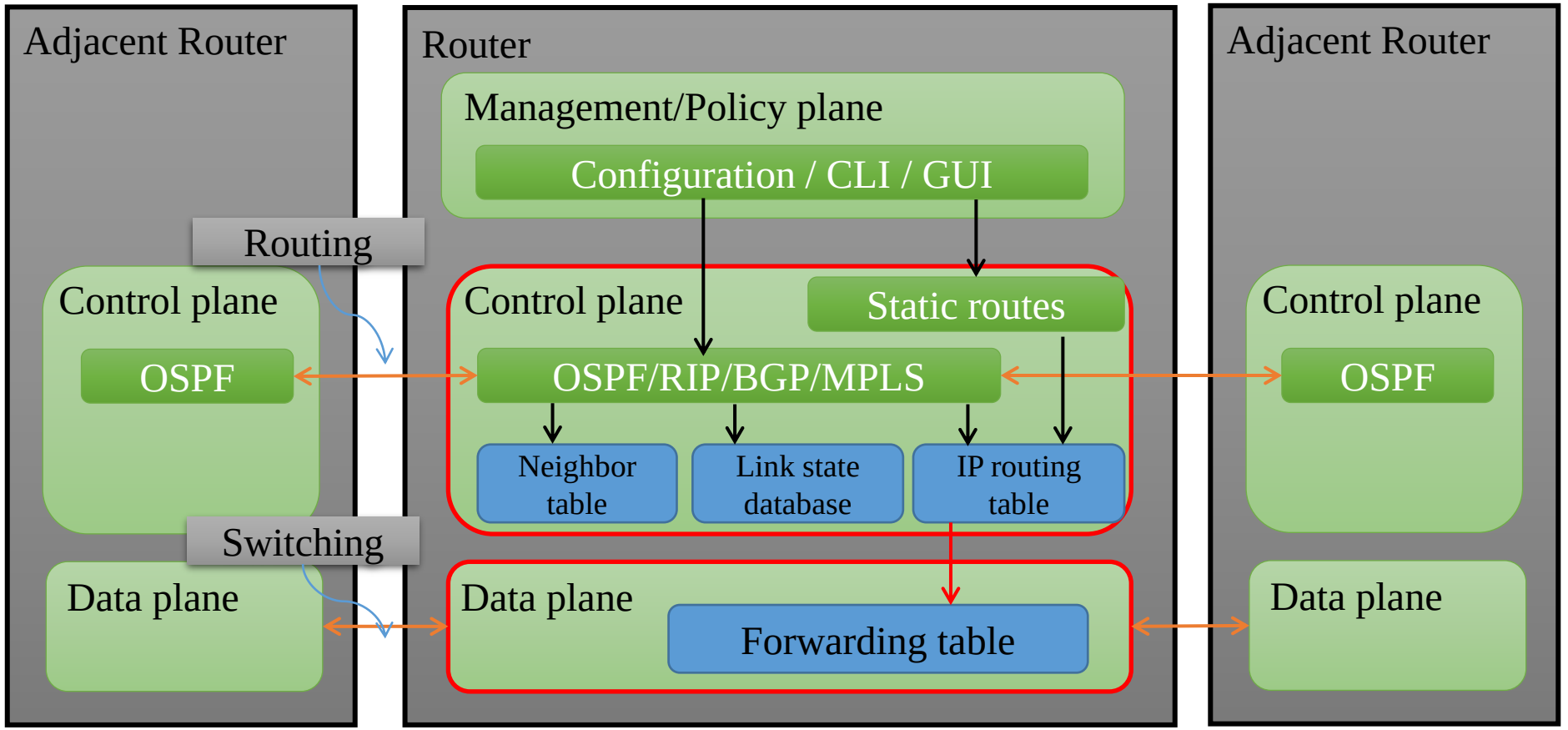
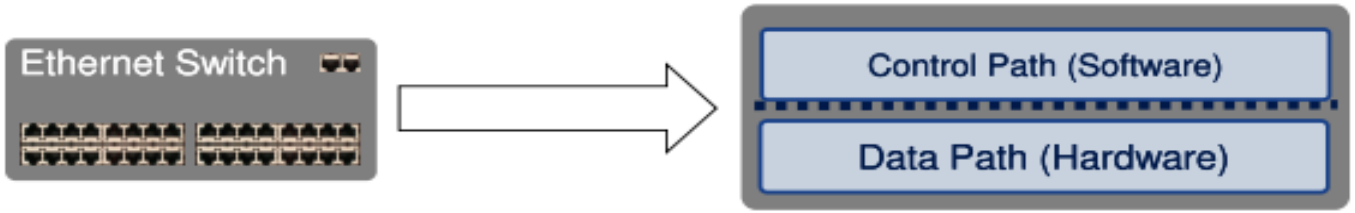
软件定义网络

计算机学院

SDN网络概述

传统网络节点：Router/Switch





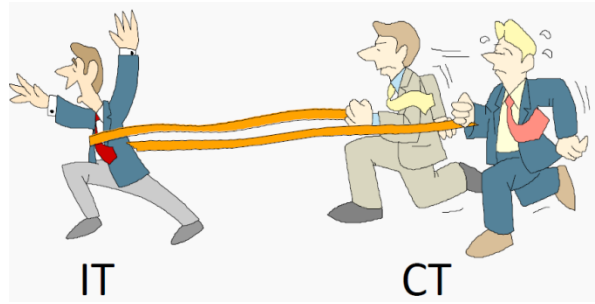
封闭的专有的网络设备

Vertically Integrated Systems Have Changed Little Over the Past 15 Years

网络体系寻求变革，需要更加灵活和富有弹性

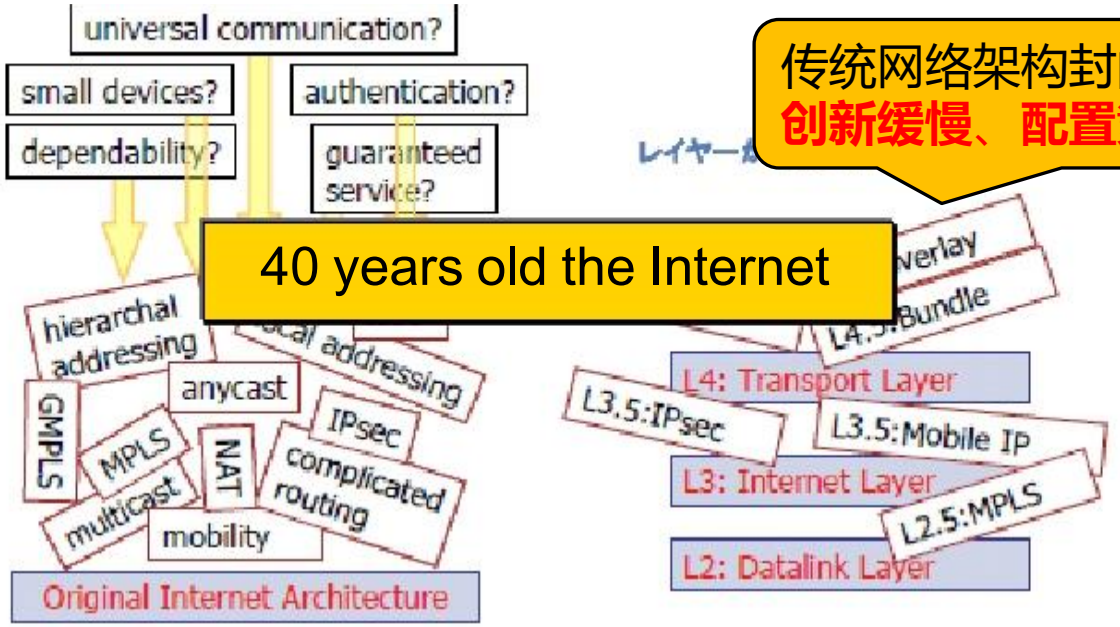
互联网业务	
产品调研	海量互联网用户，**及时反馈**
产品设计	微创新**每年上线上百款**
产品开发	快速出模，**敏捷迭代**
产品发布	灰度发布，**及时回滚**
产品消退	长尾持续，**原地复活**

新业务的发展



业务和网络天然割裂

用户、业务的多样性需求



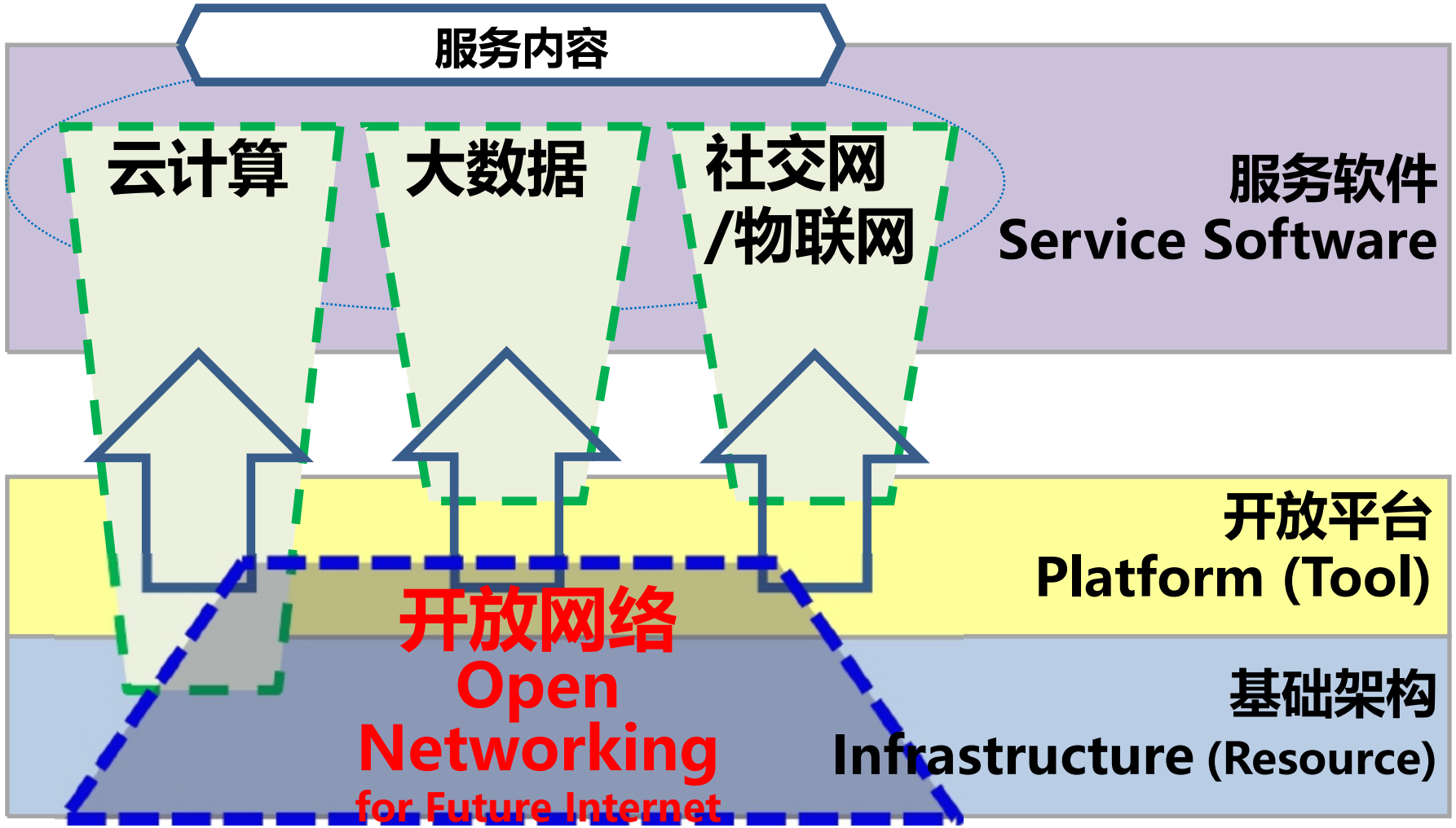
传统网络架构封闭、保守，网络**创新缓慢、配置繁琐、效率低下**

40 years old the Internet

新业务的兴起对传统的网络系统提出新的挑战

未来网络服务需要开放的网络

传统网络架构封闭保守，**网络创新缓慢、配置繁琐、效率低下**



- 各种IT技术都要求网络设备的开放，以软件获得功能灵活性将成为一种必然趋势
- 网络虚拟化是未来网络发展的关键。抽象网络能力，实现资源池化

SDN应运而生

- 便捷延伸网络覆盖
- 提供网络专用通道
- 分布式访问控制
- 租户网络资源隔离
- 开放设备接口和管理协议，底层实现透明

SDN概念与定义

SDN定义

一组直接使能编程、协同、控制和管理网络资源的技术，以利于以动态和可扩展的方式对网络资源设计、交付和运营。----**ITU-T Y.3300 (Y.SDN-FR)：Feb. 2014**

一种设计、建设和运营用于支持服务智能性网络的新方法，**SDN** 将类似于已经用在服务器设施上的抽象、虚拟化和协同的智能性带到网络。----**Gartner**

- 软件定义网络（Software Defined Networking, SDN）是一种新型的网络技术，其设计理念是将网络的控制平面与数据转发平面进行分离，并实现可编程化的集中控制。
- 传统网络设备紧耦合的网络架构被分拆成应用、控制、转发三层分离的架构。控制功能被转移到了服务器，上层应用、底层转发设施被抽象成多个逻辑实体。

SDN不是一种具体的技术，而是一种思想，一种理念

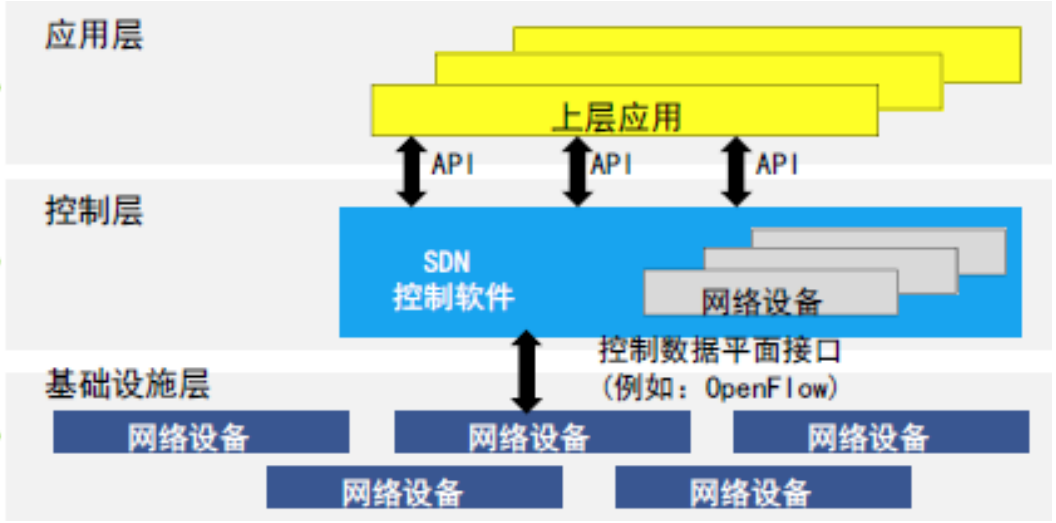
SDN的核心诉求：让软件应用参与到网络控制中并起到主导作用，而不是让而各种固定模式的协议来控制网络

