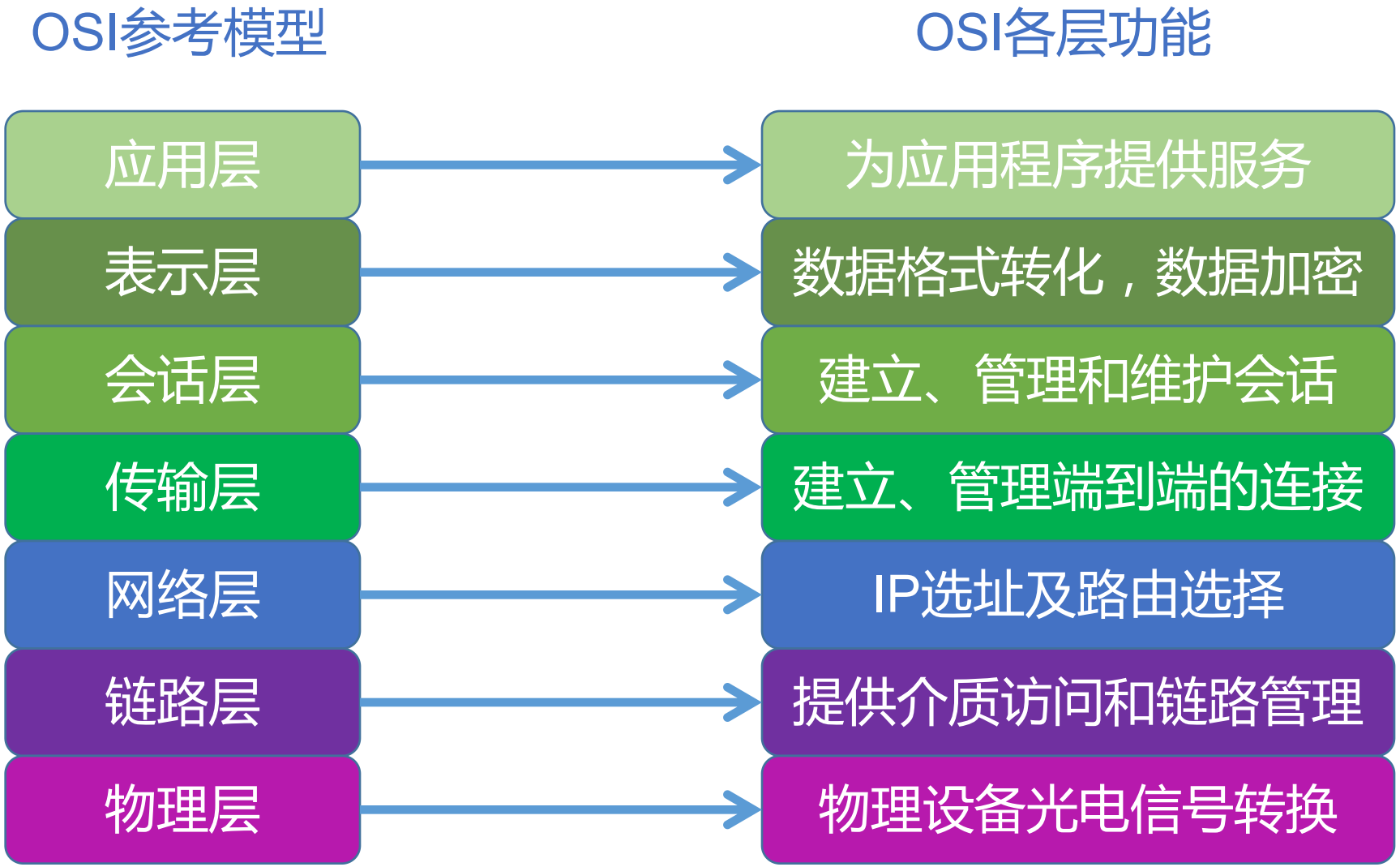
第4章　SDN数据平面

目录

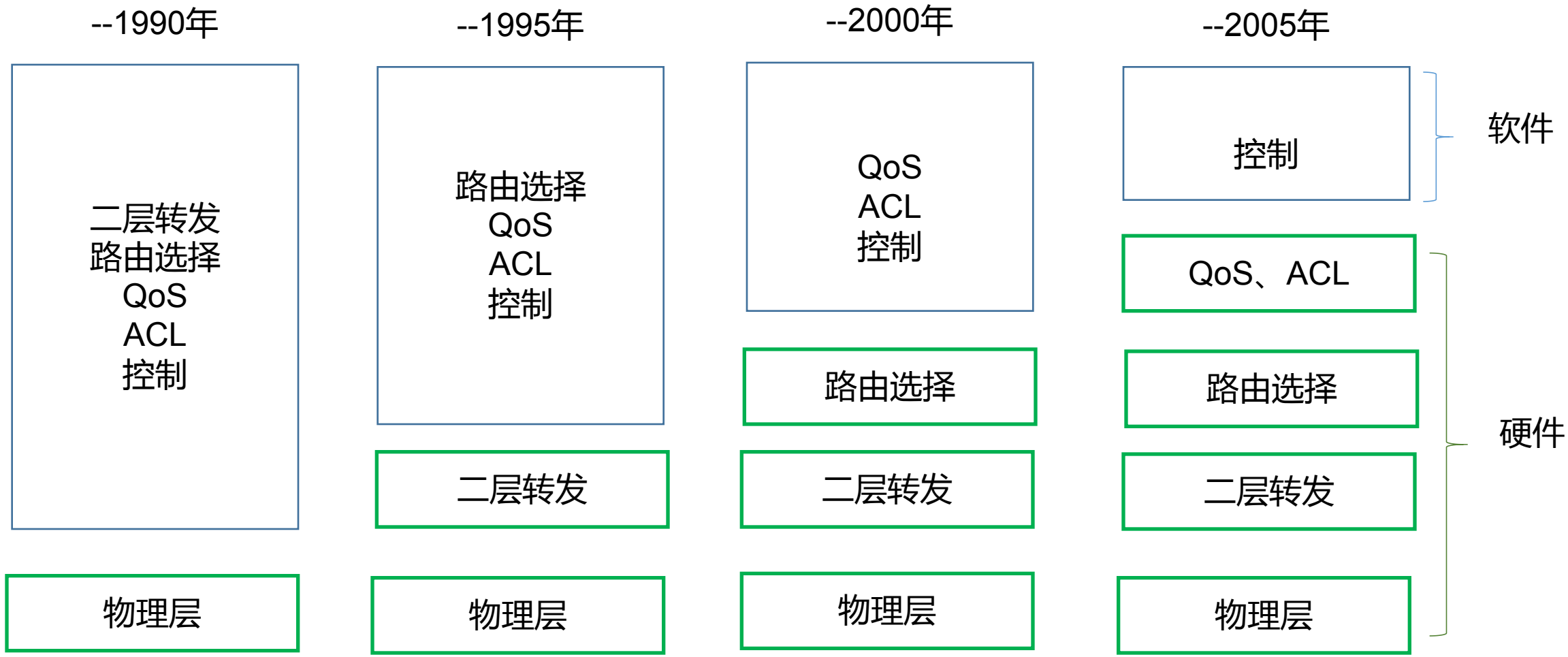
现代网络的创新与需求

- 关注点：从网络控制中心被炸时网络的保活能力逐渐演变到如何管控整个网络；
- 流量模型：数据中心的东西向流量占总流量的80%。

OSI七层模型

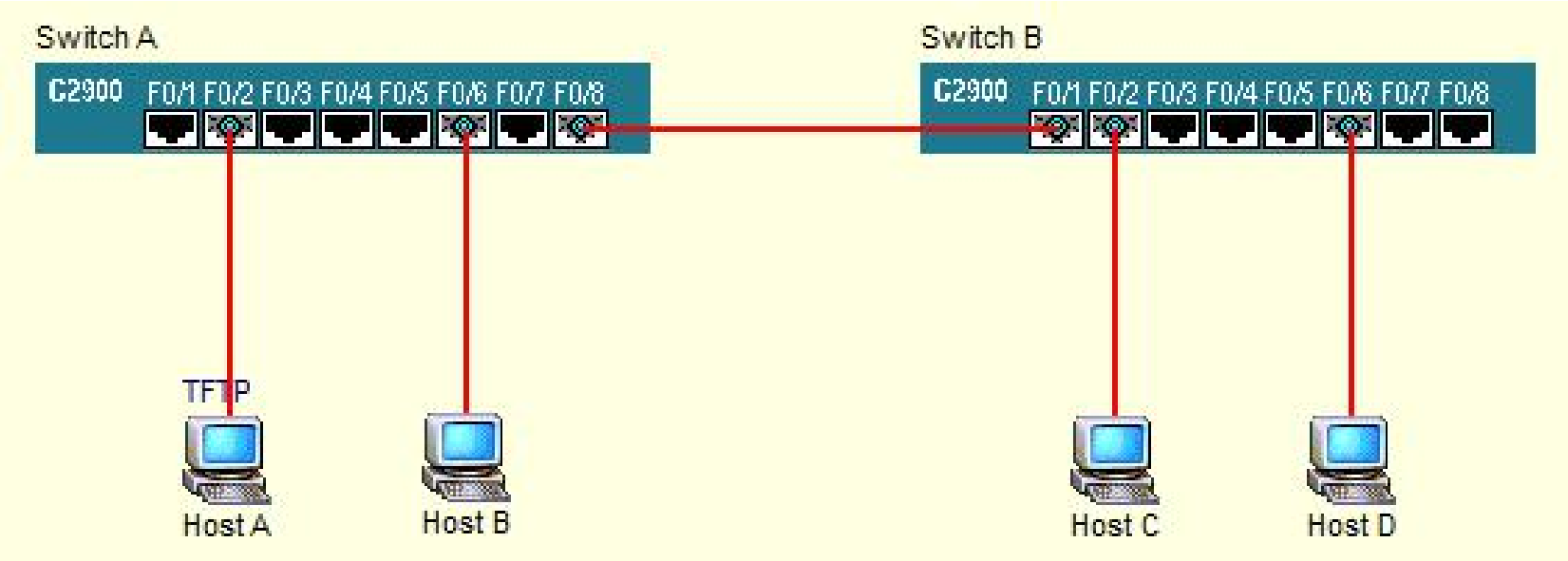

OSI参考模型
OSI各层功能
应用层
为应用程序提供服务
表示层
数据格式转化，数据加密
会话层
建立、管理和维护会话
传输层
建立、管理端到端的连接
网络层
IP选址及路由选择
链路层
提供介质访问和链路管理
物理层
物理设备光电信号转换

交换机网络功能的演进：从软件到硬件



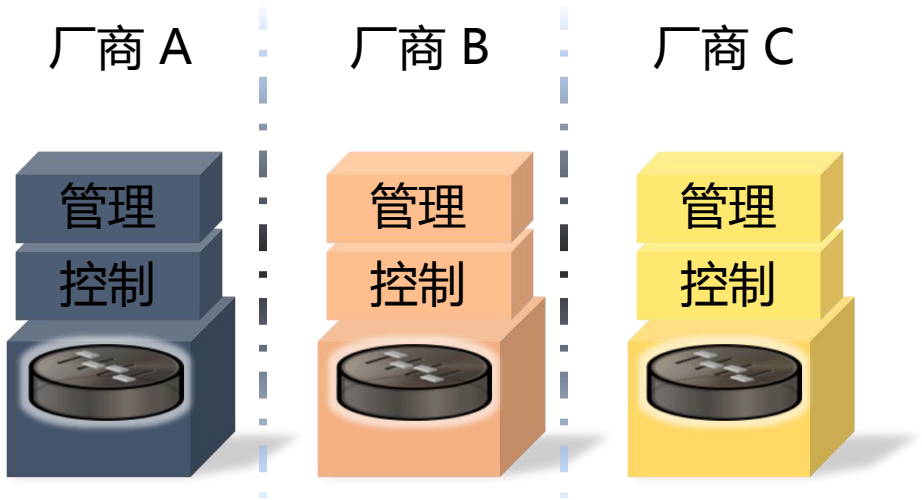
演进的背后：网络速度从1Gbps到10Gbps，再到40Gbps，ASIC\FPGA\TCAM可以以线速度执行

交换机工作流程

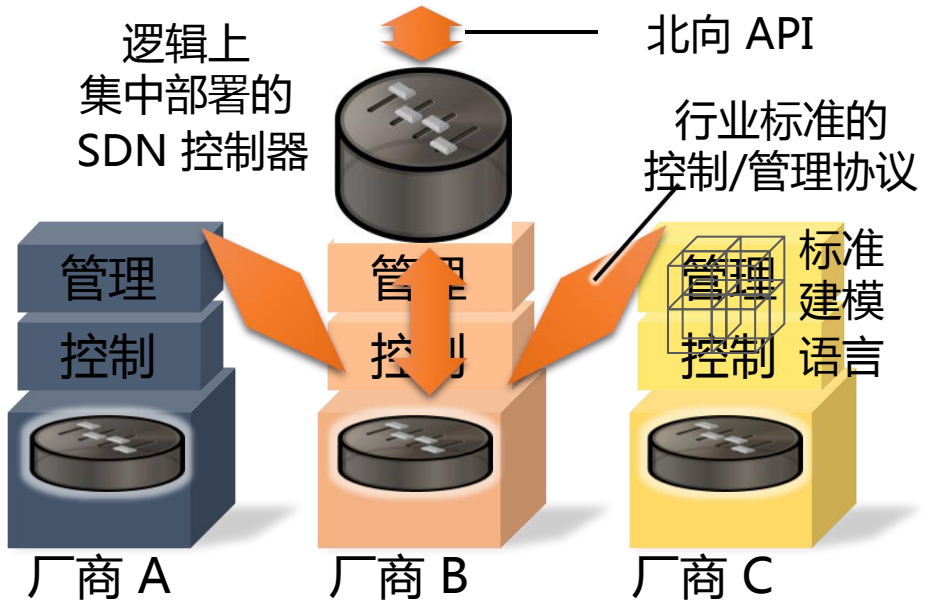


MAC	端口
…	…..
…..	…..

