并不直接对应物理交换机的硬件接口。Logical Ports是Switch中用non-OpenFlow methods（比如link aggregation groups，tunnels，loopback interfaces）定义的更高级别的抽象，它可能包括了对Packet的封装，可能会被映射到多种Physical Ports上。它们对Packet的处理对于OpenFlow processing来说一定要是透明的，而且Logical Ports必须得像Physical Ports一样与OpenFlow processing交互。

　　Logical Ports与Physical Ports唯一的区别就是与Logical Ports关联的Packets可能会有一个额外的称之为Tunnel-ID的metadata域与之关联；当一个Packet被Logical Ports接收并被发送到Remote Controller时，与其关联的Logical Port以及其潜在的Physical Port也要上报给Remote Controller。

4、Reserved ports（预留端口）：

　　OpenFlow Reserved ports是OpenFlow Switch规范定义的，它们指定了一些通用的转发动作，比如发送到Remote Controller，flooding，用non-OpenFlow methods转发（或者说是“正常的”交换机处理）。一个OpenFlow Switch并不需要实现所有的Reserved Ports，但是必须得实现下面加黑表示的Reserved Ports：

1）**ALL**：表示那些所有能被Switch用来发送特定Packets的ports，其仅仅只能被作为一个output port；在这种情况下，Packet的一个拷贝将会被发送到所有的Standard Ports，除了Packet的 ingress port以及那些配置为OFPPC\_NO\_FWD的ports。

2）**CONTROLLER**：表示与Remote Controller之间的OpenFlow Channel，可以被用作为一个 ingress port，也可以被用作为一个 output port。当被用作为一个 output port时，Packet将会被封装在 packet-in 消息体里，并被以OpenFlow Protocol的方式发送给Remote Controller；当被作为一个 ingress port时，则表明该Packet来源于Remote Controller。

3）**TABLE**：代表OpenFlow Pipeline的开始，该port仅仅只能在一个 packet-out 消息的action集合中的 action是 output 时才有效，它将Packet发给OpenFlow Pipeline的第一个Flow Table，以便开始进行常规 processing。

4）IN\_PORT：代表Packet的 ingress port，仅仅被用作为一个 output port，将Packet通过它的 ingress port发送出去。

5）**ANY**：特殊的值，用在没有指定port（或者port是通配的）的特殊OpenFlow Commands情况下；它即不能作为一个 ingress port，也不能作为一个 output port。

6）LOCAL：表示Switch本地网络栈和管理栈，可以被用作是一个 ingress port 或者是 output port。Local Ports能够使远程实体通过OpenFlow网络与Switch及其它的网络服务交互，而不用通过一个独立的控制网络。通过合适的默认Flow Entries集合，它可以被用来实现一个 in-band controller连接。

7）NORMAL：表示Switch传统的non-OpenFlow Pipeline，仅仅只能用作一个output port，且仅仅只能使用Normal Pipeline来处理Packets。如果该Switch并不能将来自于OpenFlow Pipeline的Packets转发给Normal Pipeline，那么必须得表明其不支持该action。

8）FLOOD：表示使用Switch Normal Pipeline的flooding，仅仅只能被用作为一个output port，通常通过它可以将Packets发到所有的Standard Ports上，但除了接收这个Packet的 ingress port 以及那些状态为OFPPS\_BLOCKED的ports；当然，Switch也可能会根据Packet的VLAN ID来选择往哪些port去flood。

OpenFlow-only Switches并不支持NORMAL和FLOOD端口，而OpenFlow-hybrid Switches可能会支持它们。将Packet转发到FLOOD port则依赖于Switch的实现和配置，而使用类型为all的组转发则能够使Remote Controller更加灵活地实现flooding。

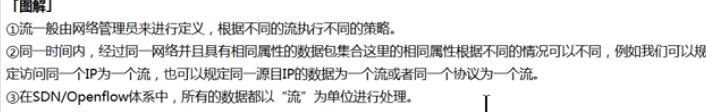
SDN的hello和OSPF的hello包的区别？

5**. Openflow协议1.0和1.3的格式是怎样的？**

<https://blog.csdn.net/zln99/article/details/50983293?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-4.edu_weight&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-4.edu_weight>