为了满足这种核心诉求，SDN思想指导下的网络必须设计一种新的架构

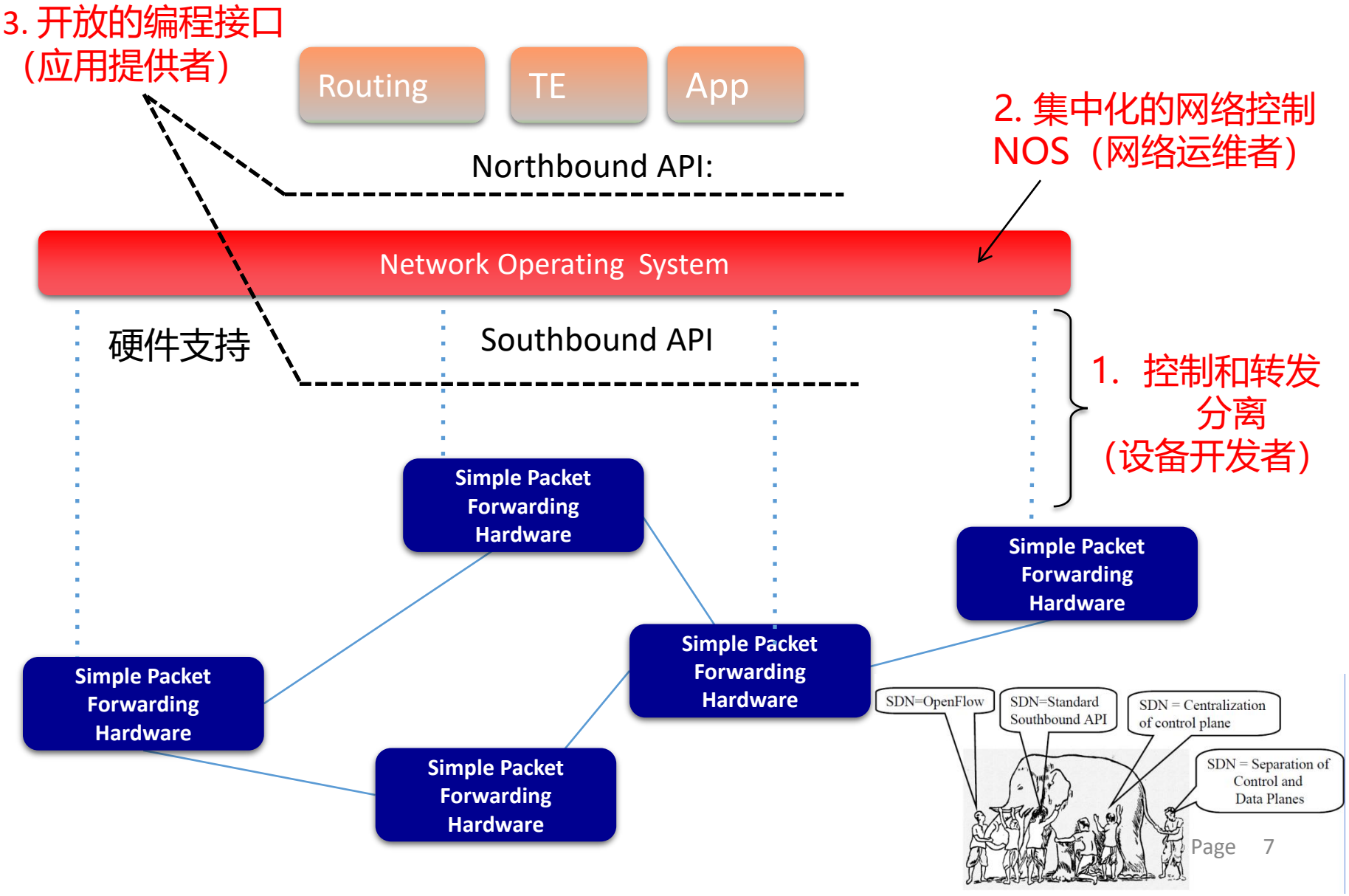
应用层，不同的应用逻辑通过控制层开放的API管理能力控制设备的报文转发功能

控制层，由SDN控制软件组成，与下层可用OpenFlow协议通信

基础设施层，由转发设备组成



SDN的本质及核心特点



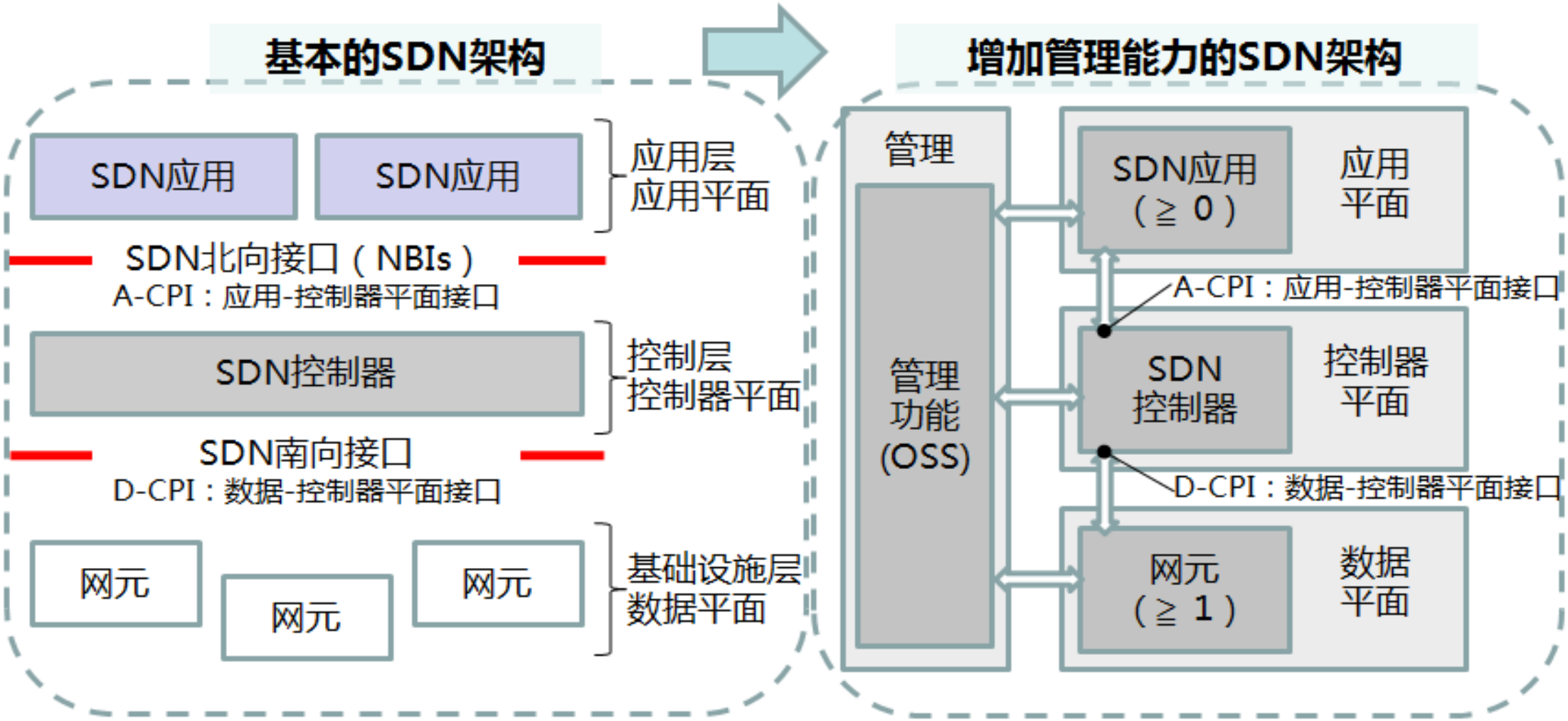
北向接口，为应用提供编程接口

控制器/Controller，对网络的抽象层，NOS网络操作系统

南向接口，设备控制协议，控制设备的转发行为

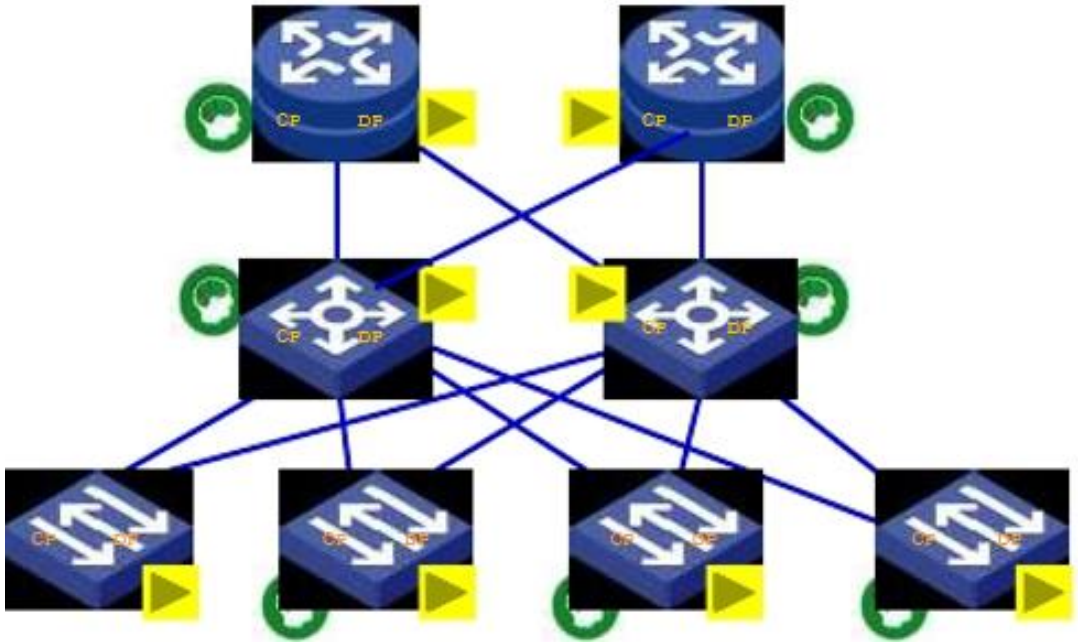
SDN的架构

- SDN＝控制与转发分离+逻辑集中控制＋控制与业务分离 **网络资源能力状态开放**



控制转发分离

- 传统网络设备的CP与DP 不分离；
- 设备之间通过控制协议交互转发信息；

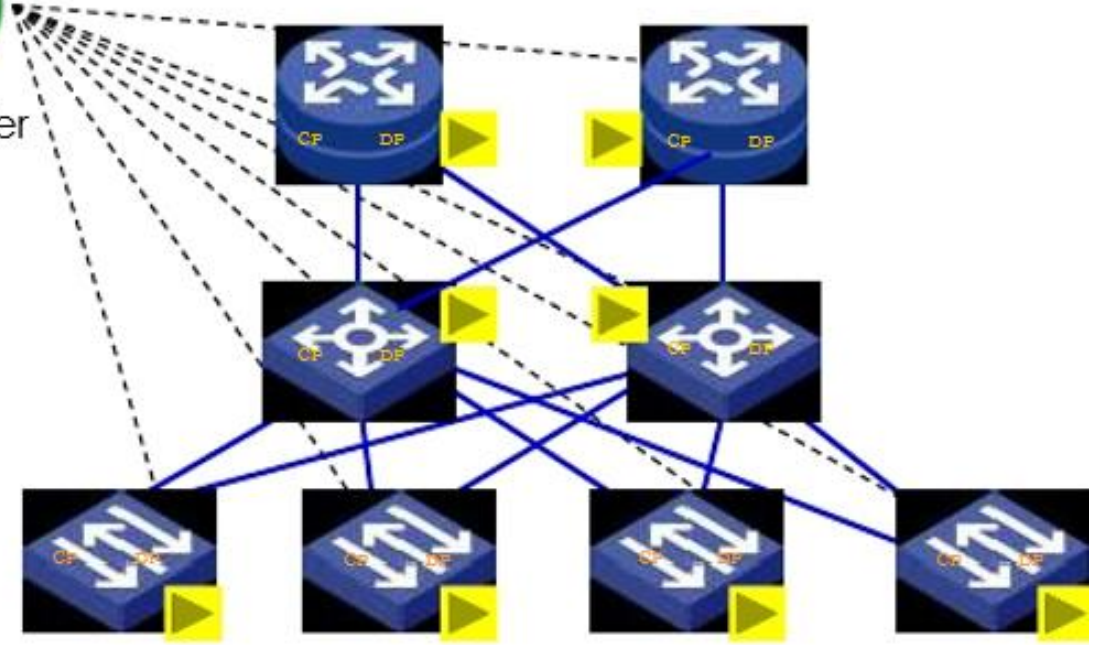


控制平面（CP）

转发平面（DP）

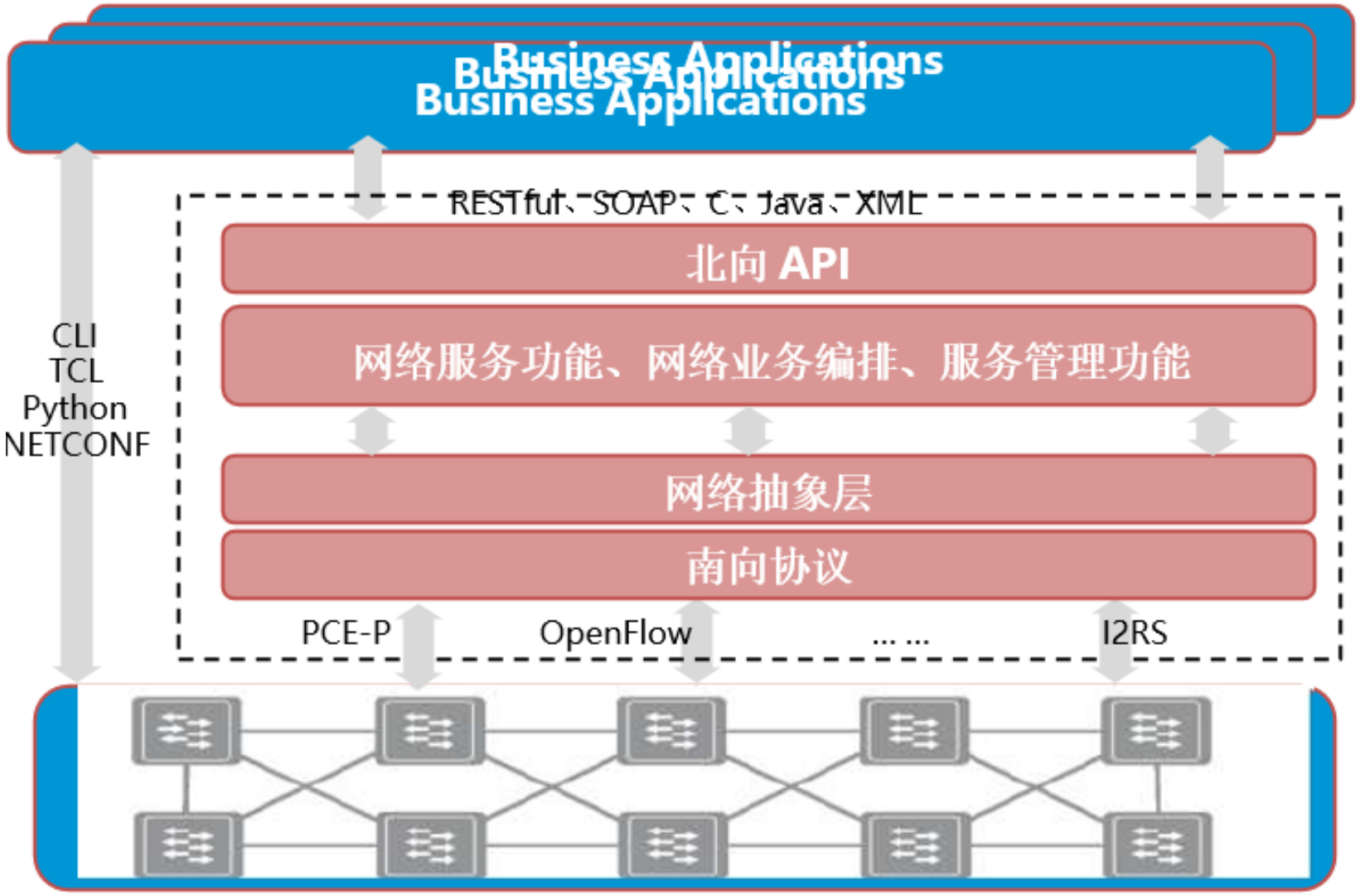
- SDN 的思路是将网络设备的控制平面集中上收到Controller;