交换机体系结构：控制平面与数据平面的分离



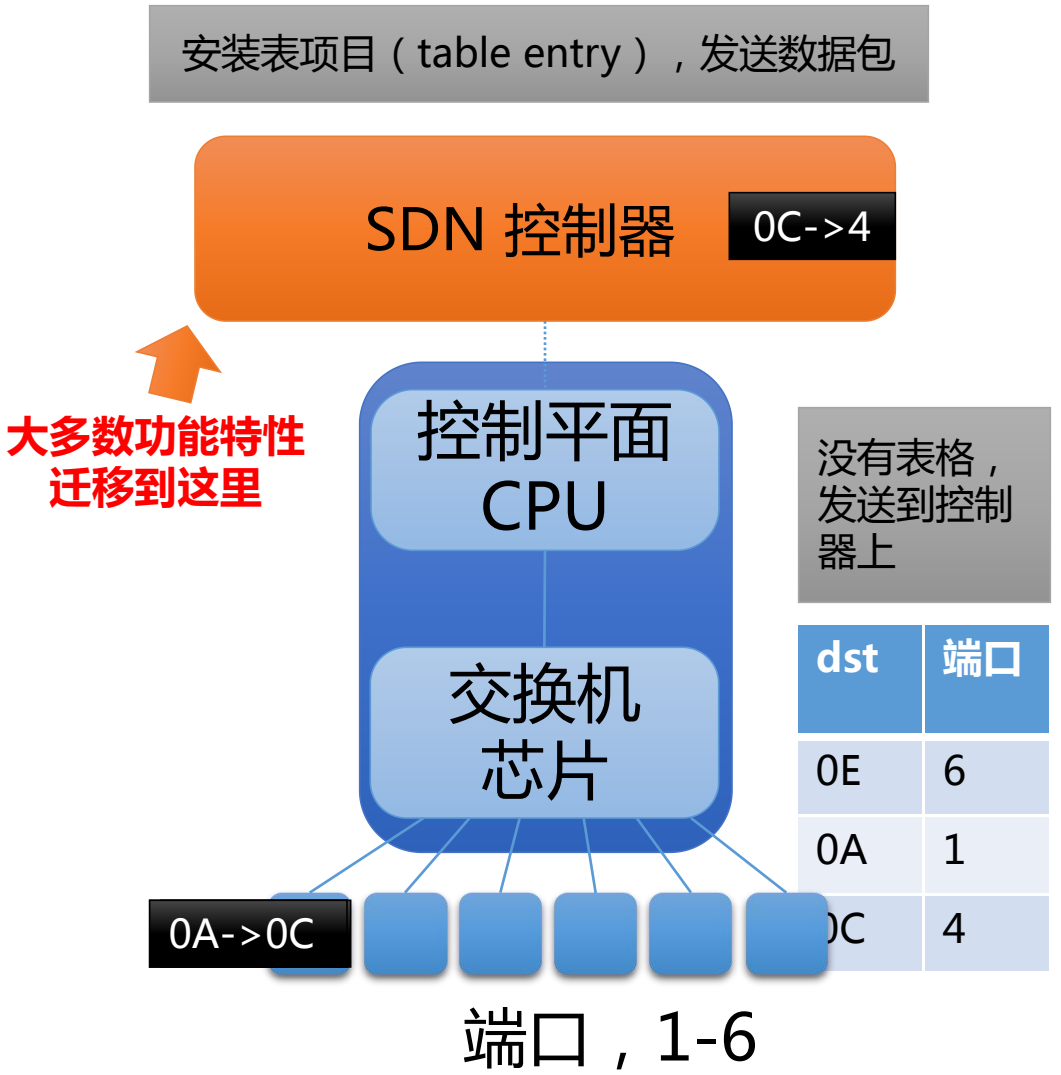
- 逐个设备操作
- 各厂商专用的垂直堆栈，用于控制、管理和编排
- 独立的孤岛，有限的创新能力



- 全网范围的操作
- 开放的控制、管理和编排，使用开放控制协议/建模语言
- 每个堆栈层上的独立创新

交换机的体系结构：控制平面与数据平面的交互

- 当今的交换机
  - 控制/数据平面均在交换机上
  - 数据平面：速度快，读取转发表项
  - 控制平面：缓慢，写入转发表项
- SDN
  - 控制/数据平面隔离开来
  - 数据平面在交换机上
  - 控制平面在别处，如 x86 服务器上，可以完成更重要的工作



交换机的芯片技术

- **ASIC**

  优缺点：专用集成电路，面向特定需求。体积小，性能强，成本低，功能固化，不支持数据平面编程。

  代表：SDN白盒交换机。

- **CPU**

  优缺点：通用集成电路，功能灵活，性能受CPU限制，效率低，支持数据平面编程。

  代表：OpenvSwitch。

- **NP**

  优缺点：网络处理器，功能灵活，性能较高，成本高，功耗高，支持数据平面编程。

  代表：华为S12700系列。

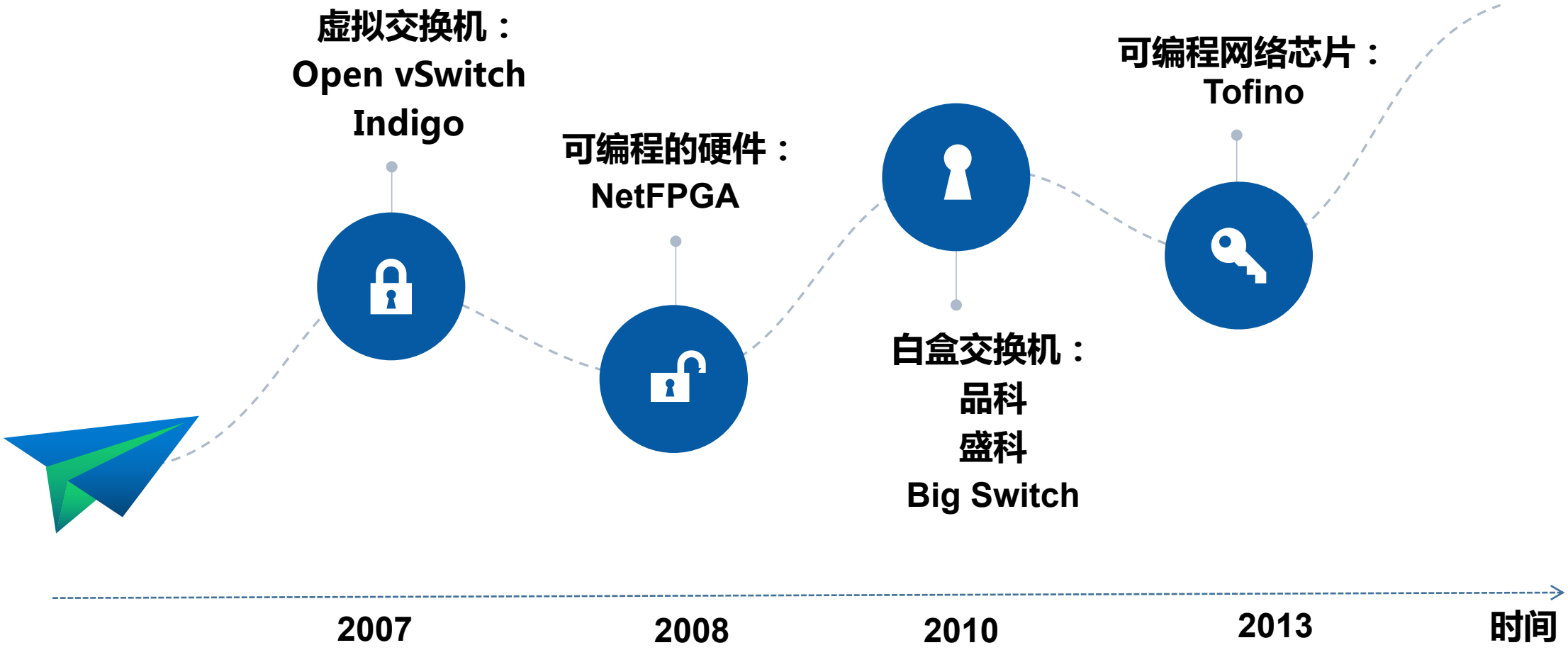
- **FPGA**

  优缺点：现场可编程门阵列功能灵活，数据转发性能低，支持数据平面编程。代表：NetFPGA。

- **PISA**

  优缺点：协议无关交换机架构，功能灵活，性能强，支持数据平面编程。代表：Tofino。

SDN交换机的演进方向



SDN交换机的软实现：Open vSwitch

Open vSwitch的提出原因：

- 网络硬件设备制造商为了成本等因素不提供对硬件进行重新编程的能力；
- 核心ASIC芯片从设计、定型到市场推广所需的超长周期，使得芯片制造商不愿意对新协议和标准轻易试水，导致硬件缺乏可编程特性。

Open vSwitch的实现：

- 面对这一两难选择，Martin Casado认为，基于x86的虚拟交换机将会弥补传统硬件交换机转发面灵活性不足这一短板。
- 2007年8月的某一天，Martin Casado提交了第一个开源虚拟机的commit，这个开源虚拟交换机在2009年五月份正式称之为Open VSwitch。

SDN交换机的实现：NetFPGA

2008年用NetFPGA实现了openflow交换机， 并在校园网进行实验， 标志着SDN在斯坦福的诞生。

Nick McKeown

Martìn Casado (TA: 2005,2006)

Photograph of a NetFPGA installed in a Desktop PC.

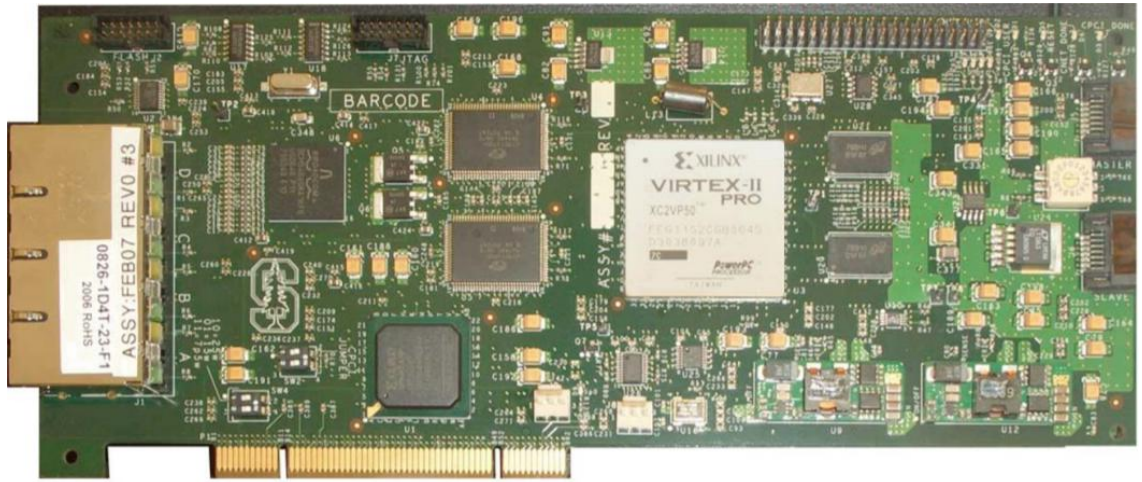
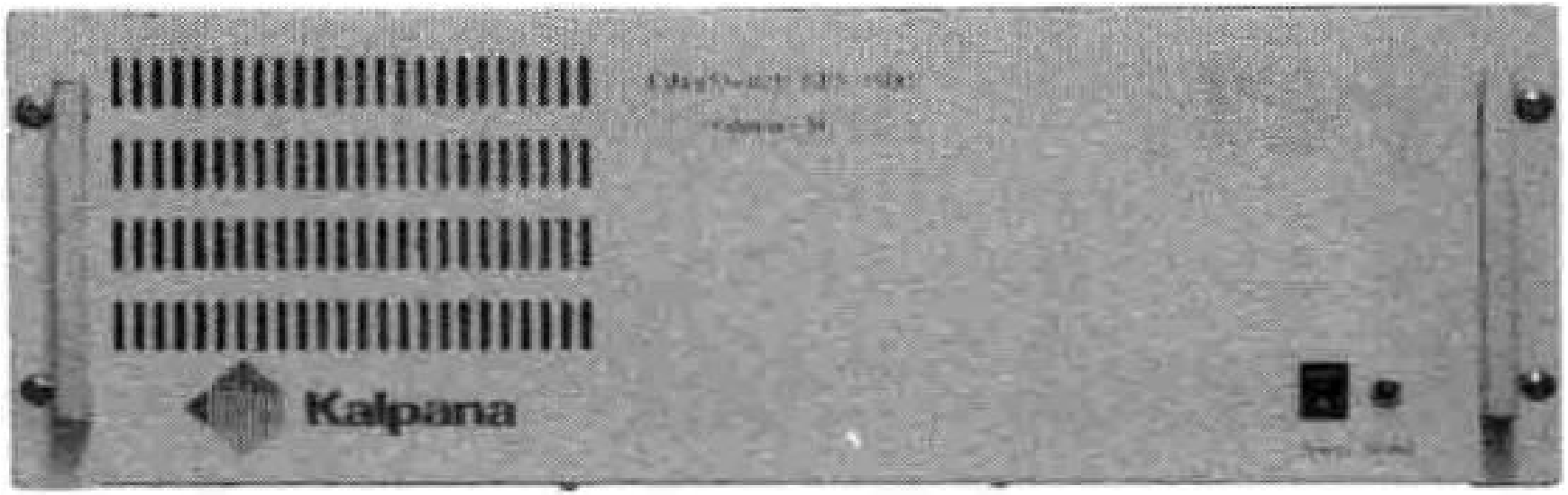


Fig. 2: Photograph of the NetFPGA Platform

目录

硬件交换机--L2 Switch

1990年，Kalpana公司发售首台交换机EtherSwitch，拥有7个端口（之前普通存储转发的网桥只有2个端口）。由于EtherSwitch没有实现IEEE规定的标准，不能称之为网桥，使用Switch一词。于1994年被思科收购。