**6. Openflow中流的概念如何解释？**

SDN核心思想史数据是控制分离，在OPENFLOW引入了“流”的概念后，控制器可以依据某次通信中流的第一个数据分组的特征，使用openflow协议提供的接口对数据平面设备（openflow交换机，简称OPF交换机）部署策略——openflow称之为流表。

  
**7. SDN中交换机的流表有哪些要素？收到一个流数据时如何进行地址比对和转发？**

每个流表都有特定条目的流表项（Flow entry），每个流表项里包括动作（丢弃，转发，交给下一个流表等）。传统网络对匹配到的流只有两种处理，丢弃或转发。　　当然，流表中不会只有动作这一个流表项，以OpenFlow1.0为例，还有源目IP，MAC地址等，这些数据放在包头域中，它们决定了交换机执行怎样的动作；同时还有计数器，以监测流的动向，实现流量可视化，充分利用链路。





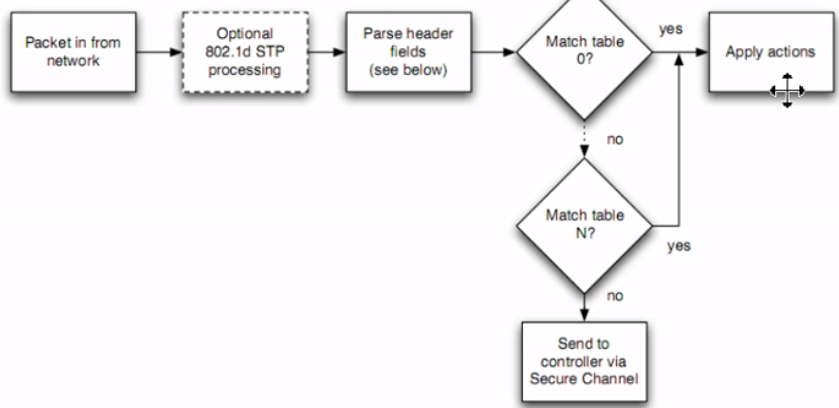
OpenFlow可以对数据包头部进行修改，这是跟传统网络最大的区别

　　到了OpenFlow1.3版本，流表项变成了

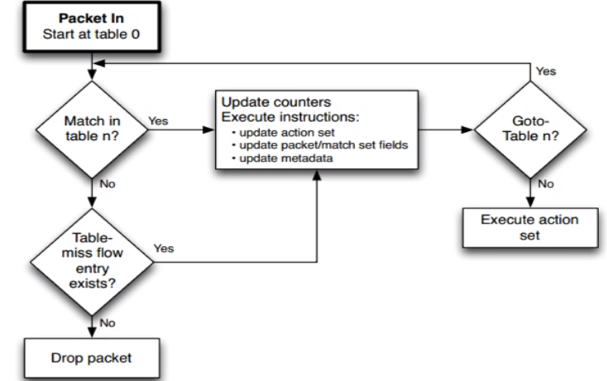
IMG_256

* 包头域改为匹配域，增加优先级，动作改为指令，增加超时时间和cookie）
* 优先级用于标志流表项匹配的优先次序，优先级越高越早匹配，默认优先级为0
* 计数器主要对每张表，每个端口，每个流等进行计数，方便流量监管（如多少流量流经这个端口，这张表查找了多少次）
* 通过指令可以生成动作集（动作集里的动作按顺序执行）

流表匹配

OpenFlow1.1版本：多流表的流表匹配称为流水线处理，交换机从流表0开始查找，序号从小到大；每个包按照流表的优先级依次去匹配流表中表项，优先级高的优先匹配，一但匹配成功，对应的计数器将立即更新；如果没能找到匹配的表项，则转发给控制器。

　　1.3版本及以后的



区别：a，之前匹配到表直接执行动作，现在可能会先丢到动作集里去

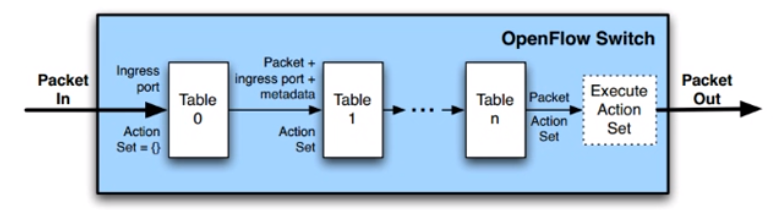
　　　b，没有表匹配，之前是交给控制器，丢弃或交给下一个流表。现在，table-miss是一个参数（即流表项只要存在这个参数，就算未匹配也