- 网络设备上只保留转发平面（转发表项）；
- 通过Controller实现网络统一部署和网络自动化；

Controller

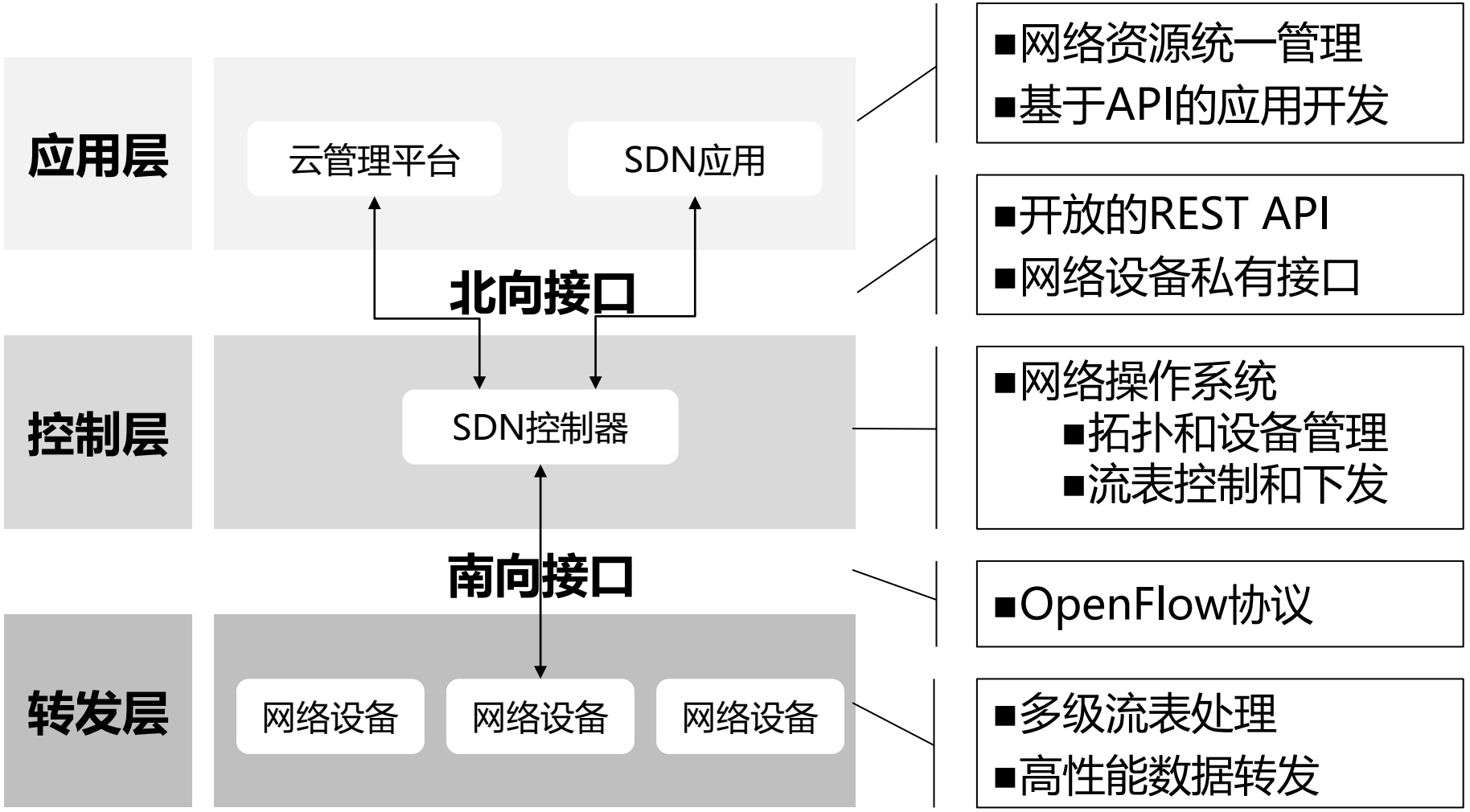


控制业务分离

- SDN可编程接口不是传统的网络管理，而是一种在应用与网元之间双向的、紧密联系的通信通道，可以实现传统网络不具备的网络快速优化能力。
- 应用开发人员使用控制器提供的API，实现网络自动化、网络编排和对网络的操控 。

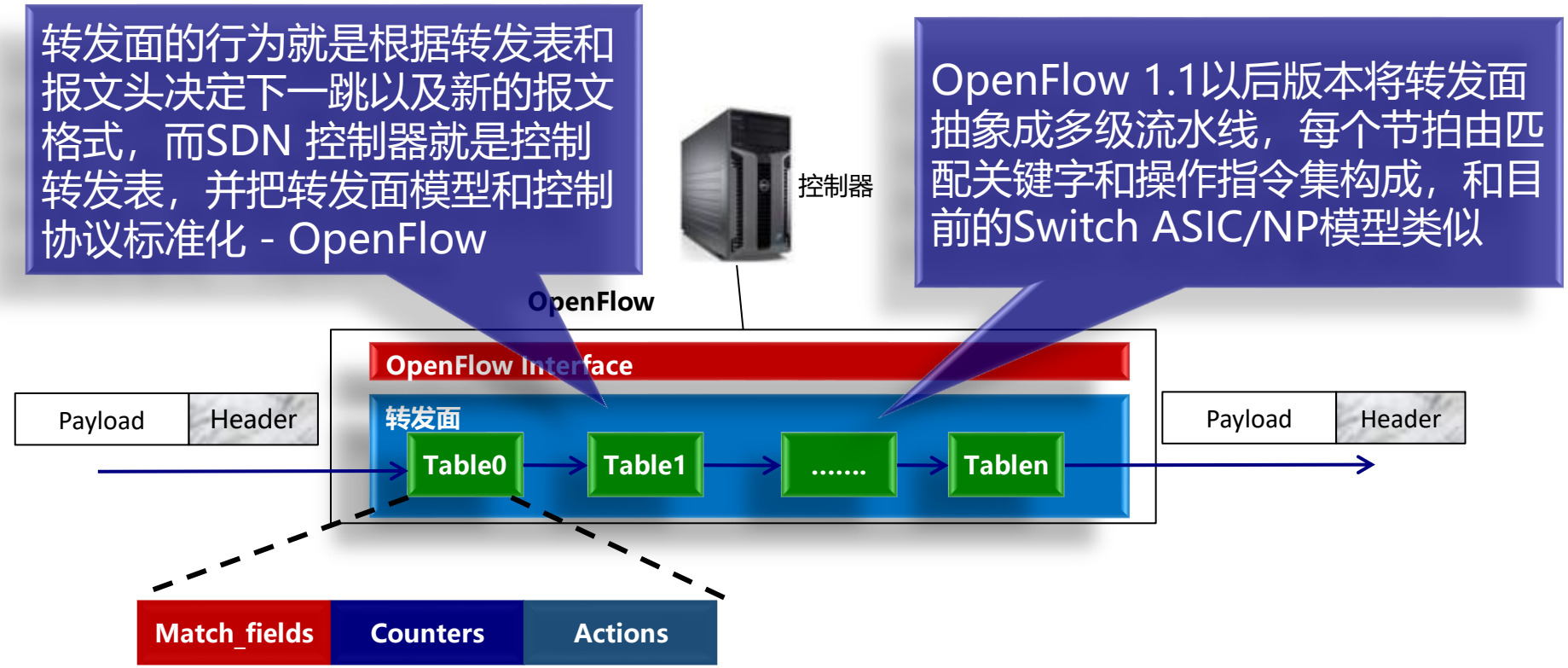


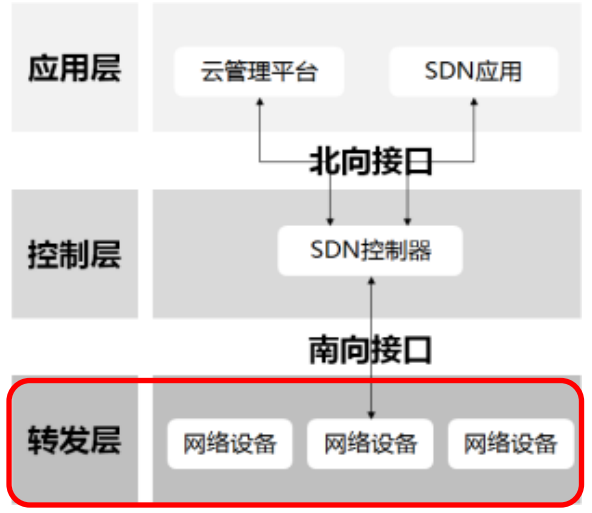
SDN关键技术体系


应用层
云管理平台
SDN应用
北向接口
控制层
SDN控制器
南向接口
转发层
网络设备
网络设备
网络设备
■网络资源统一管理
■基于API的应用开发
■开放的REST API
■网络设备私有接口
■网络操作系统
■拓扑和设备管理
■流表控制和下发
■OpenFlow协议
■多级流表处理
■高性能数据转发

1、SDN转发层关键技术

SDN转发层的关键技术是对转发面进行抽象建模。针对SDN转发面抽象模型，ONF标准组织提出并标准化了OpenFlow协议，在该协议中转发面设备被抽象为一个由多级流表(Flow Table)驱动的转发模型。