世界上最早的 EtherSwitch

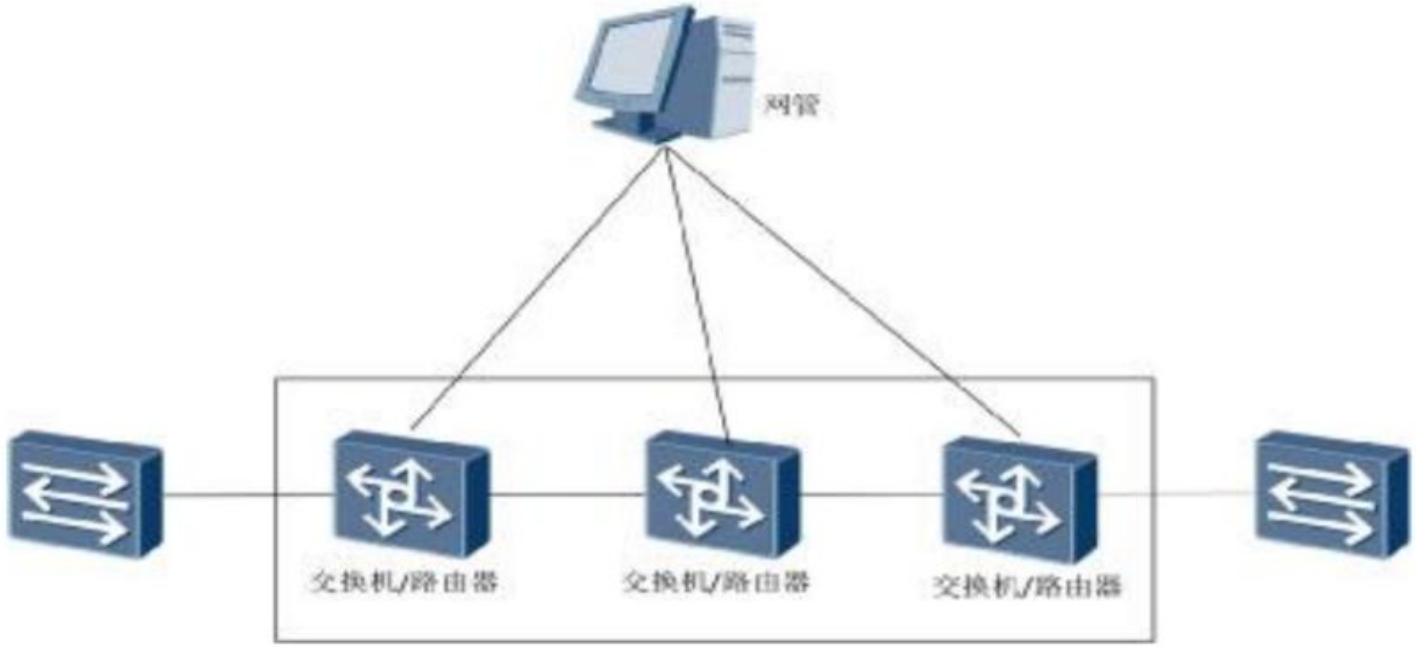
20世纪90年代，随着网络通信流量的增加，出现了更高速的网络处理需求。

同时，随着IC技术的发展，运行于CPU的L2传输控制从软件程序逐步移到ASIC和FPGA上。

硬件交换机体系结构：网络管理

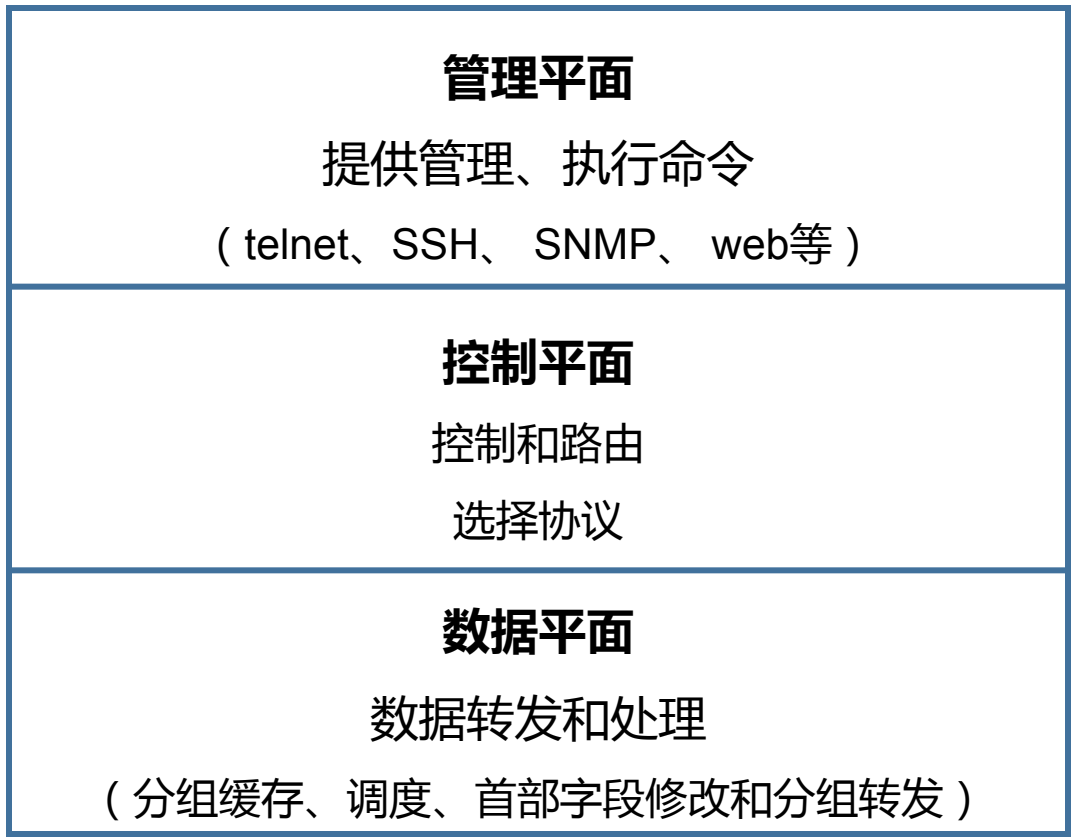
传统网络中通常布置一个集中的网管服务器作为管理面，而控制平面和数据平面是分布式的，分布在每台路由器上独立运行。当网管把网络业务配置到路由器后，如果网络状态发生变化，分布式控制面会在网络中自动扩散这些网络状态变化，然后根据新的网络状态自动重新计算路由，并刷新转发面的转发表，以确保受到影响的用户业务得以恢复。

这种分布式控制的 IP 网络架构给网络带来了健壮性，但随着通信网络发展到今天，也出现一些不可克服的局限性。



传统网络架构

硬件交换机体系结构：三平面

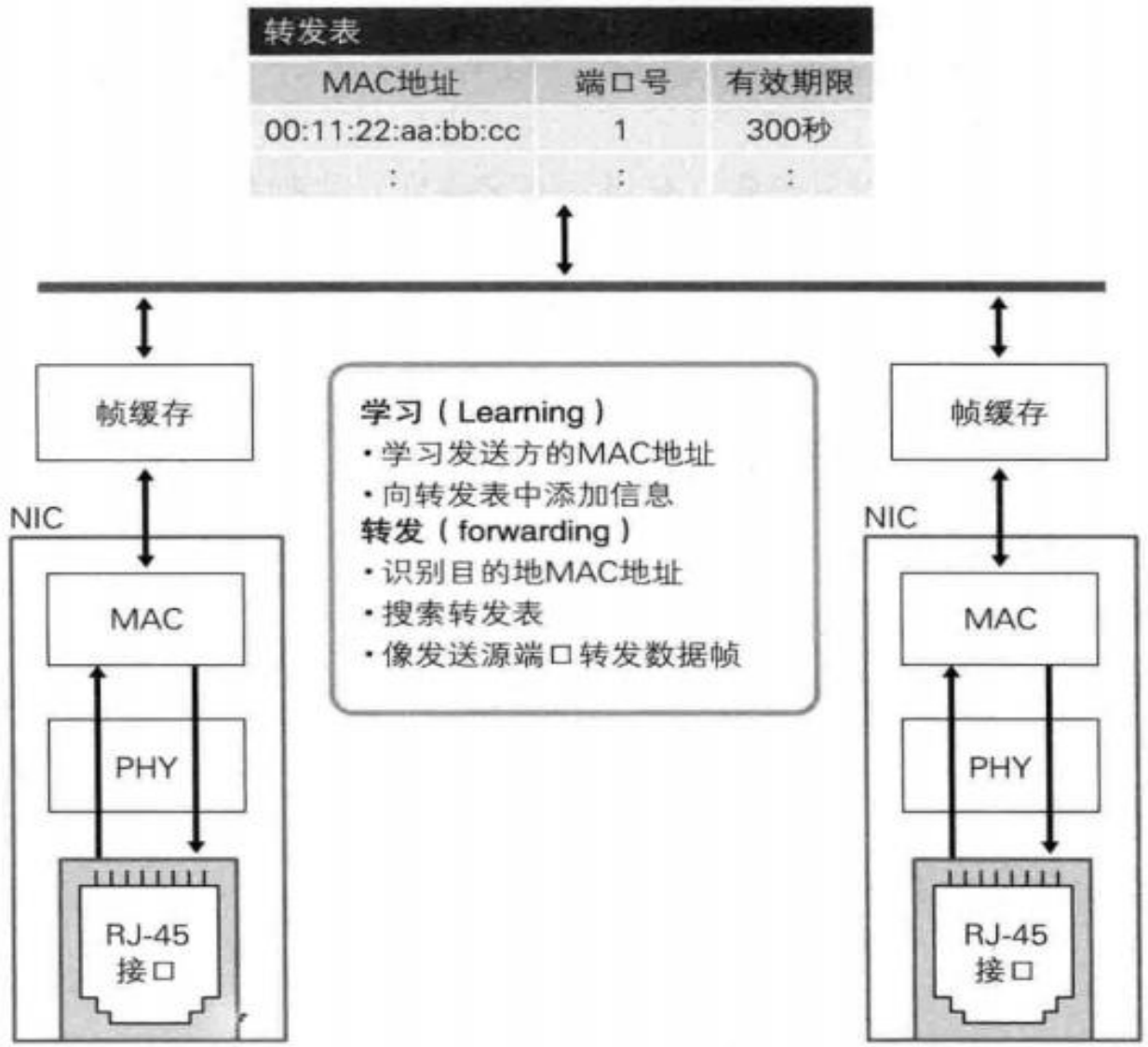


- **管理平面**
  - ①配置网络协议
  - ②查看控制平面状态
  - ③干预控制平面运行
- **控制平面**
  - ①维护网络协议运行
  - ②提供各种网络信息
  - ③提供各种转发表项
- **数据平面**
  - ①数据转发和处理

交换机通过采用数据平面、控制平面、管理平面相互分离的设计方式，保证了最耗费主机资源的数据处理转发任务不影响交换机的管理和协议运行，而在路由和环境复杂多变条件下，控制平面的任务不影响交换机的管理，高度保证了交换机系统的稳定性。

硬件交换机--L2 Switch架构

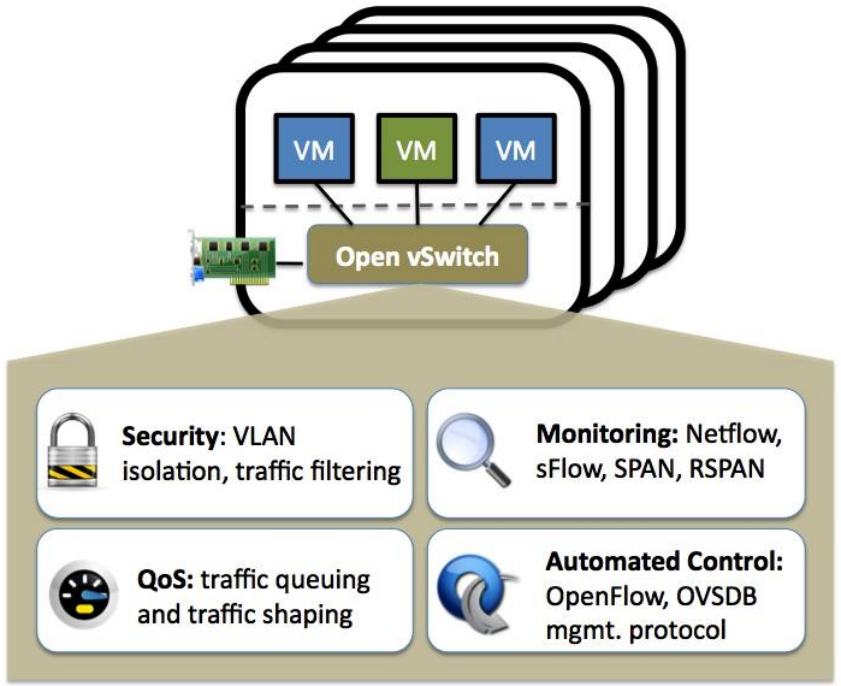
交换机的基本架构



交换机的基本架构是由带有多个RJ-45接口、PHY、MAC等模块的网络接口控制器(Network Interface Controller，NIC)和管理由各个NIC分配的收发帧缓存、转发表的软件(或ASIC)组成，通过查询转发表信息，在NIC之间进行数据帧交互。

目录

Open vSwitch简介

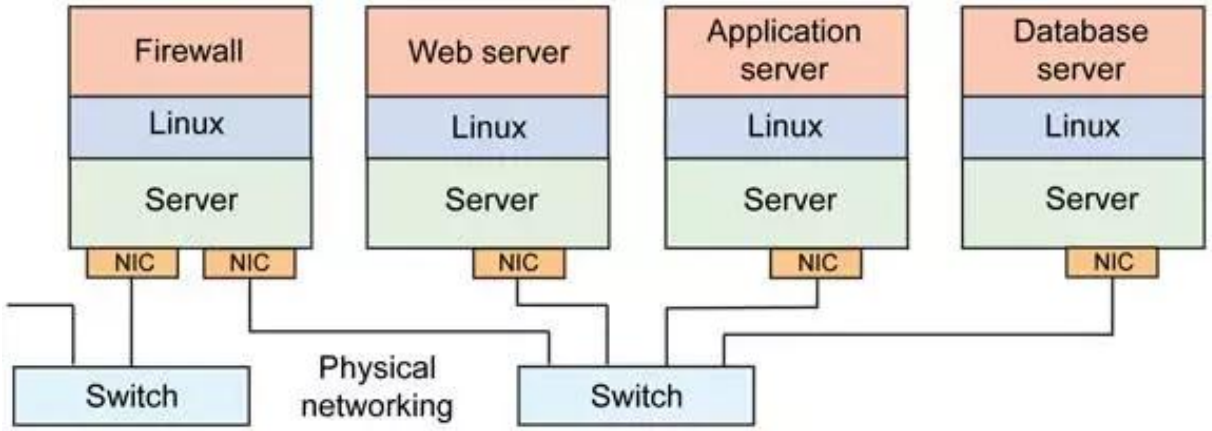


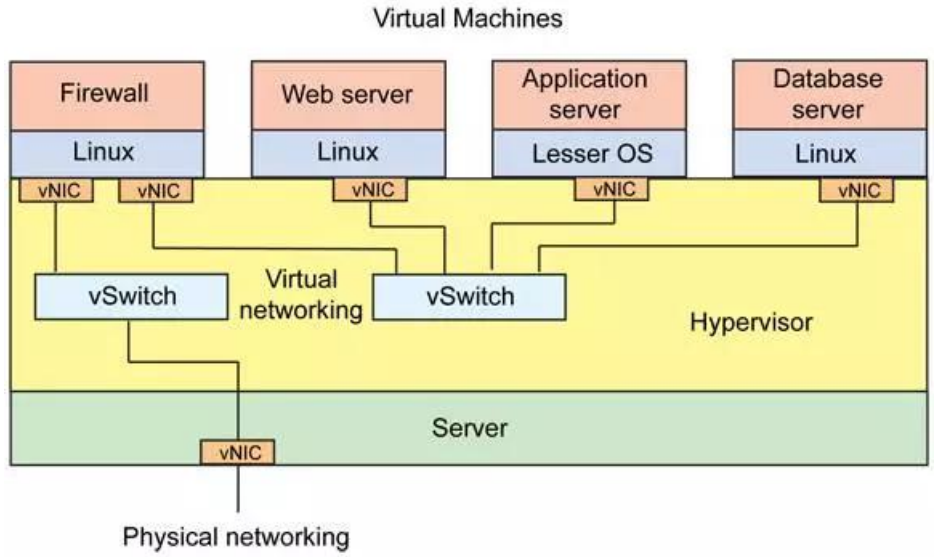
Open vSwitch简称ovs，是一个虚拟交换机软件

- 创建人：Martin Casado
- 编写语言：C语言功能：能够实现SDN架构中的SDN交换机
- 特点：工作在VM里，使用现有的主机或服务器完成SDN交换机的功能
- 支持：二层交换，网络隔离，Qos，流量监控等。最大的特性是支持openflow，能够灵活处理数据包。