　　　　会往动作集里加动作），用于解决未匹配流的丢弃和转发问题 。table-miss是通过将匹配域的所有字段（所有网络字段都可能作为匹配域）

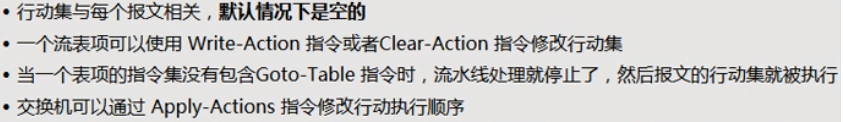
　　　　均改为通配符ANY来实现的。

　　  c，当流表项的指令集中不包含GoTo-Table时，才执行动作集里的动作。

另一种表示：



Tip：对动作集的相关操作



由于流水线匹配是按照流的特征据决定处理的方法，所以流表项都有相应的指令集。但是有些指令集对不同流都是适用的，例如：不同的流可能会有相同的下一条IP地址

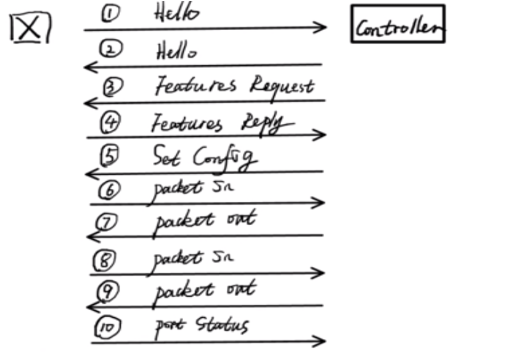
这时如果每个流表项都添加这个动作，则会在一定程度上造成资源的浪费，影响数据平面转发的效率。于是就是有了=》

 组表：OpenFlow交换机只含有一个组表，独立与流水线之外。组表中包含许多组表项，每条组表项的结构如下：

IMG_260

## 4，如何生成流表

　　首先需要了解交换机与控制器之间的交互



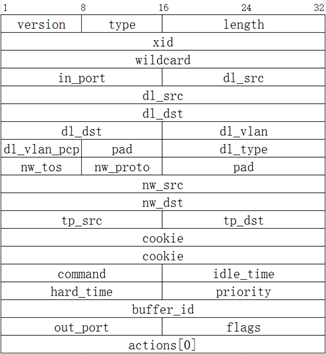
* Hello：建立连接用Hello初始化
* Features Request就是控制器问交换机有什么功能。如接口，配置，地址，宽带信息等
* set config，控制器做一个简单的适合交换机的设置 =》至此，交换机和控制器相互认识
* packet in 如果交换机缓存足够，只会将分组头信息和在缓存里的序号发给控制器；如果缓存不足，则会将整个数据分组封装进Packet-in消息发送给控制器

　　如：pc1 ping pc2，但是交换机没有流表，不知道往哪个接口发，就需要先把这个包打包好向控制器申请流表

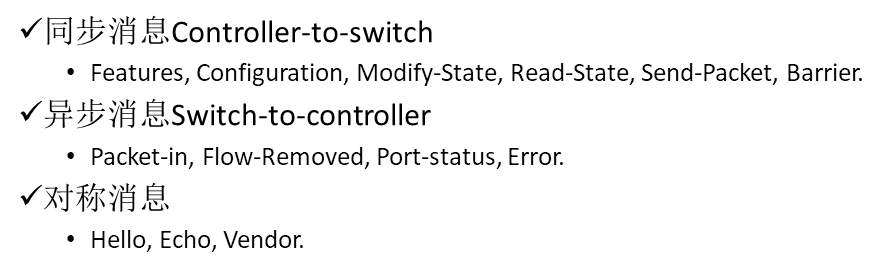
* packet out 封装好数据分组传给OF交换机
* port status 如端口被shot down掉，端口寻址到新的mac地址，维持网络视图的实时更新。