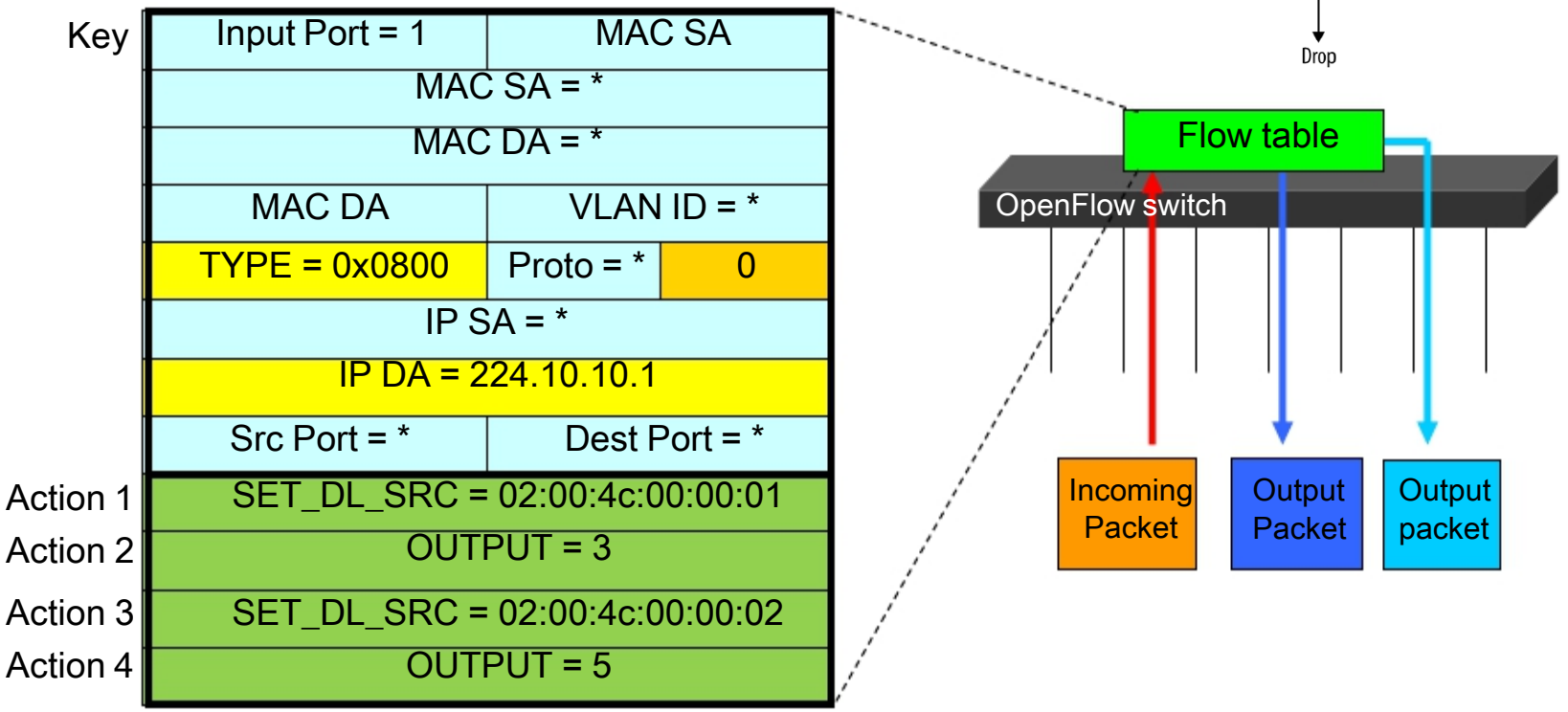


2、SDN南向接口关键技术

Openflow流表示例：任意种匹配组合

■OpenFlow只是实现网络数据平面可编程的一种方式，其它各种用于南向接口的技术/协议也在不同程度上可以实现类似的功能。

组播multicast



与现有数据网络兼容：Ethernet Switching，IP Routing，Application Firewall等

打通L2~L4形成不同维度的信息组合：Flow Switching，VLAN + App，Port + Ethernet + IP等

3、SDN控制层关键技术

- SDN控制层的关键是SDN 控制器，也可以称为网络操作系统（NOS）或网络控制器。

- 网络的所有智能、核心均在SDN 控制器中，由SDN控制器对转发面进行转发策略的调度和管理，通过无智能的快速转发面设备，支持运行在SDN控制器之上的不同业务。

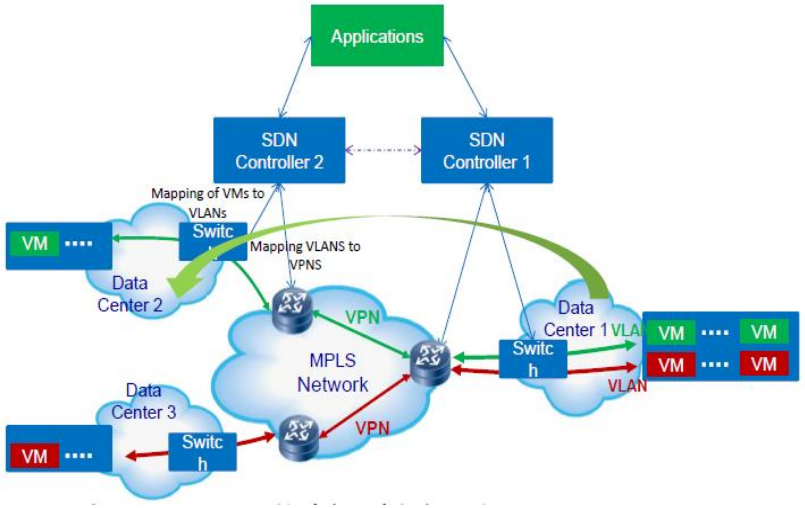
4、SDN北向接口关键技术

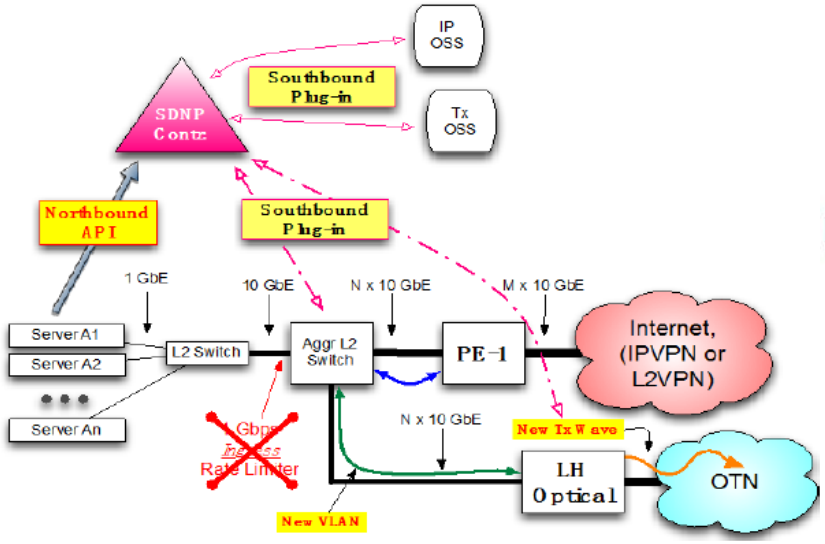
- SDN控制层将网络能力封装为开放的REST API，供上层业务调用
  - ONF当前只定义了OpenFlow作为南向API，在北向接口的制定方面，ONF也做出了重要的贡献。2013年10月，ONF成立北向接口工作组（North Bound Interface Working Group），致力于建立SDN中北向接口的原型及其信息模型，为应用开发者设计一个统一的层次清晰功能完善的北向接口。
  - 以Floodlight的北向API集合为例，包含了交换机状态采集、静态流表推送、防火墙策略等多种类型的接口

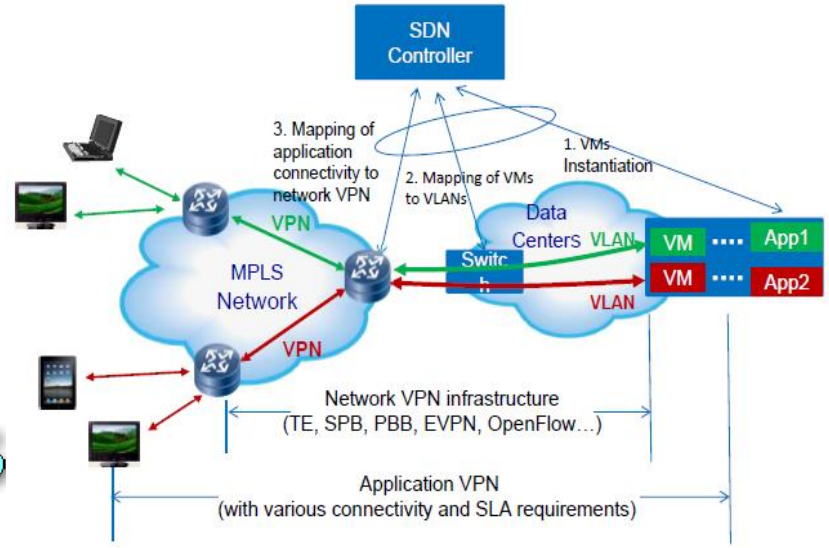
URI	方法	描述	参数
/wm/core/switch/all/<statType>/json	GET	获取跨交换机的聚合状态	statType: port, queue, flow, aggregate, desc, table, features
/wm/core/switch/<switchId>/<statType>/json	GET	获取每台交换机的工作状态	switchId: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) statType: port, queue, flow, aggregate, desc, table, features
/wm/core/counter/<counterTitle>/json	GET	获取控制器所辖交换机的全部流量计量信息	counterTitle: "all" or something of the form DPID_Port#OFEventL3/4_Type

5、SDN应用层关键技术

- 利用开放的北向API，调用网络能力
  - 编程加速网络创新业务实现
  - 制定灵活的配置和管控策略
  - 定义高级别的安全性和可靠性







通过SDN控制器协助虚拟机迁移

通过SDN控制器实现按需带宽调整

通过SDN控制器实现应用到VPN的映射

SDN的特征

集中控制

✓集中控制使得全局优化成为可能，比如流量工程、负载均衡
✓集中控制使得整个网络可以当作一台设备进行维护，设备零配置即插即用，大大降低运维成本

开放接口

✓应用和网络的无缝集成，应用告诉网络如何运行才能更好地满足应用的需求，比如业务的带宽、时延需求，计费对路由的影响等
✓用户可以自行开发网络新功能，加快新功能面世周期
✓理论上NOS和转发硬件的开放标准接口使得硬件完全PC化

网络虚拟化

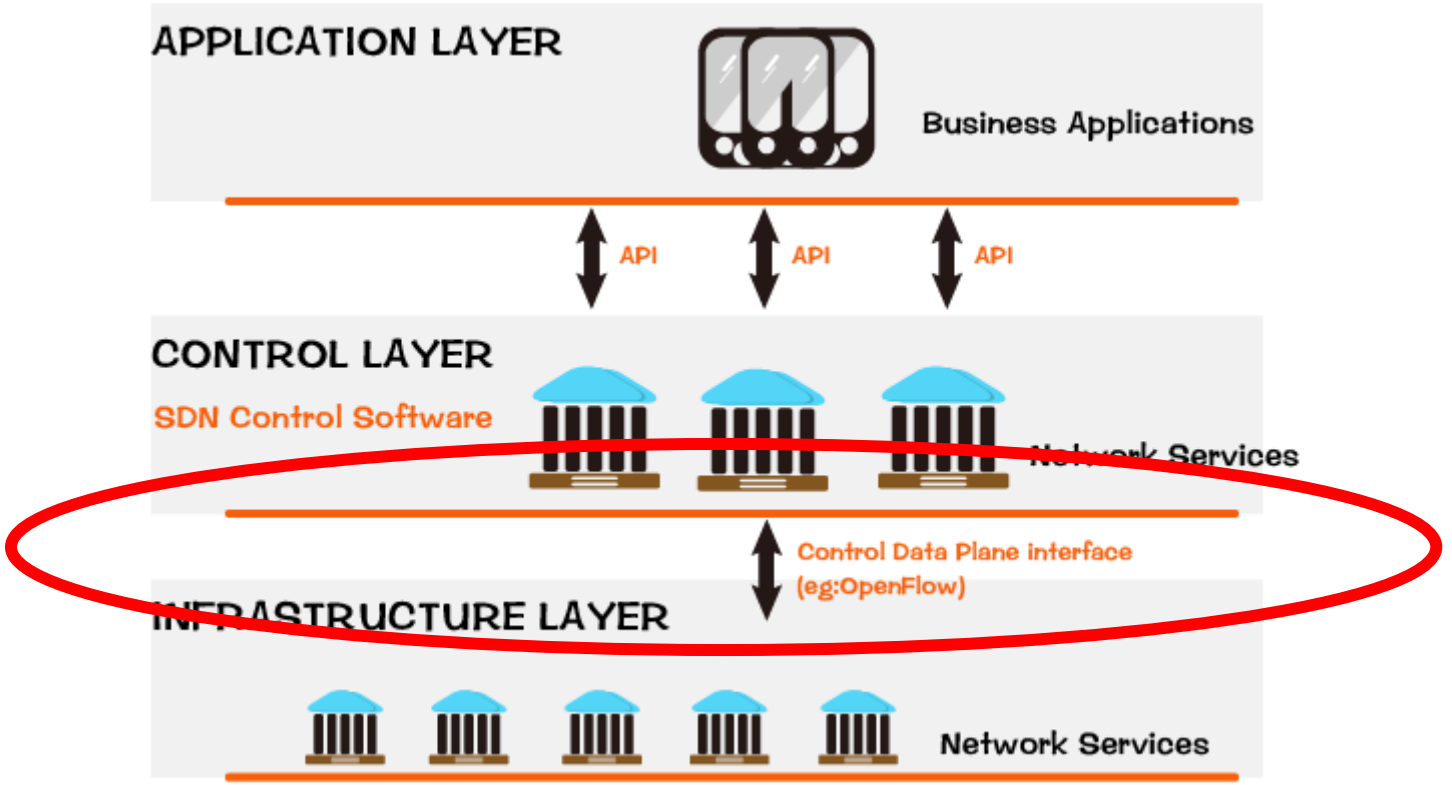
✓逻辑网络和物理网络的分离，逻辑网络可以根据业务需要配置、迁移，不受物理位置的限制
✓多租户支持，每个租户可以自行定义带宽需求和私有编址

- 架构角度：控制平面与数据平面分离，逻辑集中管理（集中至控制器）。
- 业务角度：通过控制器，网络资源被抽象成服务，实现了应用程序与网络设备操作系统的解耦。应用看到的是网络服务。
- 运营角度：网络可以通过编程的方式来访问，从而实现应用程序对网络的直接影响，易于实现网络优化。

SDN南向协议—Openflow

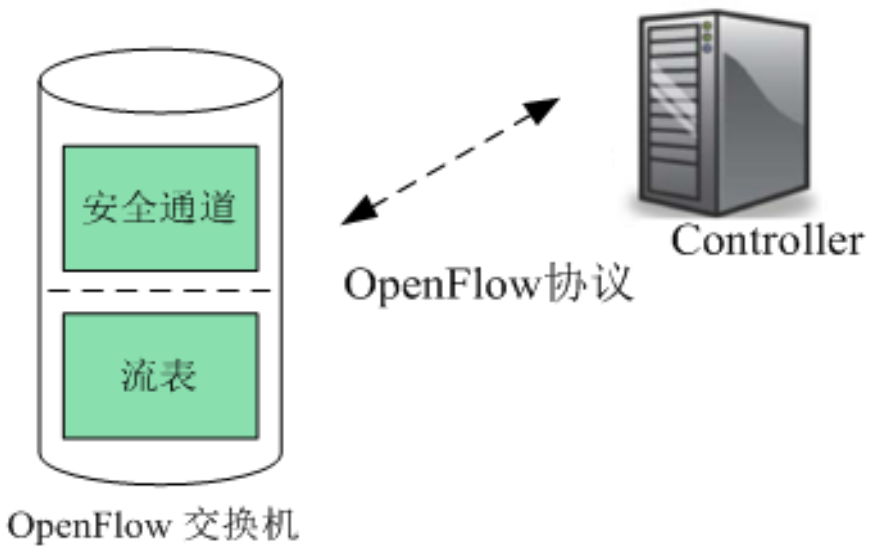
SDN南向协议分类

- 狭义SDN南向协议：
  - OpenFlow
- 广义SDN南向协议：
  - OF-Config
  - NETCONF
  - XMPP
  - PCEP
  - ForCES
  - OpFlex
- 完全可编程南向协议：
  - P4 (programming protocol-independent packet processors) 协议无关数据包处理编程语言