Open vSwitch的产生背景

进入二十一世纪，随着x86体系架构逐步统治了**数据中心**，以Intel VT-x和AMD-V为核心的**服务器虚拟化**技术获得了突飞猛进的发展。但是，只有服务器虚拟化远远不够，云计算的隔离性、弹性、动态迁移等特性都需要网络架构的紧密支持。**网络虚拟化**迫切需要解决从NIC到虚机的“最后一公里”问题，而且这一延伸也意味着控制权从硬件转移到了软件，管理方式由手动转变为自动，这一切都为基于x86的虚拟交换机的提出铺平了道路。



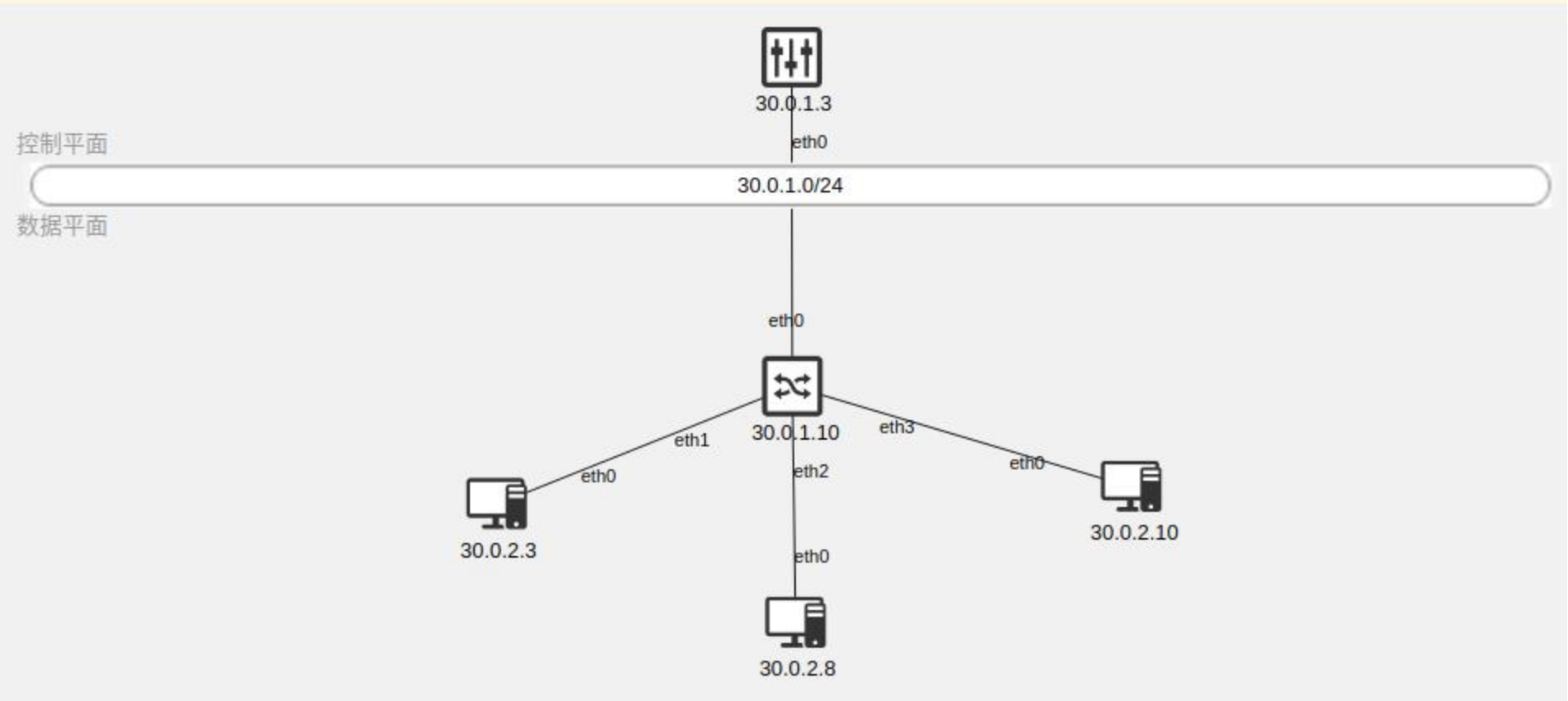


虚拟交换机开源的方案：Open vSwitch、Snabb Switch 和 Lagopus，其中Open vSwitch的知名度最高

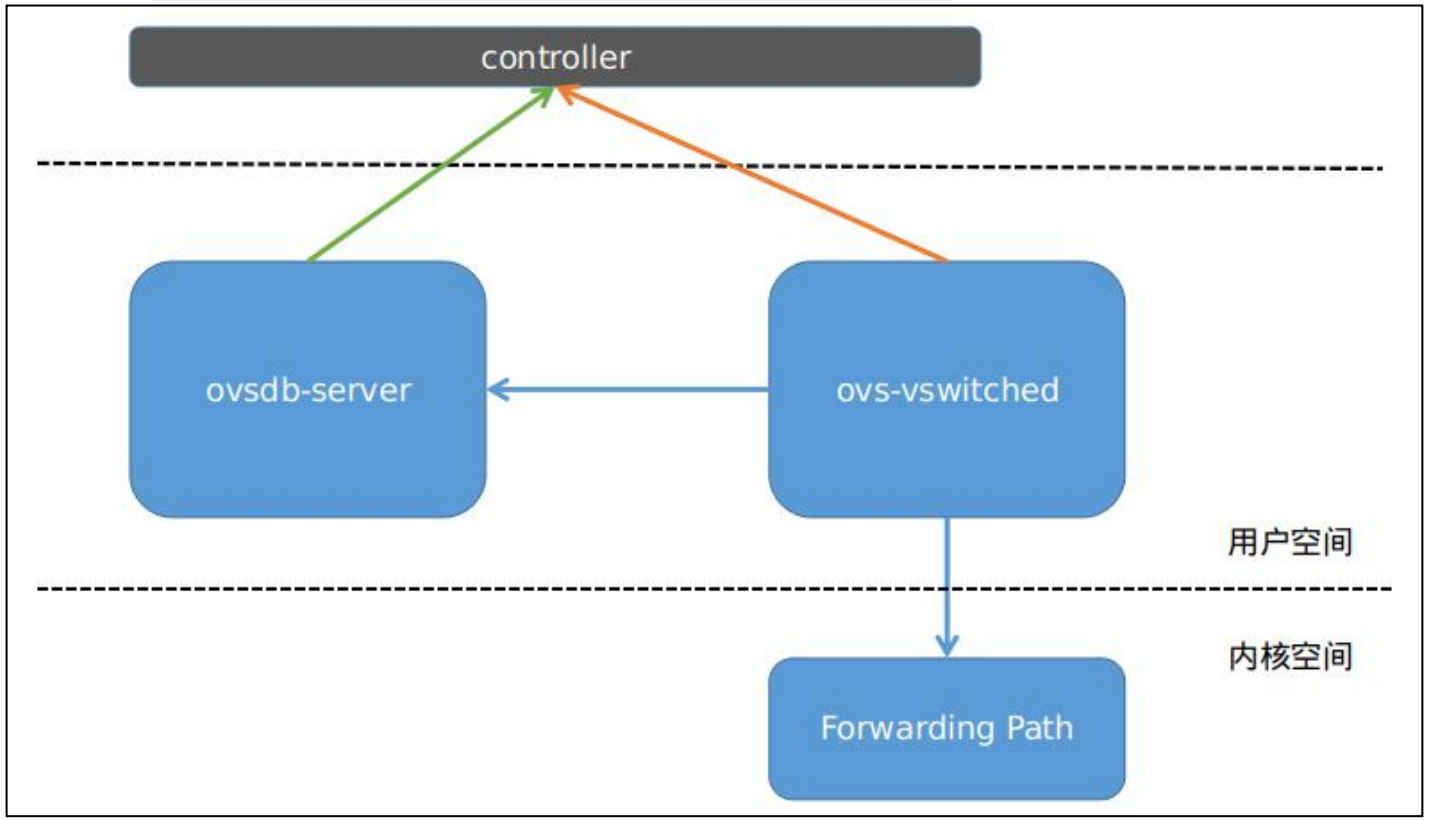
Open vSwitch 开源演进历程

OVS项目早在2009年，就已孵化诞生，于2012年7月30日发布第一个开源版本V1.7.0，内核datapath已经被纳入Linux内核作为发行版的一部分，像知名的Linux操作系统Ubuntu、Debian、Fedora也都提供了datapath的安装包

Open vSwitch使用场景

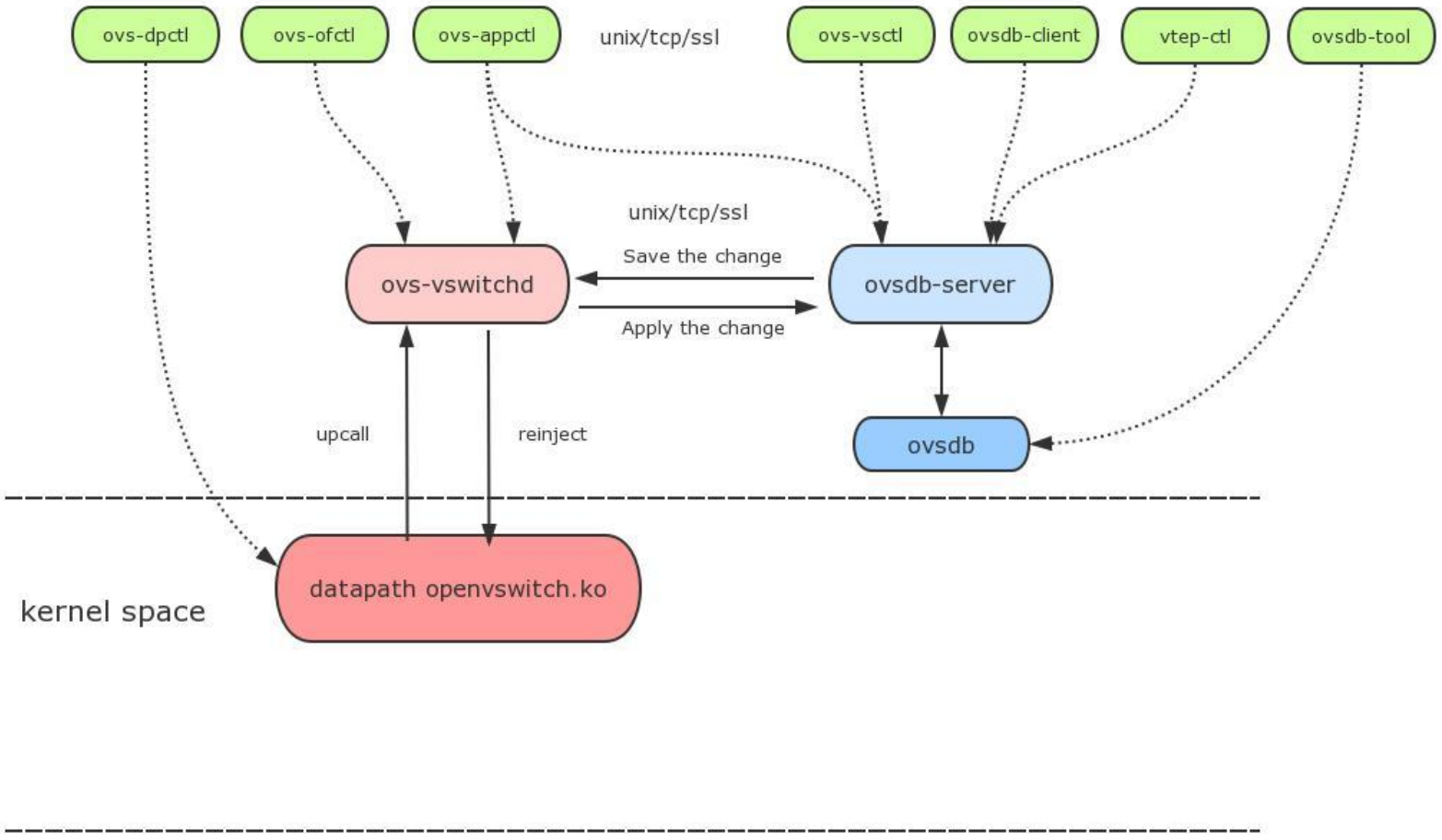


Open vSwitch组成结构



- **ovs-vswitchd:** OVS守护进程，向上和控制器通信获取流表，向下将交换信息发送到linux内核中，中间和ovsdb-server通信对设备增删改查。
- **ovsdb-server:** OVS轻量级的数据库服务器，用于保存整个OVS的设备配置信息。如端口，网桥等
- **Forwarding Path(数据通路)：** Datapath把流的match和action结果缓存，避免后续同样的流继续upcall到用户空间进行流表匹配。