　　Flow-mod：下发流表项如（match \*，output all）广播数据包=》这样就让交换机变成了集线器。也可以删除和修改流表。

　　OpneFlow v1.0中的Flow-mod消息格式

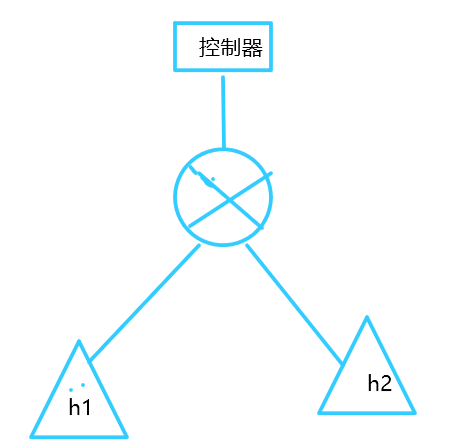


OpenFlow协议三类消息：



## 5，举个例子：

假设初始化一个最为简单SDN模型，主机h1想 ping 一下h2，会经历哪些过程？



* h1向交换机发送arp包
* 交换机将arp包封装packet-in给控制器
* 控制器packet out给交换机一个流表项，因为这个模型是刚初始化的，所以交换机里没有流表，控制器将会命令这个交换机广播这个数据包来寻找h2
* h2收到交换级转发的arp request，回复自己的mac和ip等信息
* 交换机收到h2发来的arp reply，交换机不知道怎么转发，再次packet-in
* 控制器更新，下发流表项
* 交换机从控制器所指定的端口将arp reply发送给h1

**8. 泛洪转发和广播转发的区别是什么？**

一.泛洪的产生:MAC地址表显示了主机的MAC地址与以太网交换机端口映射关系，指出数据帧去往目的主机的方向。当以太网交换机收到一个数据帧时，将收到数据帧的目的MAC地址与MAC地址表进行查找匹配。如果在MAC地址表中没有相应的匹配项，则向除接收端口外的所有端口广播该数据帧，有人将这种操作翻译为泛洪(Flooding，泛洪操作广播的是普通数据帧而不是广播帧)。在我们测试过的交换机中，有的除了能够对广播帧的转发进行限制之外，也能对泛洪这种操作进行限制。

\*交换机的老化时间设置不当,也会引起泛洪处理.因为,交换机的MAC学习也是需要一定数量的帧;重新建立MAC表也是需要时间的.(试交换机性能而定).

\*频繁地调换PC机与交换机的端口的连接也是原因之一.

注意:广播帧和组播帧是直接向所有端口转发.

二.广播帧的产生:网络中存在有广播帧是不可避免的,比如开启了DHCP服务器，每次请求，都会有以"FF.FF.FF.FF.FF.FF"的帧格式出现.它向所有端口转发.如果，局域网内的网卡有故障有时也会有广播帧出现，如果大量的这种帧出现，外在表现为网络速度变慢.

三.广播包的产生:

往往伴随着ARP而产生.

假设主机A与B在同一个网络内,当主机A要向主机B发送信息,那么需要知道主机B的IP地址和MAC地址,这里面我们假定A只知道B的IP地址,而不知道B的MAC地址,那么这时A就需要向网络中发送一个ARP请求,来获取B的MAC地址,这个ARP请求实际上就是一个广播包.

当主机A:172.168.0.1:XXXX.XXXX.XXXX向B:172.168.0.3请求MAC地直.

请求的是172.16.0.3这个IP的MAC地址，在ARP请求里，目的MAC地址是0000.0000.0000

这是一个MAC的广播地址，目的是要发送一个广播.

使本地网络内的其他主机都接收这个请求，然后除了目标方(即172.1680.3)作出回应之外，其他机均会丢弃这个请求帧.

其目的,是让172.168.0.3这台机将自己的MAC发过来.

泛洪和MAC列表相关.在L2中存在.是有确定的MAC地址的,只是在MAC表中找不到具体转发的端口和MAC的配对,才开始泛洪处理.但是泛洪并不是广播帧(FF.FF.FF.FF.FF.FF).

广播是有一个具体的行为,它的对象是整个网络.在ARP时往往需要有特定的主机来响应.当然太多的广播对于网络是有害的.容易造成广播风暴.  
**9. 目前，SDN适用的场景有哪些？**

**10. 了解SDN虚拟系统如mininet(http://mininet.org/)，ONOS等。**