OpenFlow

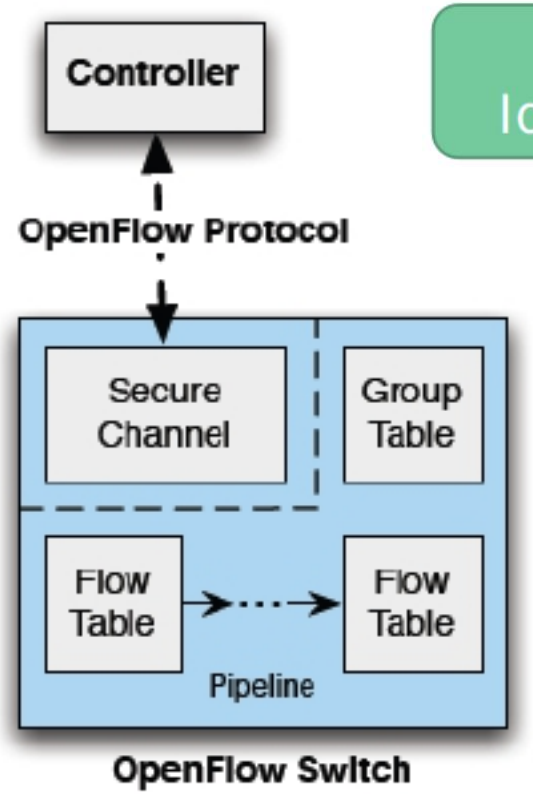
- SDN南向接口的关键技术是转发面开放协议，该协议允许网络控制器控制交换机的配置以及相关转发行为。
- Openflow是ONF定义的一个转发面控制协议，将转发面抽象为一个由多级流表组成的转发模型，网络控制器通过Openflow协议下发Openflow流表到交换机，从而定义和控制交换机的具体行为。
- OpenFlow是一整套软件应用程序接口，控制器可以与支持OpenFlow 的交换机沟通配置信息，决定数据转发平面的转发表，控制器与交换机间的通信通过SSL 加密传输。

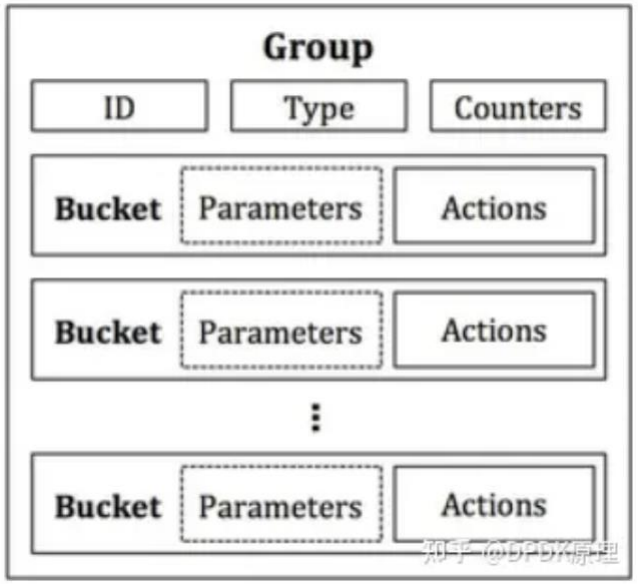


硬件OpenFlow交换机厂商

Vendor	Model / Series	Version
Arista	7050, 7150, 7300, 7500	1.0
Brocade	ICX, VDX	1.3
Brocade	MLXe, CER, CES	1.3
Brocade	NetIron XMR	1.3
Extreme Networks	BlackDiamond 8000/X8, Summit X670	1.0
HP	2029, 3500/3500yl, 3800, 5400zl	1.0, 1.3
IBM	Programmable Network Controller, RackSwitches G8264, G8264T, G8332, G8052, G8316	1.0
Juniper	EX, MX	1.0
NEC	PF5240, PF5820, PF1000	1.0
NEC	ProgrammableFlow Network Controller PF 6800	1.0, 1.3
Pica-8	P-3290, P-3295, P-3780. P-3920	1.4

Openflow 方案模型





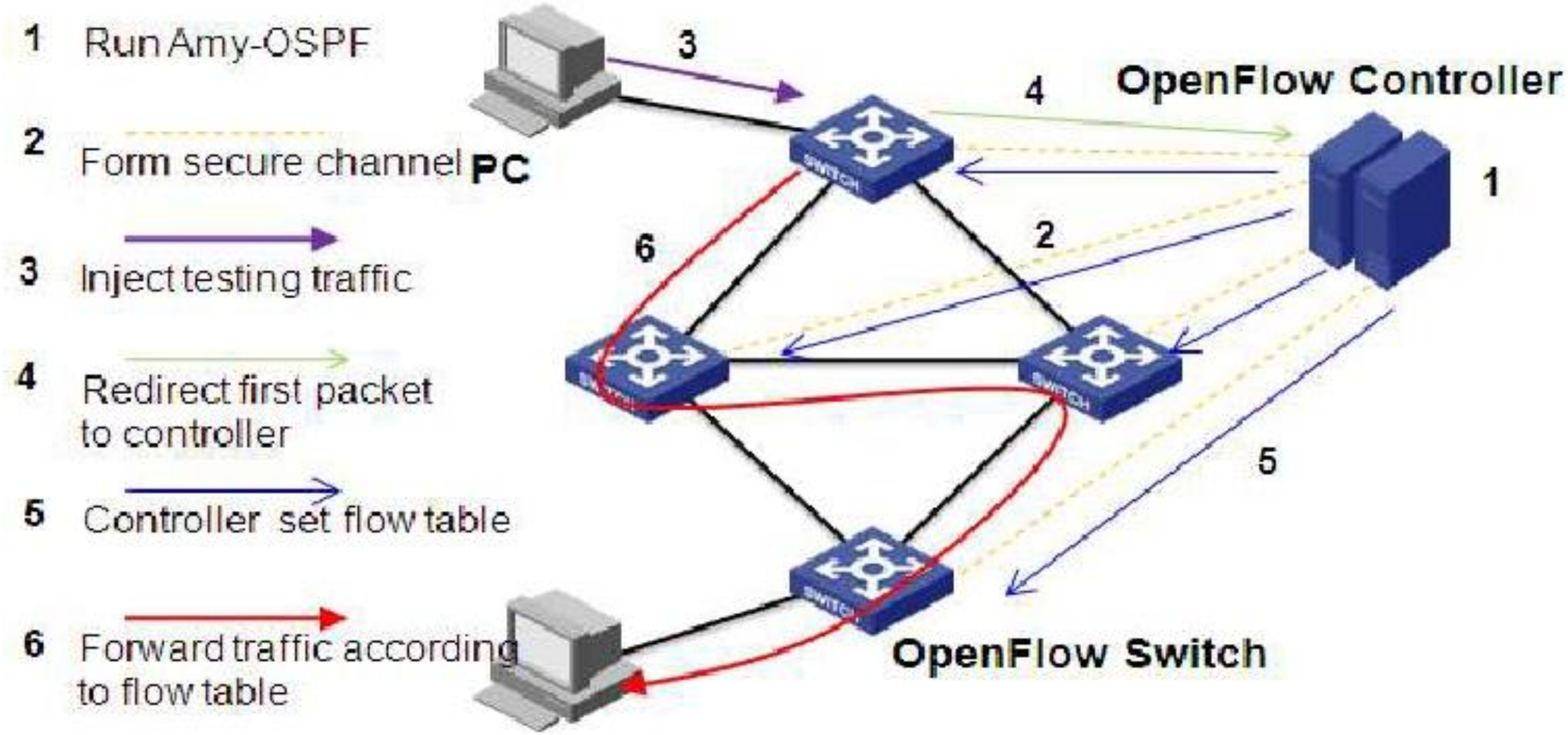
- 节省流表空间
- 数据包复制
- 容错能力
- 负载分流

- 安全通道：Controller通过安全信道向OpenFlow Switch下发命令和接收消息
- 组表 Group Table：用一个Group 使 OpenFlow可以支持额外的转发行为
- 流表 Flow Table：OpenFlow交换机的基本表项

从Controller 下发的流表（OF v1.0）

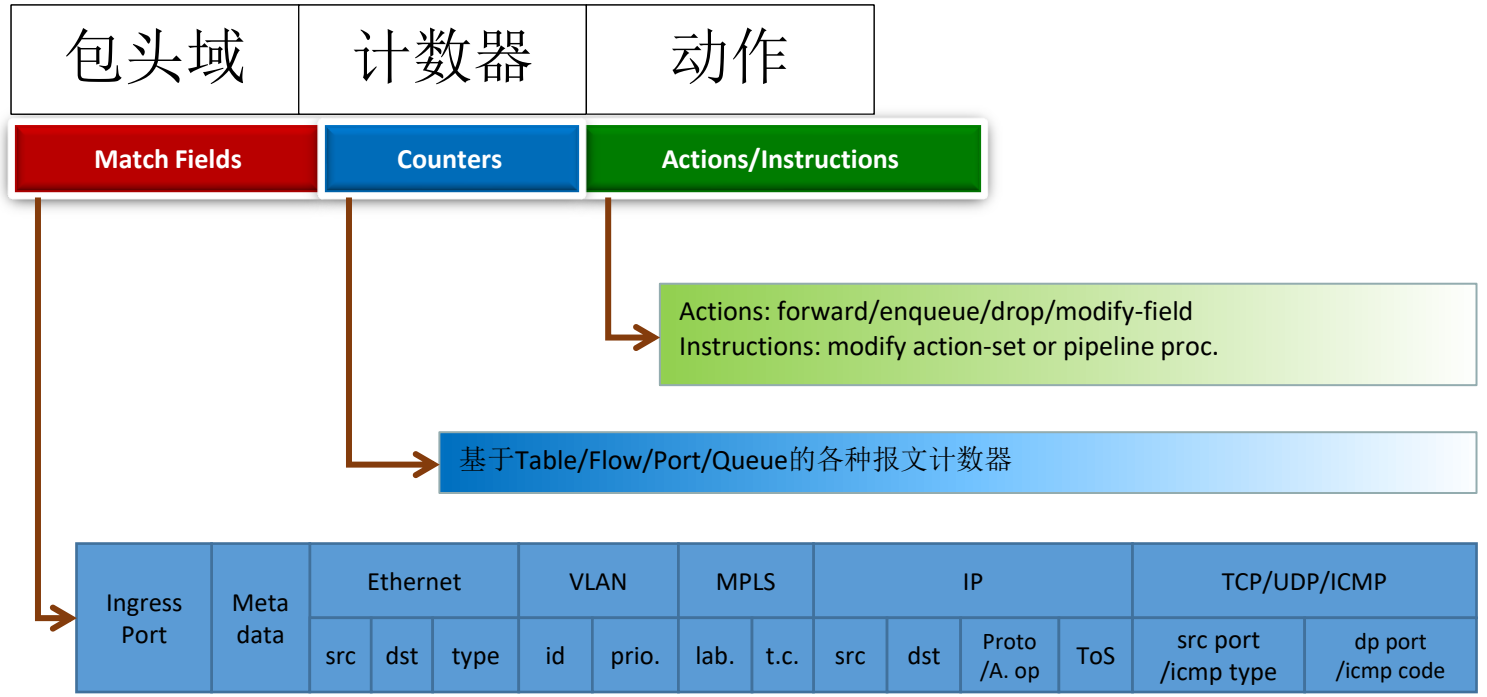
Ingress Port	Source MAC	Dest MAC	Ether Type	VLAN ID	VLAN Priority	IP SRC	IP DEST	IP Protocol	IP TOS	TCP/UDP SRC	TCP/UDP DEST		Action	Priority	Counter
*	3c:07:54:*	*	*	Switching	*	*	*	*	*	*	*		Fwd Port 10	100	
*	*	Routing		*	*	*	192.168.1.*	*	*	*	*		Fwd Port 12	100	
Port 1	*	*	Replication/SPAN		*	*	*	*	*	*	*		Fwd Port 14...24	100	
*	Firewall/Security			*	*	*	*	*	*	*	25		Drop	100	
*	*	*	*	Inspection	*	*	*	0x0800	*	*	*		Controller	100	
*	00:01:E7:*	*	*	Vlan10	*	Combinations		*	*	*	80		Fwd Port 8	200	
*	*	Multi-action ; NAT		*	*	*	192.168.1.*	*	*	*	80		Rewrite 10.1.2.3; Fwd port 9	200	
Local handling			*	*	*	*	10.*	*	*	*	*		Local	200	