Open vSwitch工作流程



① 端口收到数据包

② 内核不能处理上报 vswitchd 模块

③ vswitchd模块将数据包发送给控制器

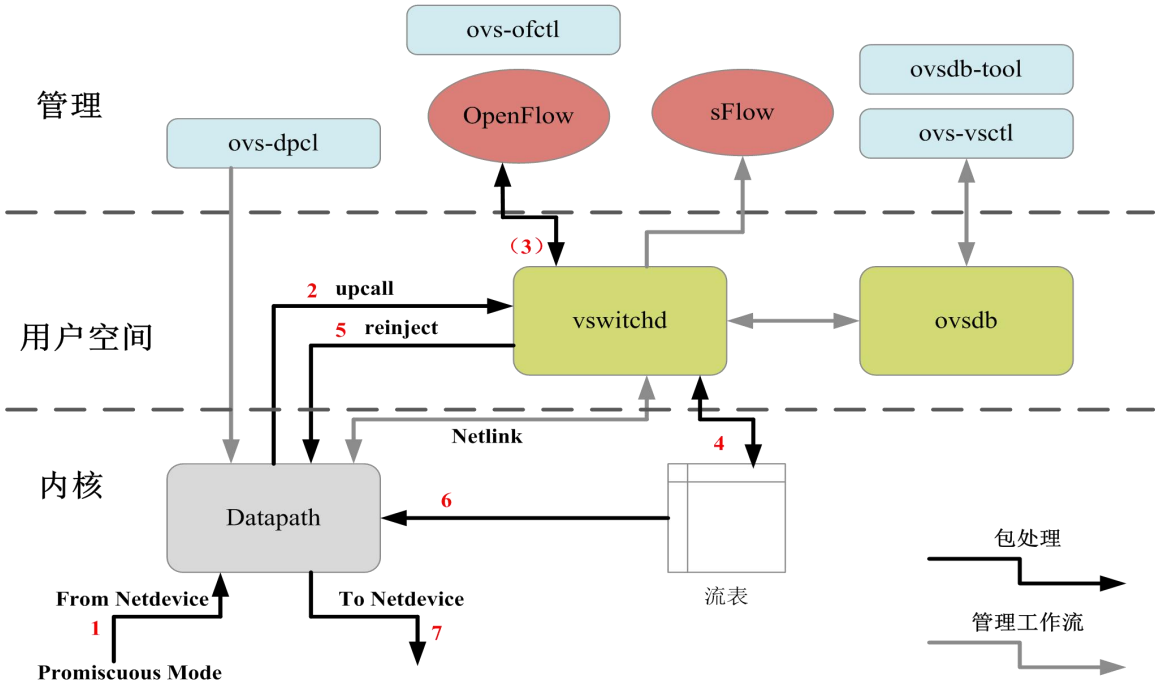
④ 控制器做出处理下发流表到ovs交换机

⑤ vswitchd模块将数据包发送给内核,同时将流表发送到 Flow Table

⑥ ovs内核匹配流表

⑦ 内核根据流表动作完成数据转发

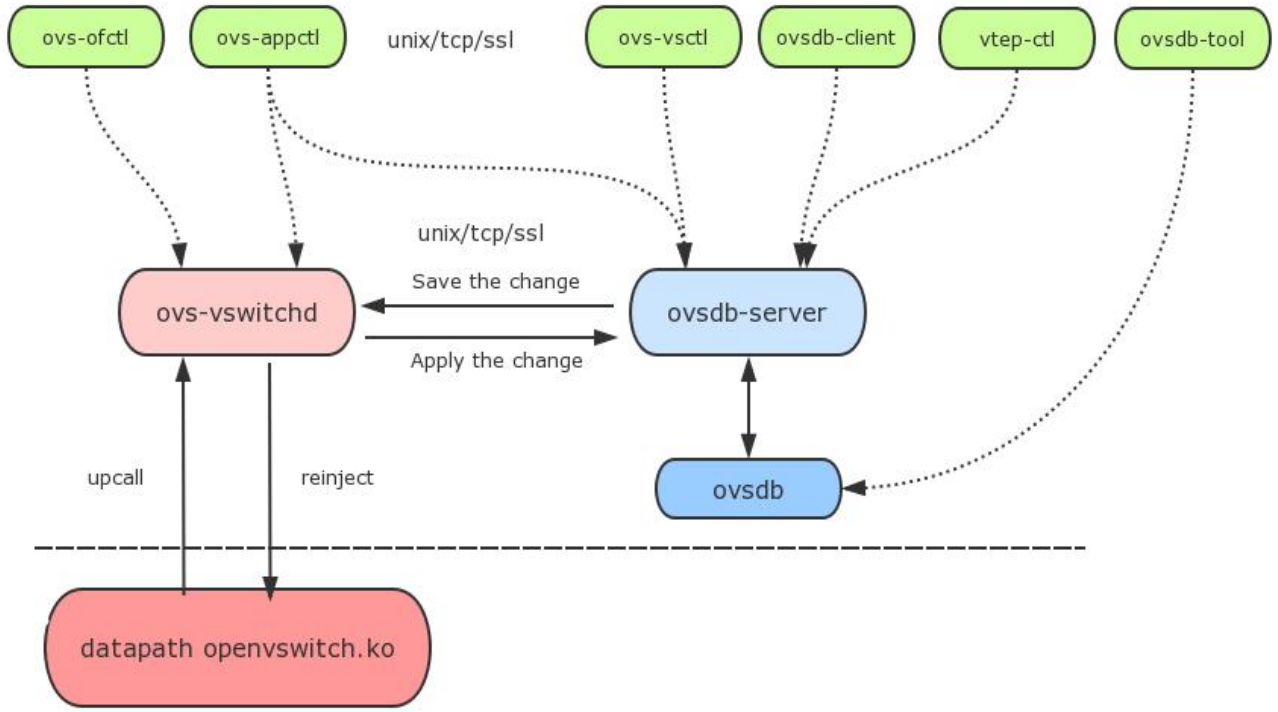
Open vSwitch的架构



- **用户数据空间和内核**
  - vswitchd：交换驱动，支持流交换的Linux内核模块
  - ovsdb：轻量级数据库服务器，为vswitchd提供配置信息
  - Datapath：数据操作流程
  - 流表：数据操作动作表
- **管理层**
  - ovs-dpctl ：用来配置vswitchd内核模块
  - ovs-ofctl ：查询和控制OpenFlow交换机和控制器
  - ovs-vsctl ：查询和更新vswitchd的配置
  - OpenFlow协议特性实现
    - OpenFlow
    - sFlow

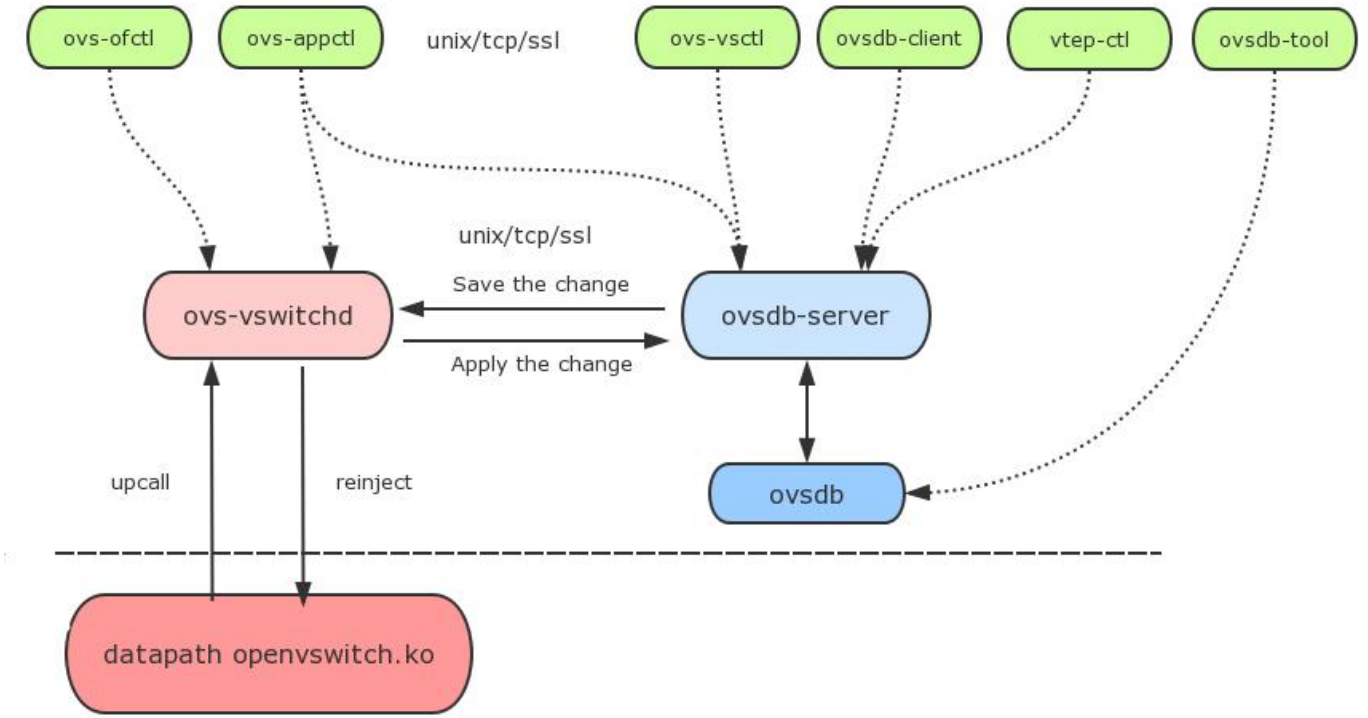
Open vSwitch——vswitchd

- **OVS的核心模块，主要负责:**
  - 检索和更新数据库信息
  - 根据数据库中的配置信息来维护和管理OVS
- **通过vswitchd可以配置一系列特性**
  - 支持IEEE 802.1Q VLAN
  - 基于MAC地址学习的二层交换
  - 端口镜像
  - sFlow监测
  - 连接OpenFlow控制器
  - 可以通过Netlink协议与内核模块datapath直接通信

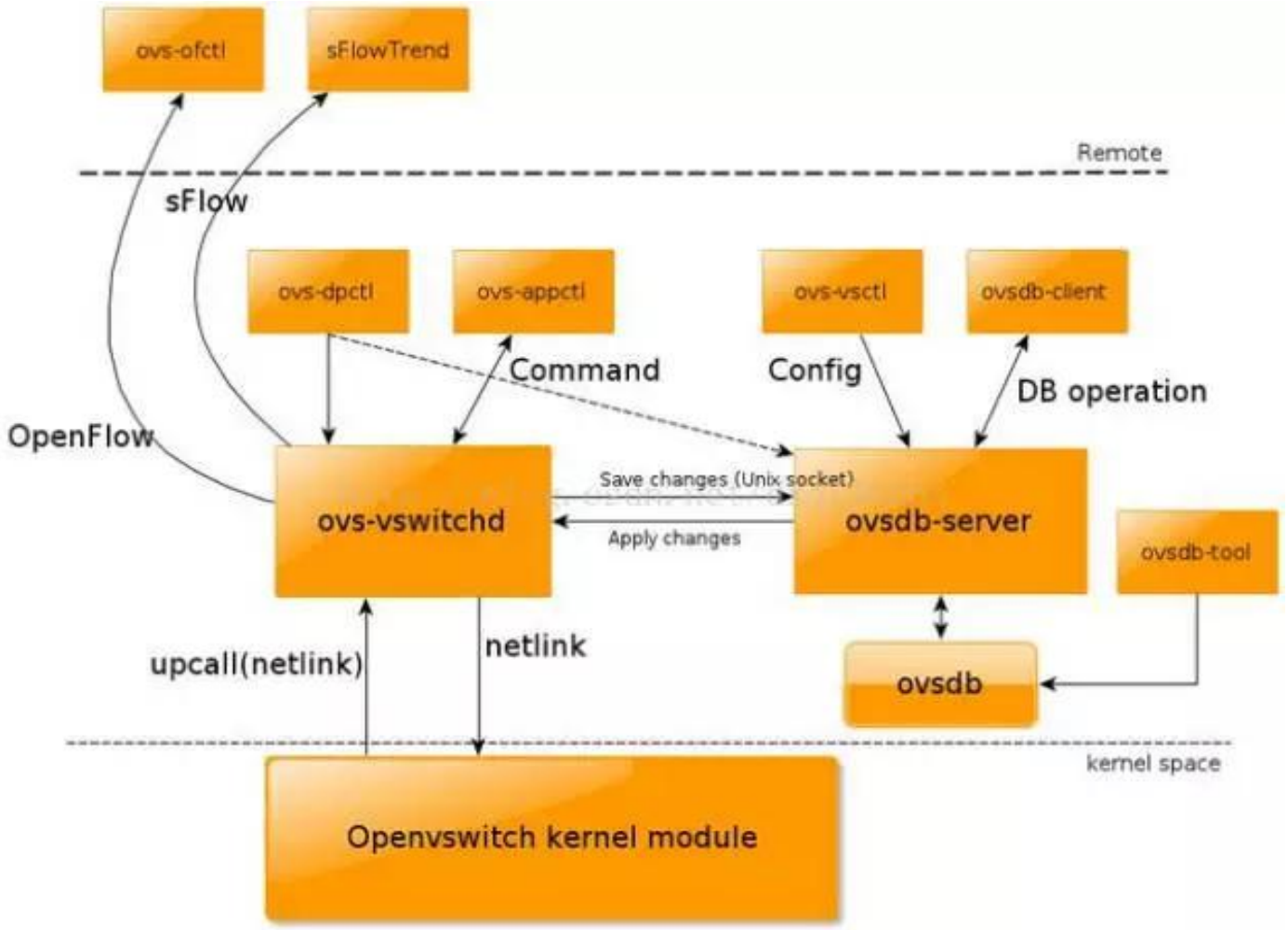


Open vSwitch—— ovsdb

- OVS的数据库
- 为vswitchd维护所有的配置信息
- 分级结构
  - 分三级
    - Table/Record/Column
    - Table包含Record，Record包含Column
  - 数据库中的信息组织形式更加有条理，便于管理和检索



Open vSwitch的管理模块



- ovs-vsctl:

管理ovsdb-server的配置，提供 OVSDB的配置方法，包括创建和删除网桥、端口等；（实验）

- ovs-ofctl:

提供ovs-vswitchd的流表配置方法；（实验）

- ovs-dpctl:

配置OVS内核模块，提供缓存流表的操作方法；

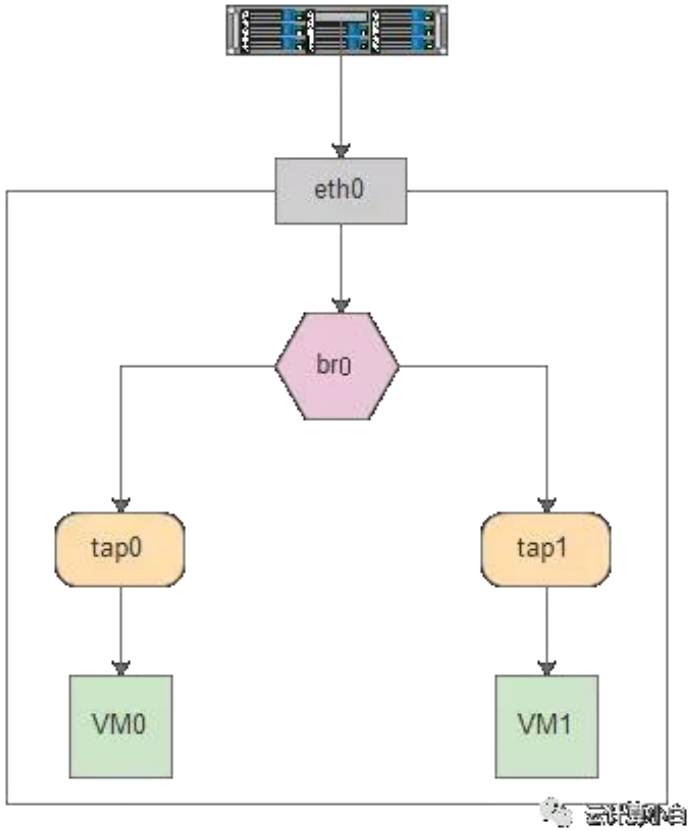
- ovsdb-tool:

创建和管理OVSDB;

Linux 下网络设备虚拟化的几种形式

Bridge

- Bridge 也是 Linux 内核实现的一个工作在二层的虚拟网络设备，但不同于 TAP/TUN 这种单端口的设备，Bridge 实现为多端口，本质上是一个虚拟交换机，具备和物理交换机类似的功能。
- Bridge 可以绑定其他 Linux 网络设备作为从设备，并将这些从设备虚拟化为端口。



Bridge 工作在二层，所以绑定到它上面的从设备 eth0、tap0、tap1 均不需要设 IP，但是需要为 br0 设置 IP，因为对于上层路由器来说，这些设备位于同一个子网，需要一个统一的 IP 将其加入路由表中。

对于实际设备 eth0 来说，本来它是有自己的 IP 的，但是绑定到 br0 之后，其 IP 就失效了，这时和 br0 共享一个 IP 网段了，在设路由表的时候，就需要将 br0 设为目标网段的地址。

Open vSwitch 环境中的各种网络设备

tap interface：命名为 tapXXXX。

linux bridge：命名为 qbrXXXX。

veth pair：命名为 qvbXXXX, qvoXXXX。

OVS integration bridge：命名为 br-int。

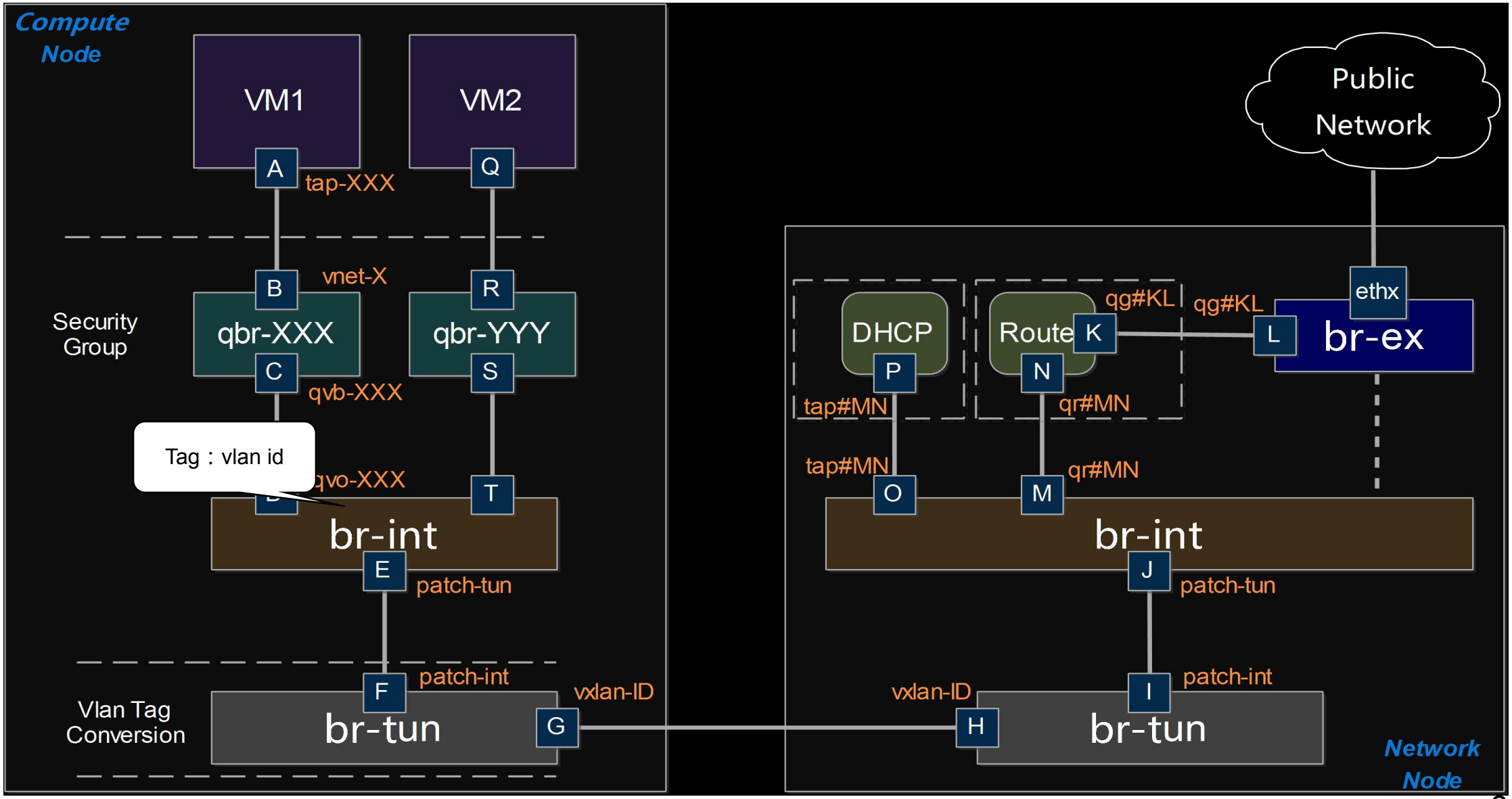
OVS patch ports：命名为 int-br-ethX 和 phy-br-ethX（X 为 interface 的序号）。

OVS provider bridge：命名为 br-ethX（X 为 interface 的序号）。

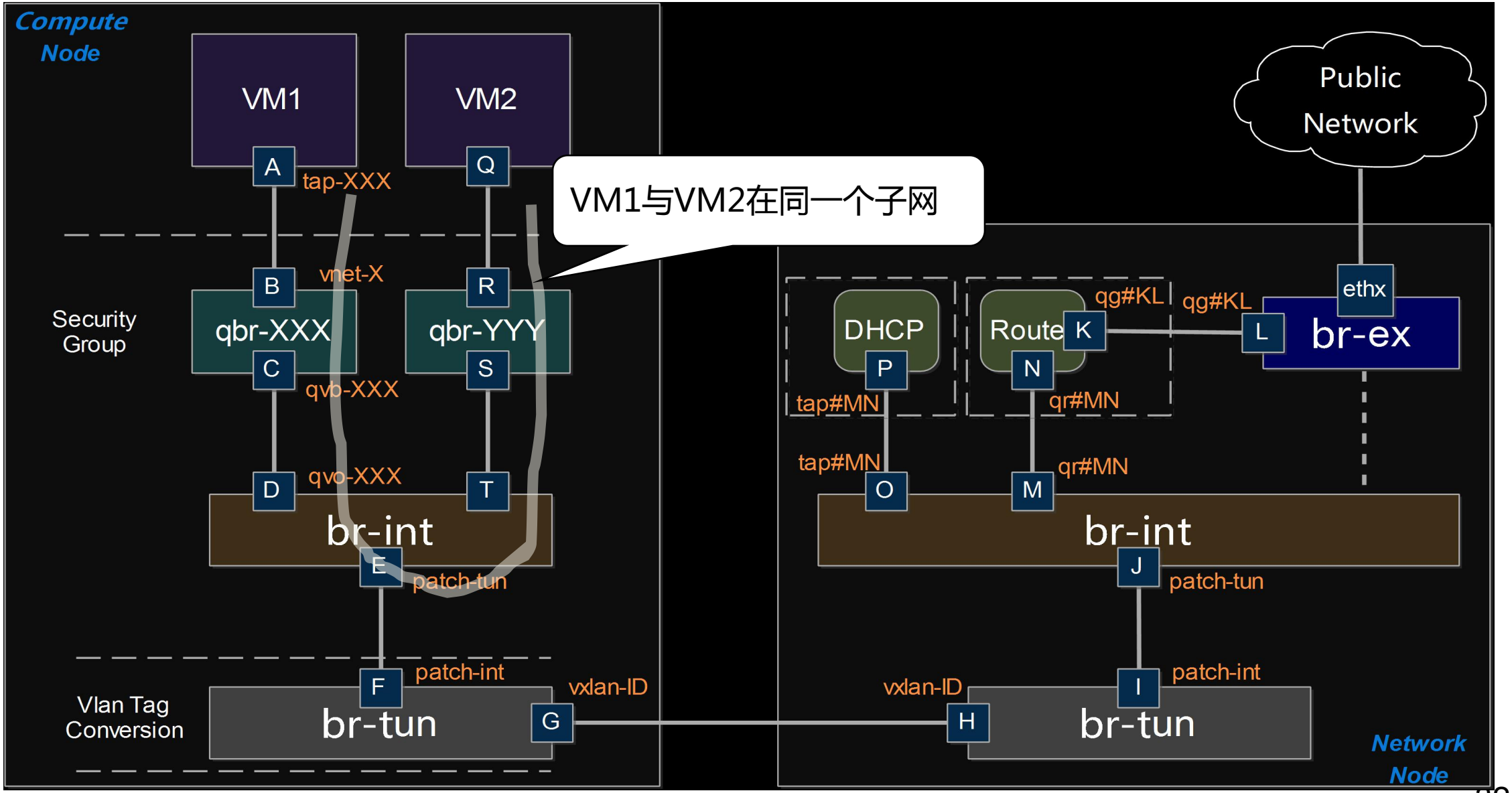
物理 interface：命名为 ethX（X 为 interface 的序号）。