Controller/OpenFlow 工作方式举例



OpenFlow 流表基本结构

■**流表是OpenFlow对网络设备的数据转发功能的抽象，**表项包括了网络中各个层次的网络配置信息

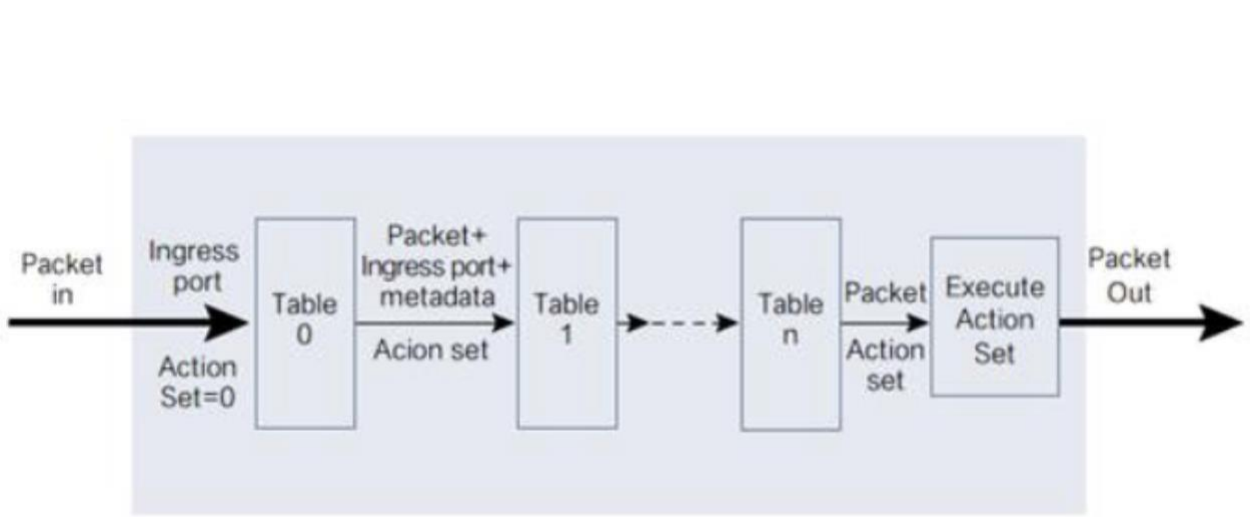


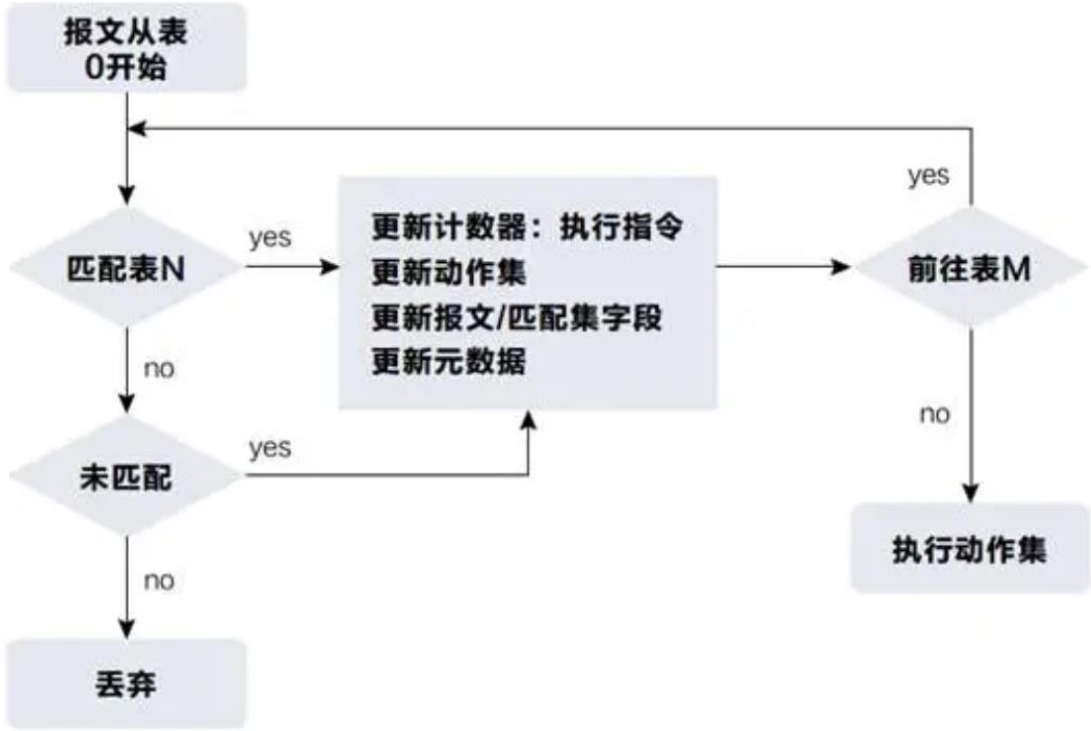
– **包头域**：用于对交换机接收到的数据包的包头内容进行匹配

– **计数器**：对匹配成功的包进行计数，用于统计数据流量相关信息，可以针对交换机中的每张流表、每个数据流、每个设备端口、每个转发队列进行维护

– **动作指令（action/Instructions）**：用于指示交换机在收到匹配数据包后如何对其进行处理（修改action set或者流水线处理）。OpenFlow交换机缺少控制平面的能力，需要用动作详细说明交换机将要对数据包所做的处理

从OpenFlow v1.1开始引入了多级流表和流水线处理机制，当报文进入交换机后，从序号最小的流表开始依次匹配，报文通过跳转指令跳转至后续某一流表继续进行匹配，这样就构成了一条流水线。





当报文进入Switch后，从最小的Flow Table开始依次匹配。当报文成功匹配一条Flow Entry后，首先更新该Flow Entry对应的统计数据（如成功匹配数据包总数目和总字节数等），然后根据Flow Table中的指令进行相应操作，比如跳转至后续某一Flow Table继续处理，修改或者立即执行该数据包对应的Action Set等。当数据包已经处于最后一个Flow Table时，其对应的Action Set中的所有Action将被执行，包括转发至某一端口，修改数据包某一字段，丢弃数据包等。具体实现时，OpenFlow交换机还需要对匹配表项次数进行计数、更新匹配集和元数据等操作。

OpenFlow 流表包头域

- **用于匹配交换机接收到的数据包的包头内容，OpenFlow 1.1及后续版本将“包头域”更名为“匹配域”**
  - OpenFlow 1.0包头域包含12个元组（tuple）
    - 涵盖OSI网络模型中第二至第四层的网络配置信息
    - 每一个元组中的数值可以是一个确定的值或者是“ANY”

入端口	源MAC地址	目的MAC地址	以太网类型	VLAN ID	VLAN优先级	源IP地址	目的IP地址	IP协议	IP TOS位	TCP/UDP源端口	TCP/UDP目的端口
Ingress Port	Ether Source	Ether Des	Ether Type	VLAN ID	VLAN Priority	IP Src	IP Dst	IP Proto	IP ToS bits	TCP/UDP Src Port	TCP/UDP Dst Port

用形如<header: actions, counters>的流表表示：

若输入分组匹配到特定的header，计数器被更新，相应的动作被执行。
若分组匹配多个表项，优先级最高的表项被选中。
若输入分组没有匹配任何一个表项，分组被转发给控制器。

Openflow流表动作

- 用于指示交换机在收到匹配的数据包后应该如何对其进行处理
- 必备动作(Required Actions) vs. 可选动作(Optional Actions)
  - 必备动作：需要所有OpenFlow交换机默认支持
  - 可选动作：需要交换机告知控制器它所能支持的动作种类

类型	名称	说明
必备动作	转发 (Forward)	交换机必须支持将数据包转发给设备的物理端口及一个或多个虚拟端口，实现组播、多路径转发、负载均衡等功能： ➢ALL：转发给所有出端口，但不包括入端口 ➢CONTROLLER：封装数据包并转发给控制器 ➢LOCAL：转发给本地的网络栈 ➢TABLE：对packet_out消息执行流表的动作 ➢IN_PORT：从入端口发出
	丢弃 (Drop)	交换机对没有明确指明处理动作的流表项，将会对与其所匹配的所有数据包进行默认的丢弃处理
可选动作	转发 (Forward)	交换机可选支持将数据包转发给如下的虚拟端口： ➢NORMAL：利用交换机所能支持的传统转发机制（例如二层的MAC、VLAN信息或者三层的IP信息）处理数据包 ➢FLOOD：遵照最小生成树从设备出端口洪泛发出，但不包括入端口
	排队 (Enqueue)	交换机将数据包转发到某个出端口对应的转发队列中，便于提供QOS支持
	修改域 (Modify-Field)	交换机修改数据包的包头内容，各个字段值、封装/去封装，具体可以包括： ➢设置VLAN ID、VLAN优先级，剥离VLAN头 ➢修改源MAC地址、目的MAC地址 ➢修改源IPv4地址、目的IPv4地址、ToS位 ➢修改源TCP/IP端口、目的TCP/IP端口

Openflow端口

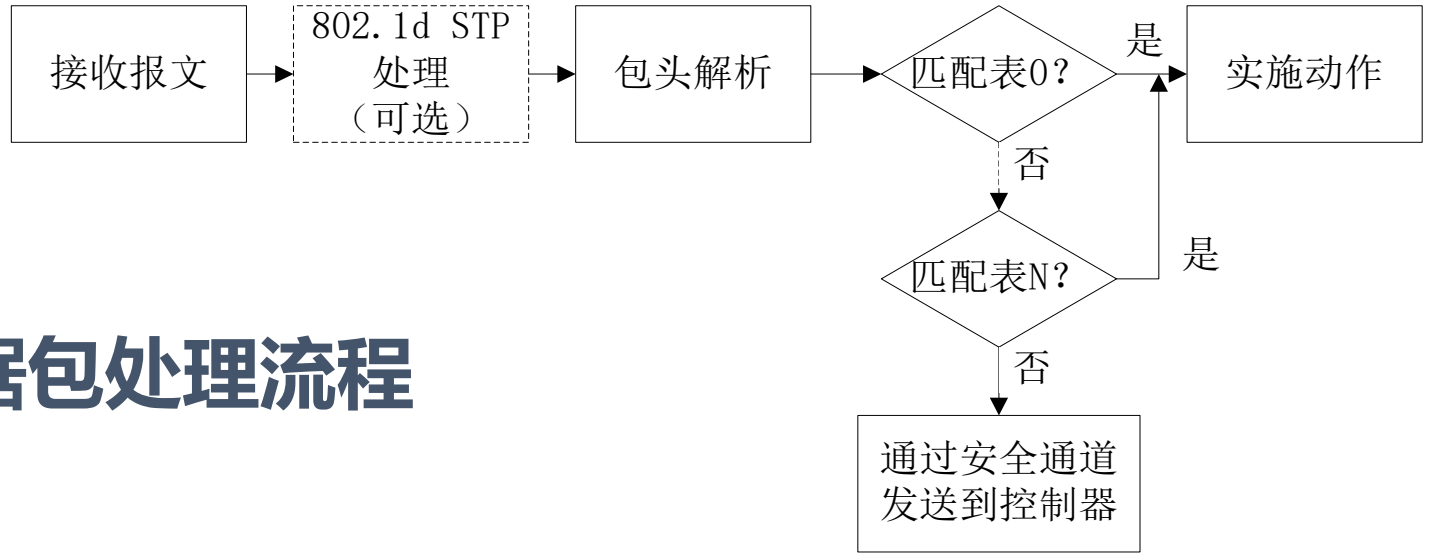
- OpenFlow 1.2及后续版本提出OpenFlow端口的概念，是OpenFlow进程和网络之间传递数据包的网络接口：
  - **物理端口**：交换机定义的端口，与OpenFlow交换机上的硬件接口一一对应。在某些部署中，OpenFlow交换机可以实现交换机的硬件虚拟化。在此情况下，一个OpenFlow物理端口可以对应交换机硬件接口的一个虚拟接口
  - **逻辑端口**：交换机定义的端口，但并不直接对应一个交换机的硬件接口。逻辑端口是更高层次的抽象概念，可以是交换机中定义的其它一些端口（例如链路聚合组、隧道、环回接口等）。
  - **保留端口**：用于特定的转发动作，如发送到控制器、洪泛，或使用非OpenFlow的方法转发，如使用传统交换机的处理过程。

OpenFlow定义了8种端口

类型	名称	说明
必备	ALL	表示所有端口均可用于转发指定数据包。当其被用作输出端口时，数据包被复制后发送到所有的标准端口，但不包括数据包的入端口及被配置为OFPPC_NO_FWD的端口
	CONTROLLER	表示到OpenFlow控制器的控制通道，它可以用作一个入端口或作为一个出端口。当用作一个出端口时，封装数据包中为数据包消息，并使用OpenFlow协议发送。当用作一个入端口时，确认来自控制器的数据包
	TABLE	表示OpenFlow流水线的开始。这个端口仅在输出行为的时候有效，此时OpenFlow交换机提交报文给第一流表使数据包可以通过OpenFlow流水线处理
	IN PORT	代表数据包进入端口。只有一种情况用于输出端口，即从入端口发送数据包
	ANY	某些OpenFlow命令中的特定值，用在没有指定端口的（端口通配符）情形，不能用于入端口和出端口
可选	LOCAL	表示交换机的本地网络堆栈和管理堆栈。可以用作一个入端口或者一个出端口。该端口使得远程实体可以与交换机通过OpenFlow网络交互，而不再需要单独的控制网络。通过配置一组合适的默认流表项，该端口可用于实现一个带内控制器的连接
	NORMAL	代表传统的非OpenFlow流水线处理。仅可用作普通流水线的输出端口。如果交换机不能从OpenFlow流水线转发数据包到普通流水线，必须表明它不支持这一动作
	FLOOD	表示使用普通流水线处理洪泛过程。可用于作为一个输出端口，除入端口或OFPPS_BLOCKED状态的端口外，可以将数据包发往其他所有标准端口。交换机也可以通过数据包的VLAN ID选择哪些端口洪泛

Openflow 交换机

- **根据交换机的应用场景及其所能够支持的流表动作类型，OpenFlow交换机可以被分为**
  - **OpenFlow专用交换机（OpenFlow-only）**
    - 只支持OpenFlow协议
  - **OpenFlow使能交换机（OpenFlow-enabled）**
    - OpenFlow 1.1及后续版本将其更名为“OpenFlow-hybrid”
    - 考虑了OpenFlow交换机与传统交换机混合组网时可能遇到的协议栈不兼容问题，能同时运行OpenFlow协议和传统的二层/三层协议栈
    - 支持OpenFlow可选转发动作中的NORMAL动作。