OVS tunnel bridge：命名为 br-tun。

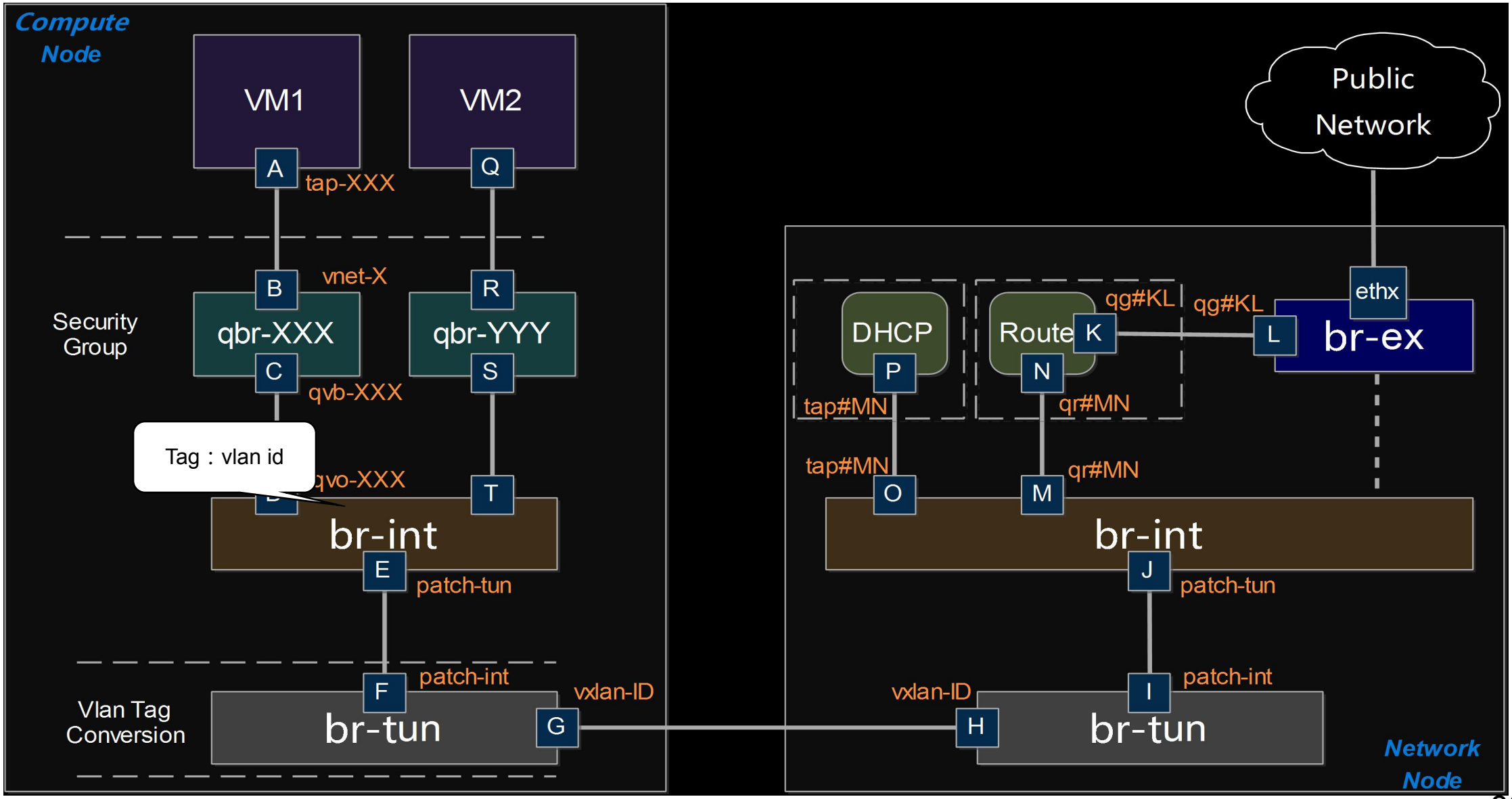
Open vSwitch在OpenStack中的应用



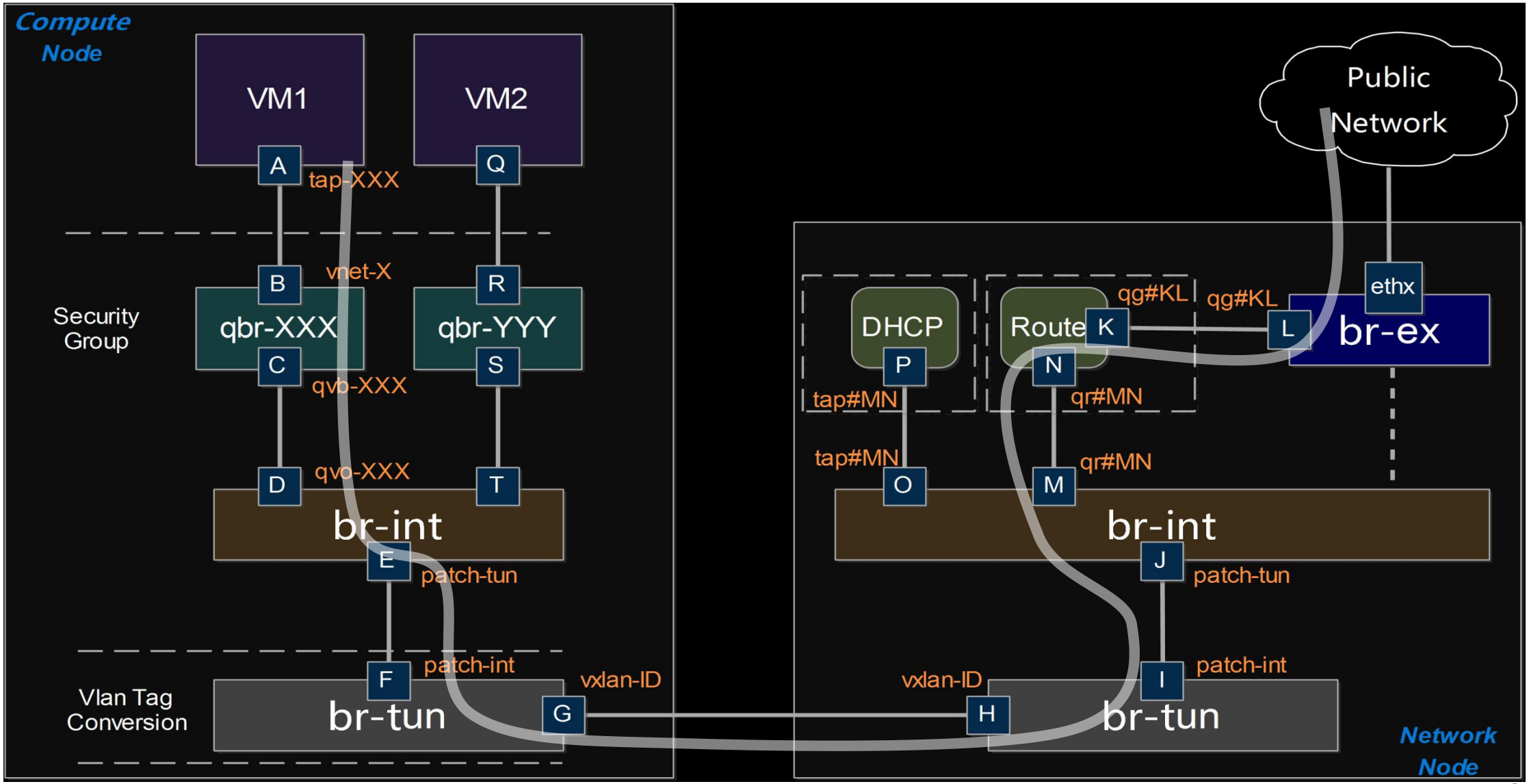
Open vSwitch在OpenStack中的应用



Open vSwitch在OpenStack中的应用



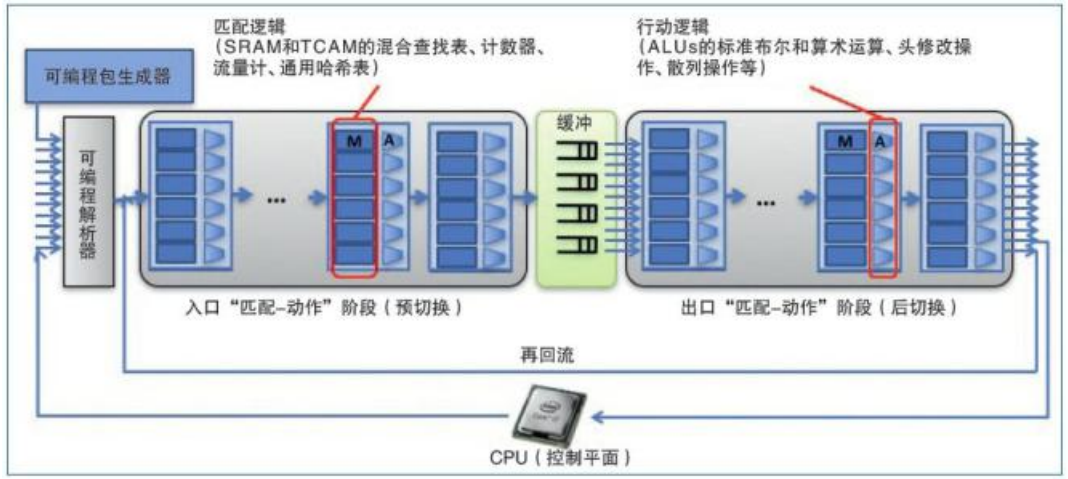
Open vSwitch在OpenStack中的应用

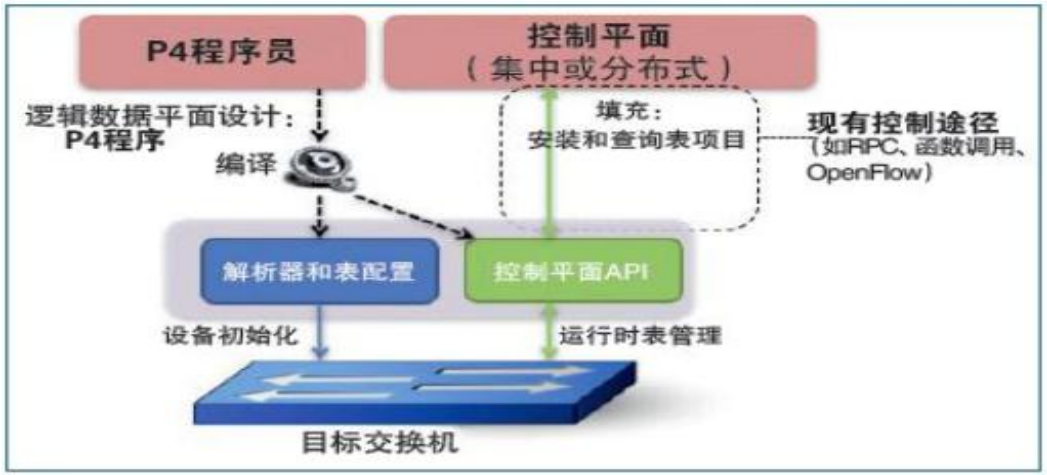


可编程数据平面和P4（2013-）

可编程数据平面具有快速实现新的协议和分组处理性能， 设计复用率高的特点

- 基于多级Match-Action的协议无关的交换结构（PISA）
- 可编程协议无关的分组处理（P4） 语言

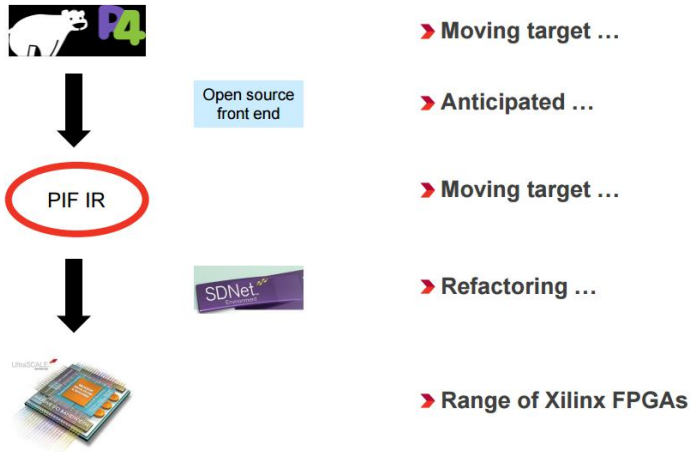




数据面可编程之P4：发展历史

数据面可编程之P4：社区成员

数据面可编程之P4：相关产品



Xilinx、Cornel推出支持P4的FPGA

INT : P4 Code Snippet

Exact-match Table Definition

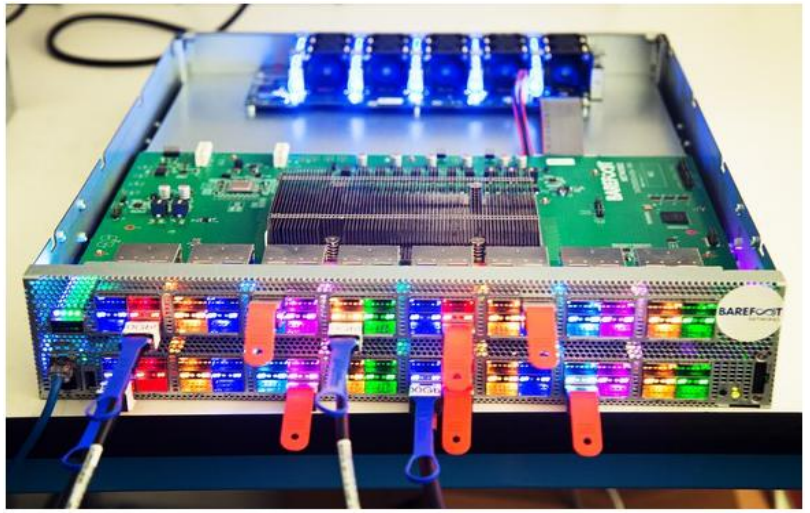
```
table int_inst {
  reads {
    int_header.instruction_mask : exact;
  }
  actions {
    int_set_header_i0;
    int_set_header_i1;
    int_set_header_i2;
    int_set_header_i3;
    .....
  }
```

Action Definitions

```
action int_set_header_i0() {
}
action int_set_header_i1() {
  int_set_header_3();
}
action int_set_header_i2() {
  int_set_header_2();
}
action int_set_header_i3() {
  int_set_header_3();
  int_set_header_2();
}
.....
```

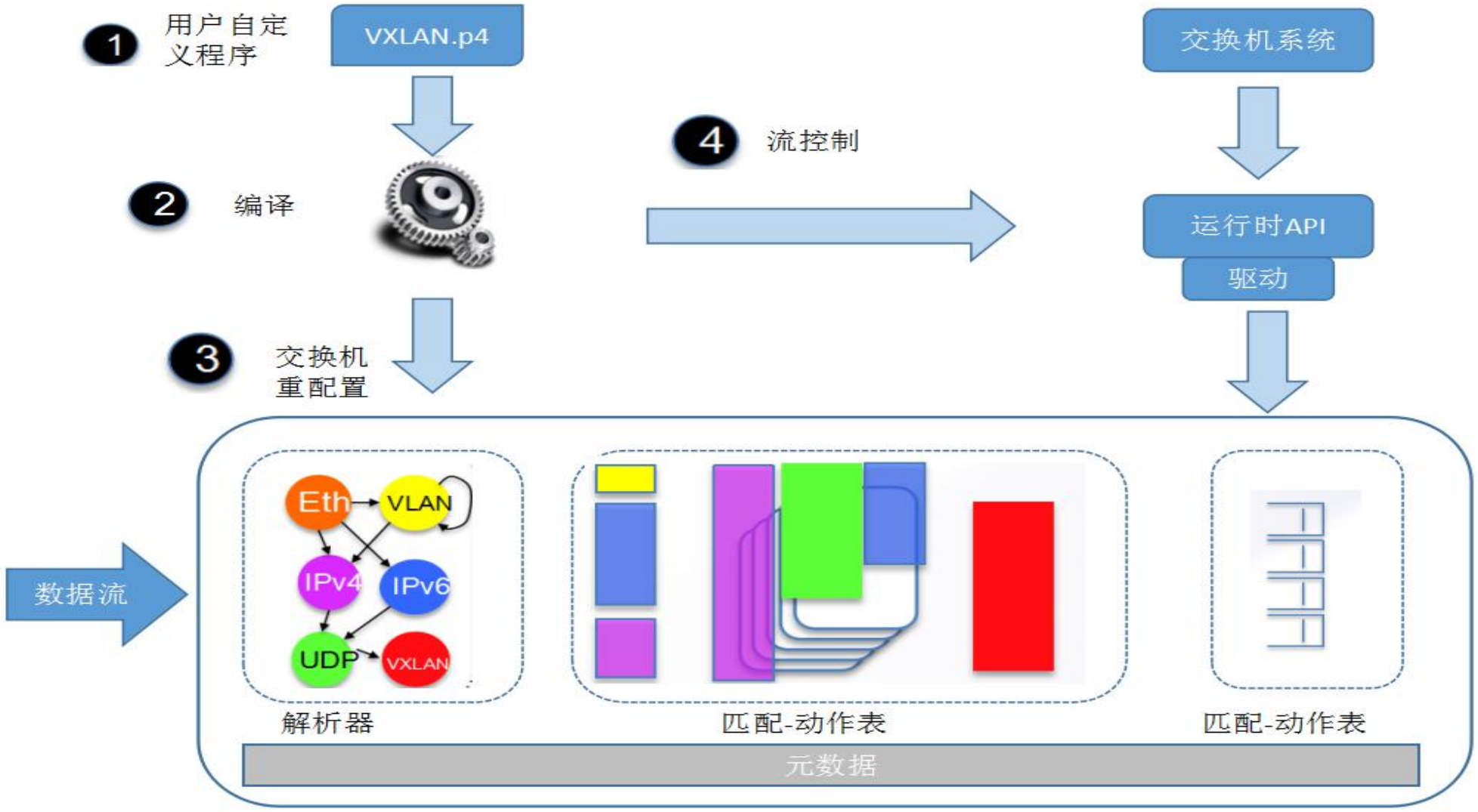
Princeton、VMWare等合作推出支持P4的vSwitch

Netronome推出的Agilio网卡及P4C - SDK



Barefoot推出6.4T可编程Tofino芯片及配套开发工具

数据面可编程之P4：工作流程



P4交换机

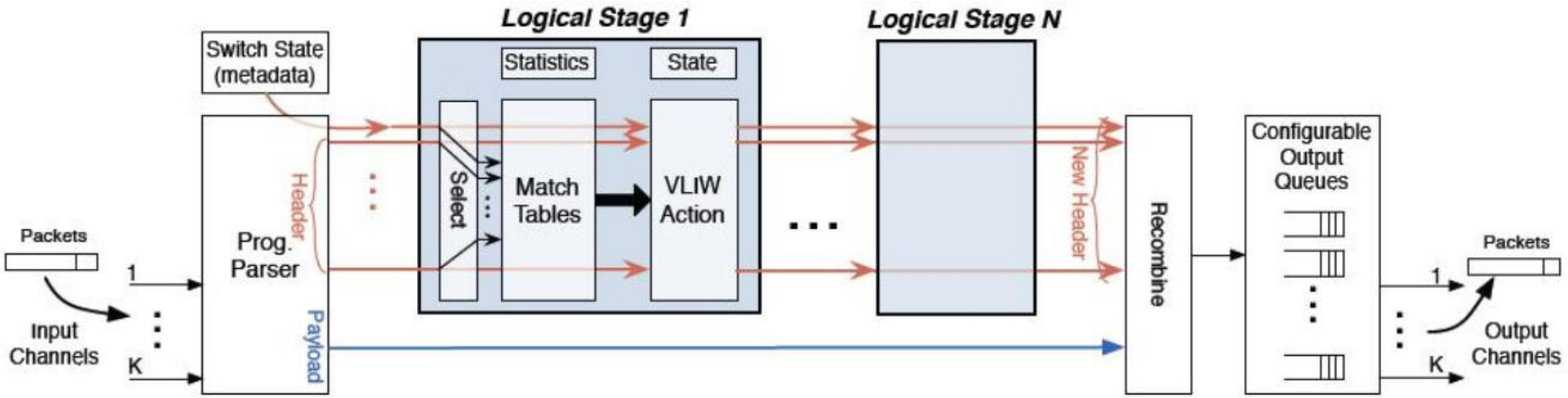
PISA的原理

- 将分组处理的表依赖图映射到固定的Match-Action匹配表中。

  32级Ingress， 32级Egress

- 支持识别和解析新的分组域， 支持新的分组处理指令以及流水线之间的队列关系。

可实现网关， 路由器和防火墙等功能， 可支持现有协议， 如MPLS和ECN， 也可以支持正在研究的协议， 如利用非标准拥塞域的RCP协议。

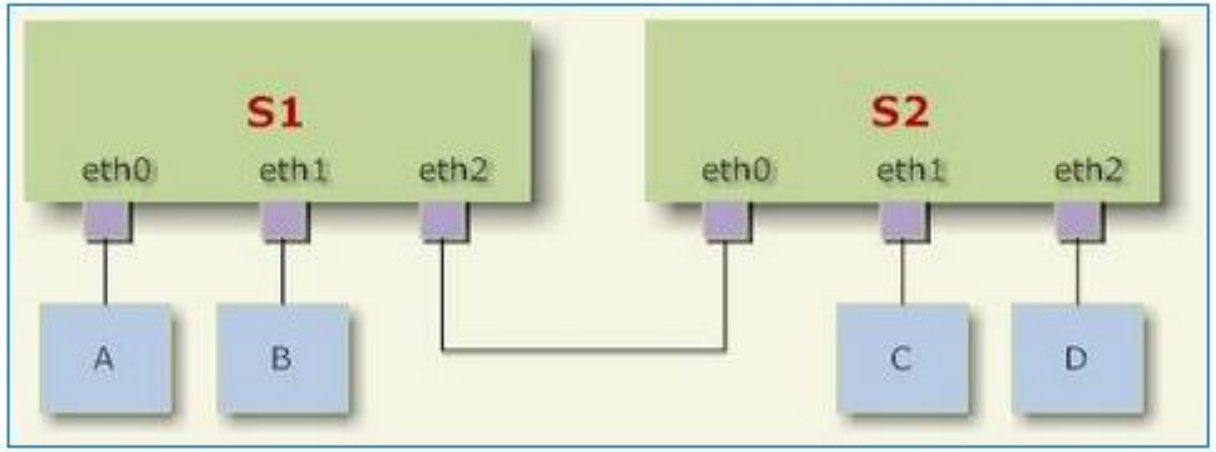


(a) RMT model as a sequence of logical Match-Action stages.

目录

桥接的概念

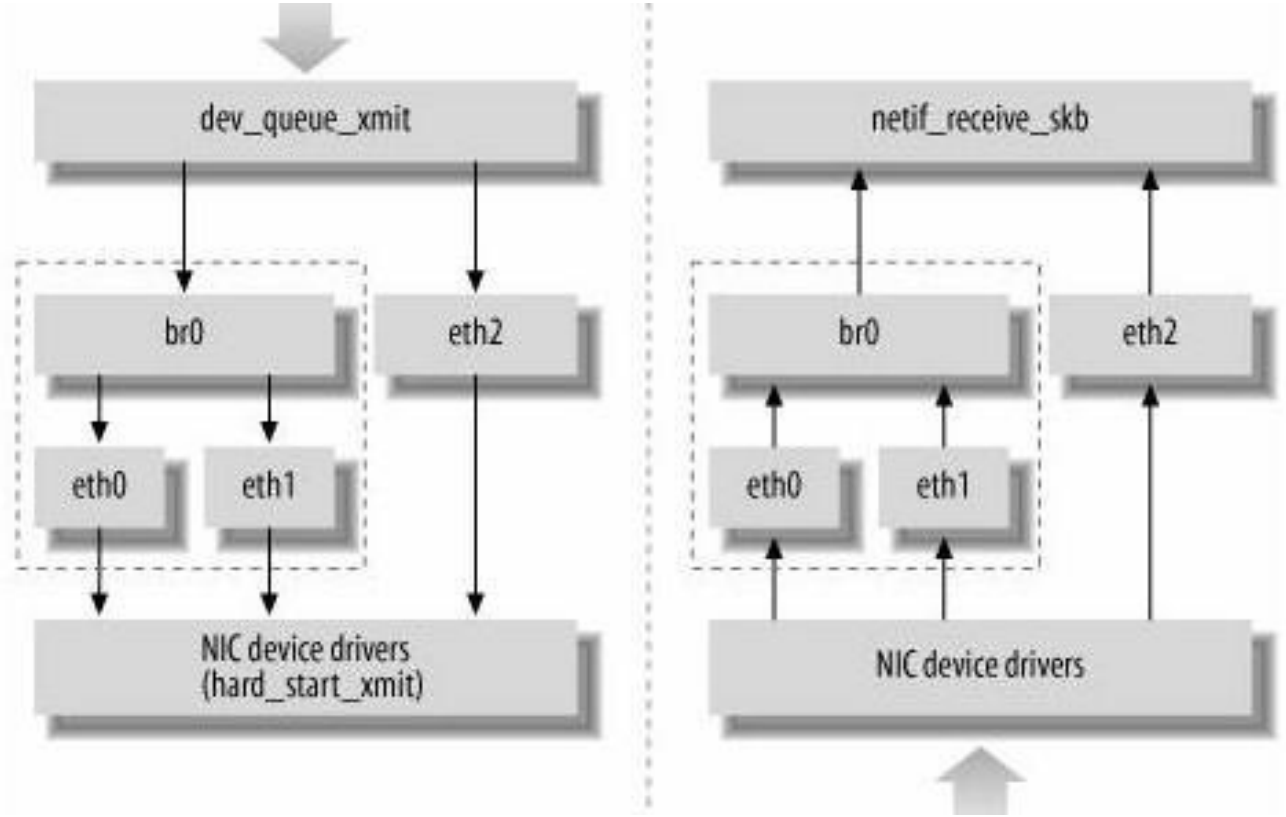
- 桥接就是把一台机器上的若干个网络接口“连接”起来。其结果是，其中一个网口收到的报文会被复制给其他网口并发送出去。以使得网口之间的报文能够互相转发。
- 交换机就是这样一个设备，它有若干个网口，并且这些网口是桥接起来的。于是，与交换机相连的若干主机就能够通过交换机的报文转发而互相通信。



主机A发送的报文被送到交换机S1的eth0口，由于eth0与eth1、eth2桥接在一起，故而报文被复制到eth1和eth2，并且发送出 去，然后被主机B和交换机S2接收到。而S2又会将报文转发给主机C、D

Linux的桥接实现

Linux内核是通过一个虚拟的网桥设备来实现桥接的。这个虚拟设备可以绑定若干个以太网接口设备，从而将它们桥接起来。



- 网桥设备br0绑定了eth0和eth1。
- 协议栈上层需要发送的报文被送到br0，网桥设备的处理代码再来判断报文该被转发到eth0或是eth1，或者两者皆是
- 反过来，从eth0或从eth1接 收到的报文被提交给网桥的处理代码，在这里会判断报文该转发、丢弃、或提交到协议栈上层

网桥的工作流程与功能

■ 工作流程

1. 从某个端口收到的二层报文，解析二层报文的源MAC和目的MAC
2. 根据源MAC学习形成MAC表
3. 根据目的MAC，原封不动的将该报文转发到适当的出端口，从而保证最终目的设备能收到这个报文

■ 功能

- MAC学习：学习MAC地址，起初，网桥是没有任何地址与端口的对应关系的，它发送数据，还是得像HUB一样，但是每发送一个数据，它都会关心数据包的来源MAC是从自己的哪个端口来的，通过学习，建立地址-端口的对照表（CAM表）。
- 报文转发：每发送一个数据包，网桥都会提取其目的MAC地址，从自己的地址-端口对照表(CAM表)中查找由哪个端口把数据包发送出去。

Open vSwitch中的网桥

- Open vSwitch中的网桥对应物理交换机，其功能是根据一定流规则，把从端口收到的数据包转发到另一个或多个端口
- 在Open vSwitch中创建一个网桥后，此时网络功能不受影响，但是会产生一个虚拟网卡，之所以会产生一个虚拟网卡，是为了实现接下来的网桥（交换机）功能
- Open vSwitch的内核模块实现了多个“数据路径”（类似于网桥），每个都可以有多个vports（类似于桥内的端口），每个数据路径也通过关联流表（flow table）来设置操作

```
root@localhost:~# ovs-vsctl show
bc12c8d2-6900-42dd-9c1c-30e8ecb99a1b
Bridge "br0"
    Port "eth0"
        Interface "eth0"
    Port "br0"
        Interface "br0"
            type: internal
ovs_version: "1.4.0+build0"
```

显示了一个名为br0的桥（交换机），这个交换机有两个接口，一个是eth0，一个是br0。

网桥管理的命令

命令	含义
init	初始化数据库（前提数据分组为空）
show	打印数据库信息摘要
add-br BRIDGE	添加新的网桥
del-br BRIDGE	删除网桥
list-br	打印网桥摘要信息
list-ports BRIDGE	打印网桥中所有port摘要信息
add-port BRIDGE PORT	向网桥中添加端口
del-port BRIDGE PORT	删除网桥上的端口
get-controller BRIDGE	获取网桥的控制器信息
del-controller BRIDGE	删除网桥的控制器信息
set-controller BRIDGE TARGET...	向网桥添加控制器

示例：

ovs-vsctl add-br br0 #添加名为br0的网桥

ovs-vsctl del-br br0 #删除名为br0的网桥

ovs-vsctl add-port br0 eth0 #将网络接口eth0挂接到网桥br0上

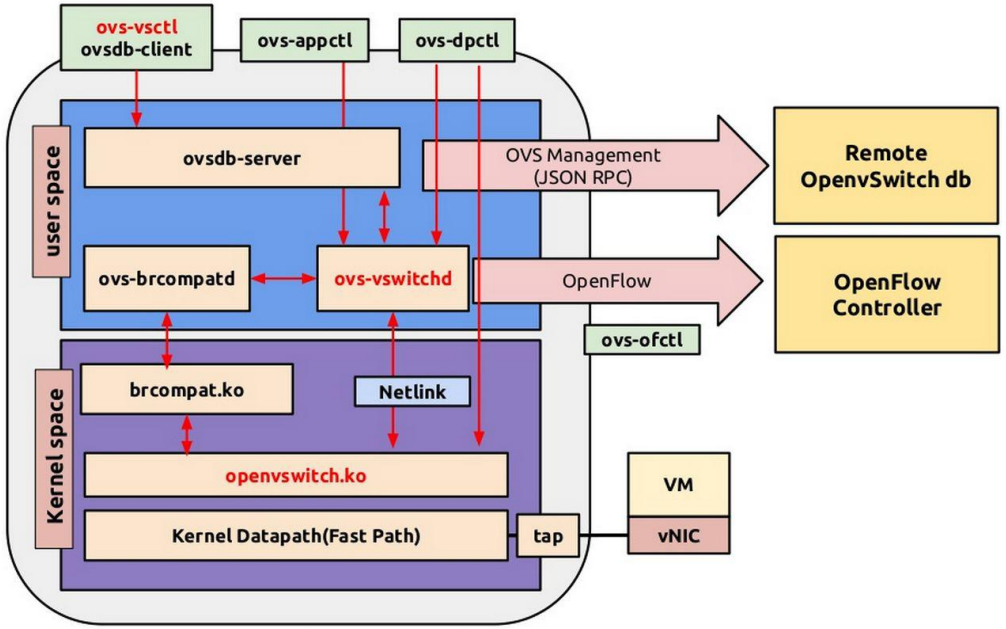
ovs-vsctl del-port br0 eth0 #删除网桥br0上挂接的eth0网络接口

目录

Open vSwitch的流表概述

在SDN环境下，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，交换机将此数据包发送给控制器，由控制器决策数据包如何处理。控制器下发决策后，交换机根据控制器下发的信息来进行数据包的处理，即转发或者丢弃该数据包。可以通过对流表操作来控制交换机的转发行为。

- OpenFlow是用于管理交换机流表的协议，ovs-ofctl是Open vSwitch提供的命令行工具。
- 在没有配置OpenFlow控制器的模式下，用户可以使用**ovs-ofctl**命令通过OpenFlow协议去连接Open vSwitch来创建、修改或删除Open vSwitch中的流表项，并对Open vSwitch的运行状况进行动态监控。



流表管理命令

ovs-ofctl 命令格式为：**ovs-ofctl [命令] [选项]**

命令	含义
show SWITCH	输出OpenFlow信息。
dump-ports SWITCH PORT	输出端口统计信息。
dump-ports-desc SWITCH	输出端口描述信息。
dump-flows SWITCH	输出交换机中所有的流表项。
dump-flows SWITCH FLOW	输出交换机中匹配的流表项。
add-flow SWITCH FLOW	向交换机添加流表项。
add-flows SWITCH FILE	从文件中向交换机添加流表项。
mod-flows SWITCH FLOW	修改交换机的流表项。
del-flows SWITCH FLOW	删除交换机的流表项。

示例：ovs-ofctl dump-flows br0　#查看交换机中的所有流表项

```
root@openlab:~# ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=559.618s, table=0, n_packets=0, n_bytes=0, idle_age=559, priority=0 actions=NORMAL
 cookie=0x0, duration=20.837s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, idle_age=20, priority=17,in_port=3 actions=out
put:2
```

流表管理的命令

ovs-ofctl的Flow语法由一系列field=value形式的键值对组成，用英文逗号和空格隔开

字段名称	说明
in_port=port	传递数据包的端口的OpenFlow端口编号。
dl_vlan=vlan	数据包的VLAN Tag值，范围是0-4095，0xffff代表不包含VLAN Tag的数据包。
dl_src=<MAC>dl_dst=<MAC>	匹配源或者目标的MAC地址：01:00:00:00:00:00/01:00:00:00:00:00 代表广播地址。00:00:00:00:00:00/01:00:00:00:00:00 代表单播地址。
dl_type=ethertype	匹配以太网协议类型，其中：dl_type=0x0800 代表IPv4协议。dl_type=0x086dd 代表IPv6协议。dl_type=0x0806 代表ARP协议。
nw_src=ip[/netmask]nw_dst=ip[/netmask]	当dl_type=0x0800时，匹配源或者目标的IPv4 地址，可以使用IP地址或者域名。
nw_proto=proto	和dl_type字段协同使用。当dl_type=0x0800时，匹配IP协议编号。当dl_type=0x086dd代表IPv6协议编号。
table=number	指定要使用的流表的编号，范围是0-254，在不指定的情况下，默认值为0。通过使用流表编号，可以创建或者修改多个Table中的Flow。
reg<idx>=value[/mask]	交换机中的寄存器的值。当一个数据包进入交换机时，所有的寄存器都被清零，用户可以通过Action的指令修改寄存器中的值。

示例：ovs-ofctl add-flow br0 idle_timeout=1000,priority=17,in_port=3,actions=output:2

表示将端口3接收到的数据包从端口2输出

流表管理的命令

ovs-ofctl的add-flow,add-flows以及mod-flows命令都需要actions字段，描述对匹配的数据包执行的动作，在上述的三条命令中，actions字段是flow的一部分，actions字段中可以有多个action，它们之间用逗号隔开，一个flow的完整语法如下：

<field>=<value>,[<field>=<value>]...,actions=<action>[,<action>...]

操作	说明
output:port	输出数据包到指定的端口，port是指端口的OpenFlow端口编号。
mod_vlan_vid	修改数据包中的VLAN tag。
strip_vlan	移除数据包中的VLAN tag。
mod_dl_src/ mod_dl_dest	修改源或者目标的MAC地址信息。
mod_nw_src/mod_nw_dst	修改源或者目标的IPv4地址信息。
resubmit:port	替换流表的in_port字段，并重新进行匹配。
load:value->dst[start..end]	写数据到指定的字段。

谢谢！