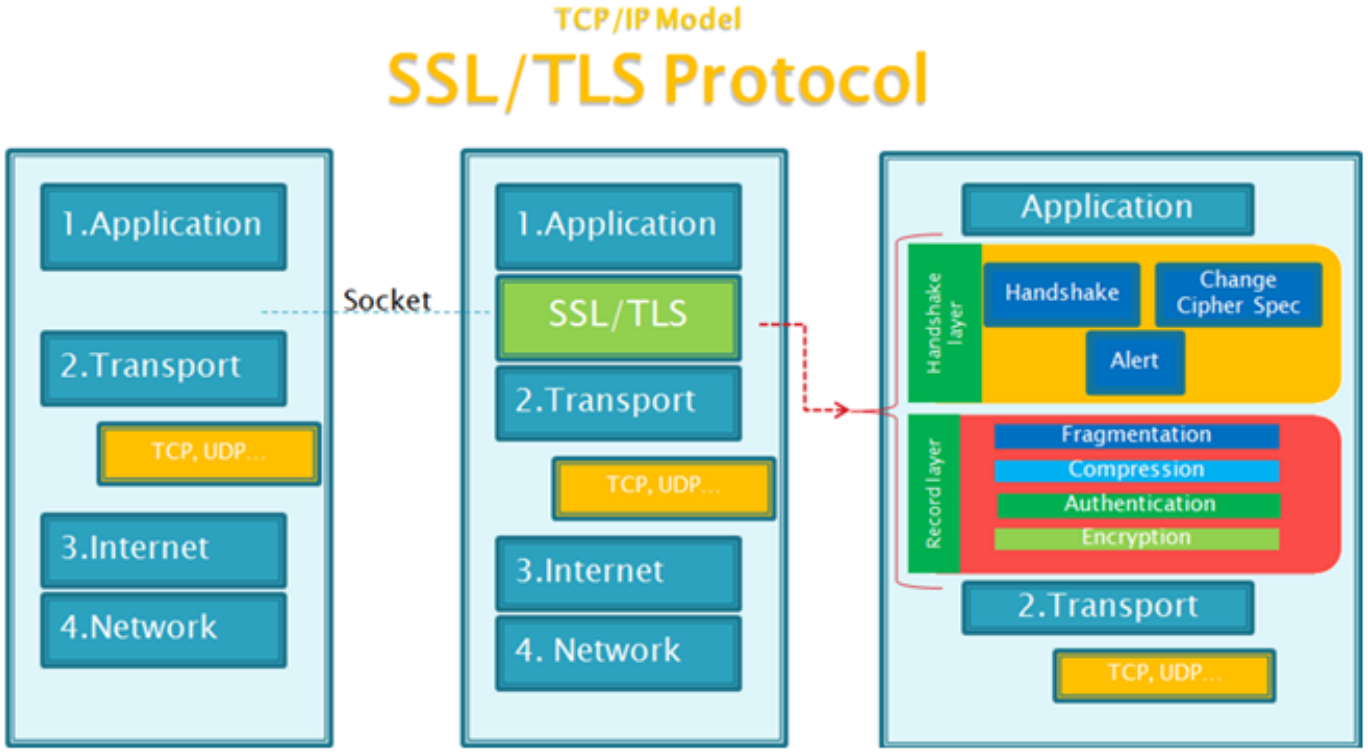
OpenFlow交换机数据包处理流程

Openflow 安全通道

■**OpenFlow的集中控制架构对控制器与交换机之间信息传送通道提出了极高的要求**

–控制器与交换机之间的数据通路必须确保安全

–采用TLS（Transport Layer Security）技术

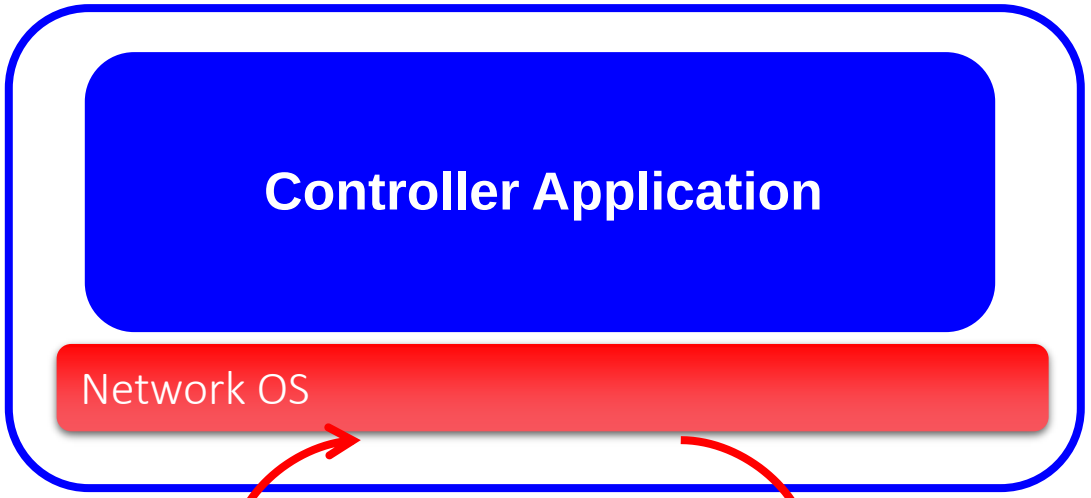


认证过程

■交换机启动时，尝试连接到控制器的**6633** TCP 端口，进而双方通过交换证书进行认证

■交换机和控制器相互交换证书进行认证，证书用特定站点的私钥签名，用户必须能够对每个交换机进行配置，用其中的一个证书对控制器进行认证（控制器证书），用另一个证书向控制器提供交换机认证（交换机证书）。

Openflow 协议消息



Events from switches
Topology changes,
Traffic statistics,
Arriving packets

Commands to switches
(Un)install rules,
Query statistics,
Send packets

- OpenFlow协议是用来描述控制器和OpenFlow交换机之间数据交互所用信息的接口标准
  - **controller-to-switch**：由控制器发起，用来管理或获取OpenFlow交换机的状态
  - **asynchronous（异步）**：由OpenFlow交换机发起，用来将网络事件或交换机状态变化更新到控制器
  - **symmetric（对称）**：由交换机或控制器发起

Openflow 协议消息类型

- Controller-to-Switch控制器至交换机消息
  - Controller用来对Switch进行状态查询和修改配置等操作，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 用来发生数据包
    - Barrier 阻塞消息
- Asynchronous异步消息
  - Switch发送给Controller，用来通知Switch上发生的某些异步事件的消息。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
- Symmetric 对称消息
  - 双向操作，用来建立连接，检测对方是否在线等
    - 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

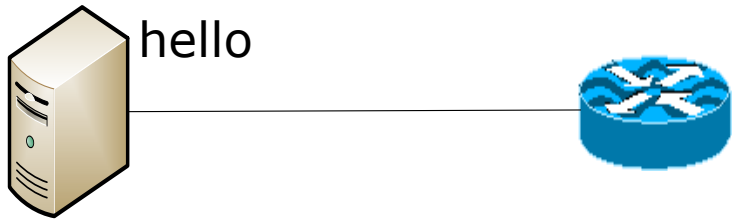
Openflow 协议主要交互过程

1.建立连接(Switch registration)

- OFPT_HELLO

控制器

Server： TCP6653端口



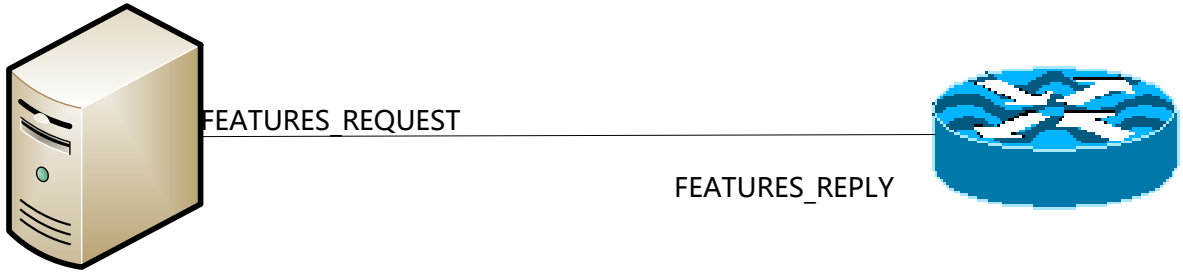
- 目的：协议协商
- 内容：双方都支持的最高版本
- 结果：使用双方都支持的最低版本
- 成功：建立连接
- 失败：产生一个错误消息类型
  OFPT_ERROR，终止连接

```
    35 0.625708        127.0.0.1          127.0.0.1          OFP          16
    37 0.941364        ::                 ff02::1:ff4d:f1e9  OFP+ICMPv    16
▸ Frame 12: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:
▸ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▸ Transmission Control Protocol, Src Port: 35306 (35306), Dst Port: 6633 (6633), Seq: 1,
▾ OpenFlow Protocol
  ▾ Header
      Version: 0x01
      Type: Hello (SM) (0)
      Length: 8
      Transaction ID: 3
```

- OFPT_ECHO
- 分类：对称信息 OFPT_ECHO_REQUEST, OFPT_ECHO_REPLY
- 作用：查询连接状态，确保通信通畅。
- 当没有其他的数据包进行交换时，controller会定期循环给sw发送OFPT_ECHO_REQUEST。

2.获取交换机特性（Features）信息

- OFPT_FEATURES
- 当sw跟controller完成连接之后，控制器会向交换机下发OFPT_FEATURES_REQUEST的数据包，目的是请求交换机的信息。
- 发送时间：连接建立完成之后
- 发送数据：OFPT_FEATURES_REQUEST
- 对称数据：OFPT_FEATURES_REPLY
- 目的：获取交换机的信息



FEATURES_REQUEST:请求交换机的信息
从返回的FEATURES_REPLY来看，交换机上有4个端口，其中local port用来连接控制器所用

```
▼ Physical Port
    Port #: Local  (local openflow "port")
    MAC Address: 26:b8:7d:ff:7d:44 (26:b8:7d:ff:7d:44)
    Port Name: s1
  ▶ Port Config Flags
  ▶ Port State Flags
  ▶ Port Current Flags
  ▶ Port Advertsied Flags
  ▶ Port Supported Flags
```

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
current, advertised,supported为端口物理属性

```
▼ OpenFlow Protocol
  ▶ Header
  ▼ Switch Features
      Datapath ID: 0x0000000000000001
      Max packets buffered: 256
      Number of Tables: 255
    ▶ Capabilities: 0x000000c7
    ▶ Actions: 0x00000fff
    ▼ Port Definitions
        # of Ports: 4
      ▶ Physical Port
      ▶ Physical Port
      ▶ Physical Port
      ▶ Phvsical Port
```

datapath_id 为交换机独一无二的ID号
buffered 为交换机可以同时缓存的最大数据包个数
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）
ports 为交换机的物理端口描述列表

3.配置交换机

通过SET_CONFIG消息来实现配置，两个属性：

第一个属性为flags，用来指示交换机如何处理IP分片数据包
第二个属性用来指示当一个交换机无法处理的数据包到达时，发给控制器的数据包的最大字节数。

可以看到控制器设置了交换机可以向它发送的新流字节数为128

```
▼ Header
    Version: 0x01
    Type: Set Config (CSM) (9)
    Length: 12
    Transaction ID: 3
▼ Switch Configuration
  ▼ Flags
      .... .... .... ..00 =   Handling of IP fragments: No special fragment handling (0)
    Max Bytes of New Flow to Send to Controller: 128
```

```
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
};
```

控制器用OFPT_SET_CONFIG消息设置交换机的配置参数，交换机对设置配置参数的请求不做出应答；控制器用OFPT_GET_CONFIG_REQUEST消息查询交换机的配置参数，交换机用OFPT_GET_CONFIG_REPLY消息响应查询配置的请求。

4.网络topo检测

通过LLDP数据包和广播包的混合使用来查询网络拓扑信息。

Packet_out消息——控制器命令交换机S3将LLDP数据包发送给所有的端口，OF交换机会执行相应的查找流表的操作。

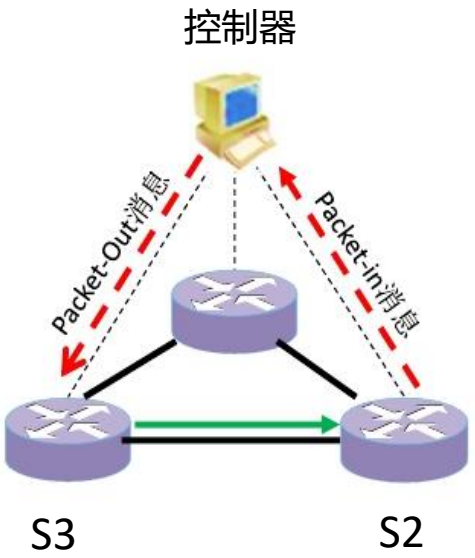
Packet_in消息——交换机将未匹配的数据包发往控制器，控制器将收到的LLDP数据包与发出的LLDP数据包进行比对；接着对S2进行同样的过程，则可以得到S3与S2之间的连接关系

是否继续保持连接

Topo检测完成后，交换机还需要向控制器发送echo_request消息，控制器回复echo_reply消息，考察连接是否保持

Switches		Controller（POX）
	←	Echo request（SM）
Echo reply（SM）	→	
Echo request（SM）	←	
	→	Echo reply（SM）

这个过程在POX控制器下每隔5秒中重复一次



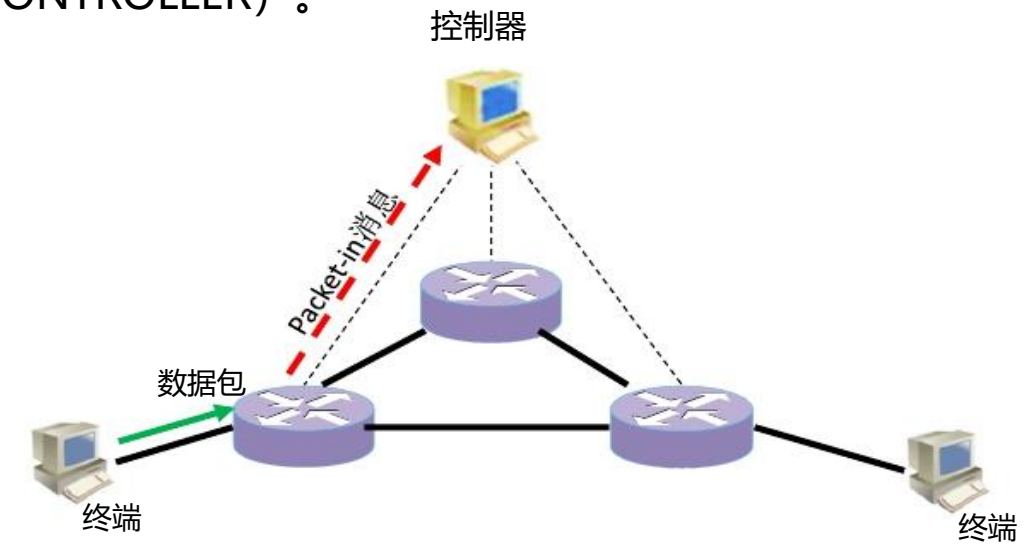
5.Packet-in事件

- OFPT_PACKET_IN
- 在控制器获取完交换机的特性之后，交换机开始处理数据。
- 对于进入交换机而没有匹配流表，不知道如何操作的数据包，交换机会将其封装在packet_in中发给controller。包含在packet_in中的数据可能是很多种类型，arp和icmp是最常见的类型。
- 产生packet_in的原因主要有两种。
- **OFPR_NO_MATCH**
- **OFPR_ACTION**
- 无法匹配的数据包会产生packet_in,action也可以指定将数据包发给packet_in,也就是说我们可以利用这一点，将需要的数据包发给控制器。

- packet_in事件之后，一般会触发两类事件：
- **packet_out**
- **flow_mod**
- 如果是广播包，如arp，控制器一般会将其包装起来，封装成packet_out数据包，将其发给交换机，让其flood,flood操作是将数据包往除去in_port以外的所有端口发送数据包。

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）。

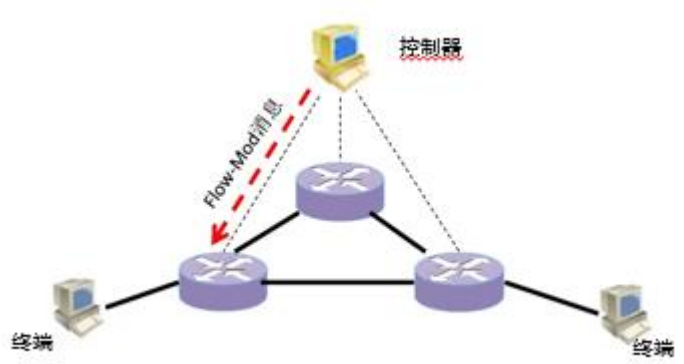


6.控制器配置流表（Flow-Mod）

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类型的消息——添加一条新的流表项
- DELETE类型消息——删除所有符合一定条件的流表项
- DELETE-STRICT类型消息——删除某一条指定的流表项
- MODIFY类型——修改所有符合一定条件的流表项
- MODIFY-STRICT类型——修改某一条指定的流表项




```
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

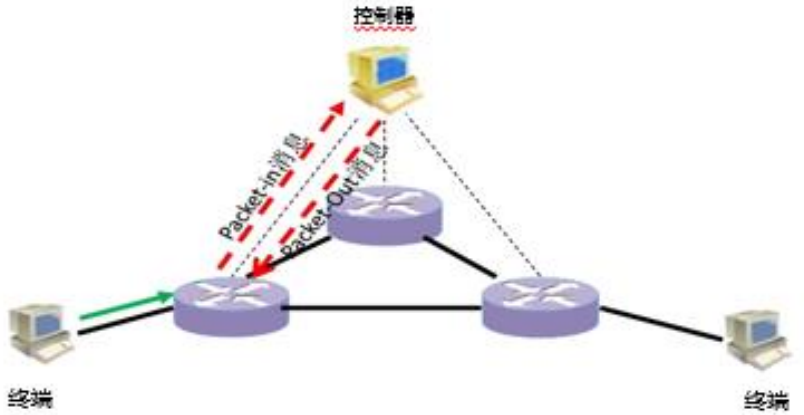
Priority为流表项的优先级，交换机优先匹配高优先级的流表项

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

7.交换机转发（Packet-Out）

- OFPT_PACKET_OUT
- 通过控制器发送交换机希望发送的数据



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），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Packet-Out消息，告诉交换机某一个数据包如何处理。

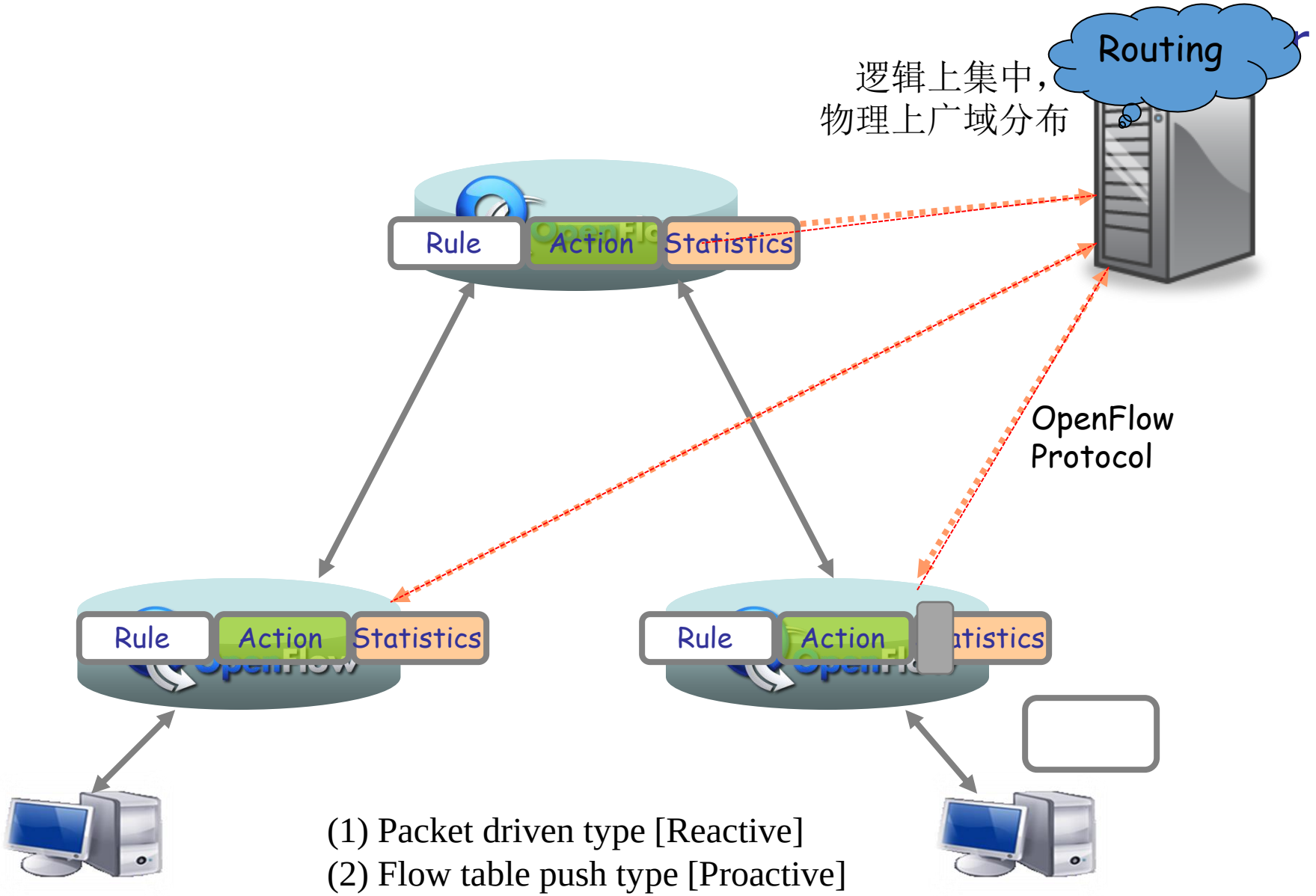
```
struct ofp_packet_out {
    struct ofp_header header;
    uint32_t buffer_id;           /* ID assigned by datapath (-1 if none). */
    uint16_t in_port;             /* Packet's input port (OFPP_NONE if none). */
    uint16_t actions_len;         /* Size of action array in bytes. */
    struct ofp_action_header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

In_port为Packet-Out消息提供额外的匹配信息

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

Openflow 转发原理示意



(1) Packet driven type [Reactive]
(2) Flow table push type [Proactive]

开源SDN控制器

主要开源控制器

1.OpenDaylight

OpenDaylight以开源社区为主导，使用Java语言实现的的开源框架，旨在推动创新实施以及软件定义网络透明化。面对SDN型网络，OpenDaylight作为项目核心，拥有一套模块化、可插拔且极为灵活的控制器，还包含一套模块合集，能够执行需要快速完成的网络任务。

相关网站：http://www.opendaylight.org/

2.ONOS

ONOS是由ON.Lab使用Java及Apache实现发布的首款开源的SDN网络操作系统，主要面向服务提供商和企业骨干网。ONOS的设计宗旨是满足网络需求实现可靠性强、性能好、灵活度高。此外，ONOS的北向接口抽象层和API支持简单的应用开发，而通过南向接口抽象层和接口则可以管控OpenFlow或者传统设备。

相关网站:http://onosproject.org/

3.Floodlight

Floodlight是由Big Switch Networks使用apache协议及Java语言开发的一款OpenFlow控制器，被用来与交换机、路由器、虚拟交换机及其他支持OpenFlow标准的设备一起工作。

相关网站:http://floodlight.openflowhub.org/

4. RYU

RYU由日本NTT公司负责设计研发的一款开源SDN控制器，同POX一样，也是完全由Python语言实现，使用者可以在Python语言的基础上实现自己的应用，采用Apache License开源协议标准，目前支持协议OpenFlow1.0、1.2、1.3，同时支持在OpenStack上的部署应用。提供逻辑上的集中化管理，通过提供API使网络管理更加方便。

相关网站:http://www.osrg.net/ryu/

5. POFController

POFController是由华为公司采用BSD/ Apache授权基于Java语言实现的OpenFlow控制器，提供了一个GUI管理界面，用于交换机的控制和配置。POF主要包含控制器和交换机两个原型文件，旨在提高OpenFlow的规范及支持无感知转发协议和数据包格式。

相关网站:http://www.poforwarding.org/

6. MUL

MUL是一个用c语言实现多线程架构的OpenFlow控制器，用来连接app的多种北向接口，目前支持openFlow的1.0、1.3版本，主要为性能和可靠性设计，是一款轻量级高效的控制器。支持一键安装、CTL命令行管理和WEB GUI。

相关网站:http://sourceforge.net/projects/mul/

7. POX

POX是由斯坦福使用Python语言开发的基于OpenFlow的一种控制器，是NOX的兄弟，它具有能将交换机送上来的协议包交给指定软件模块的功能。

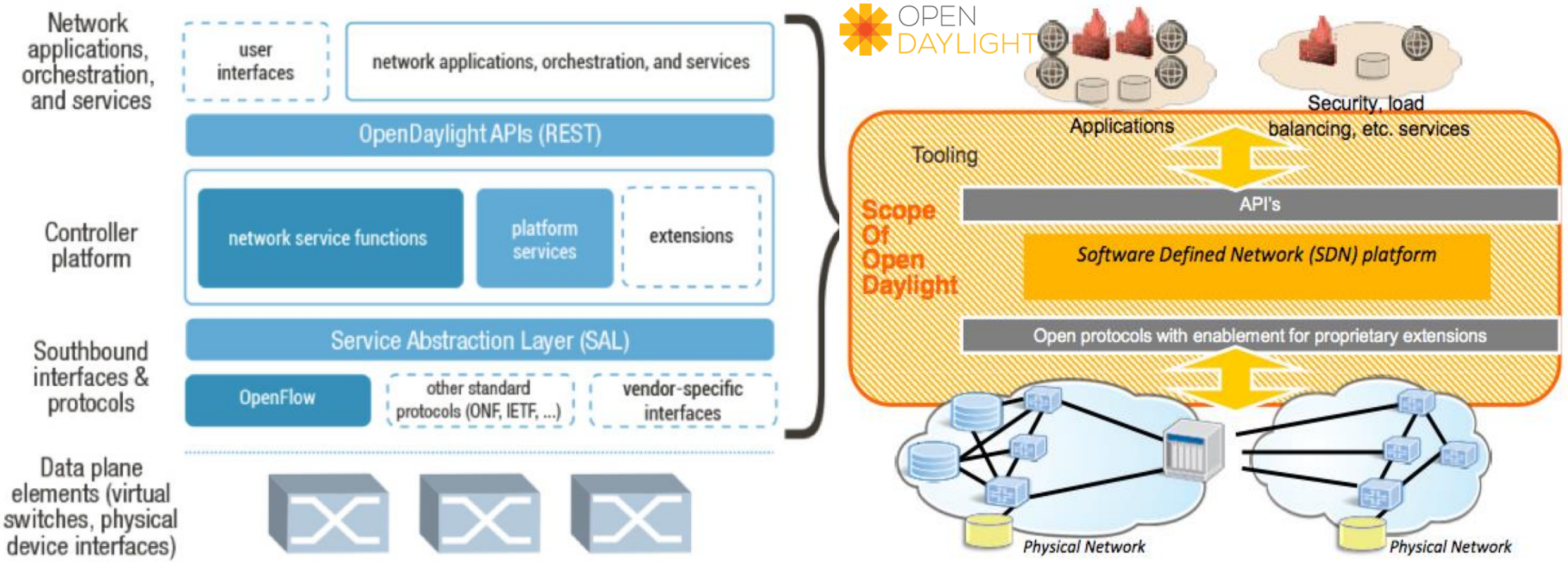
相关网站:http://www.noxrepo.org/support/about-pox/

8. NOX

NOX 是Nicira使用C语言开发的首个提供尽可能通用接口的 SDN 软件定义网络生态系统的控制器，也是用来构建网络控制应用的平台。

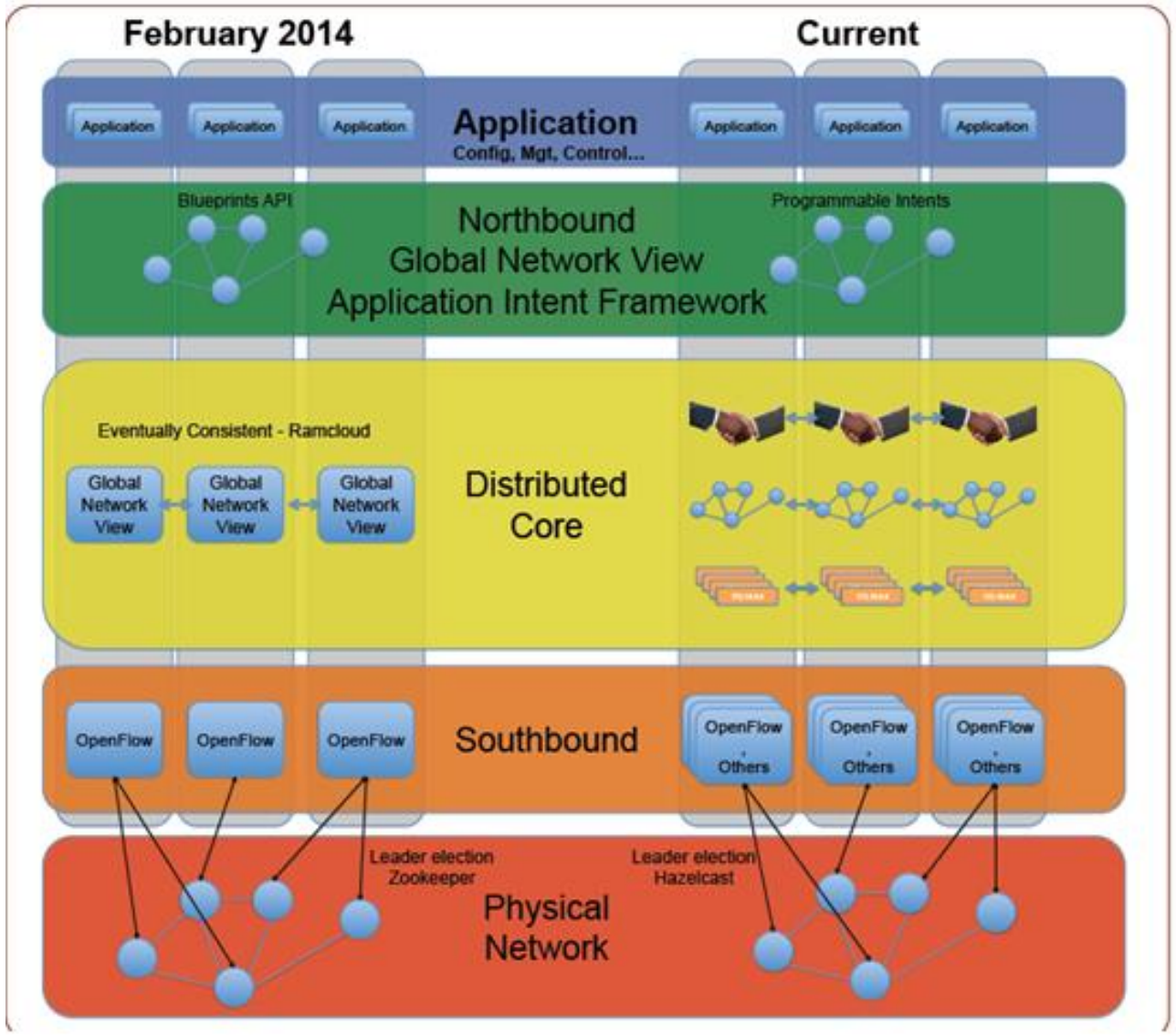
相关网站:http://www.noxrepo.org/nox/versions-downloads/

OpenDaylight



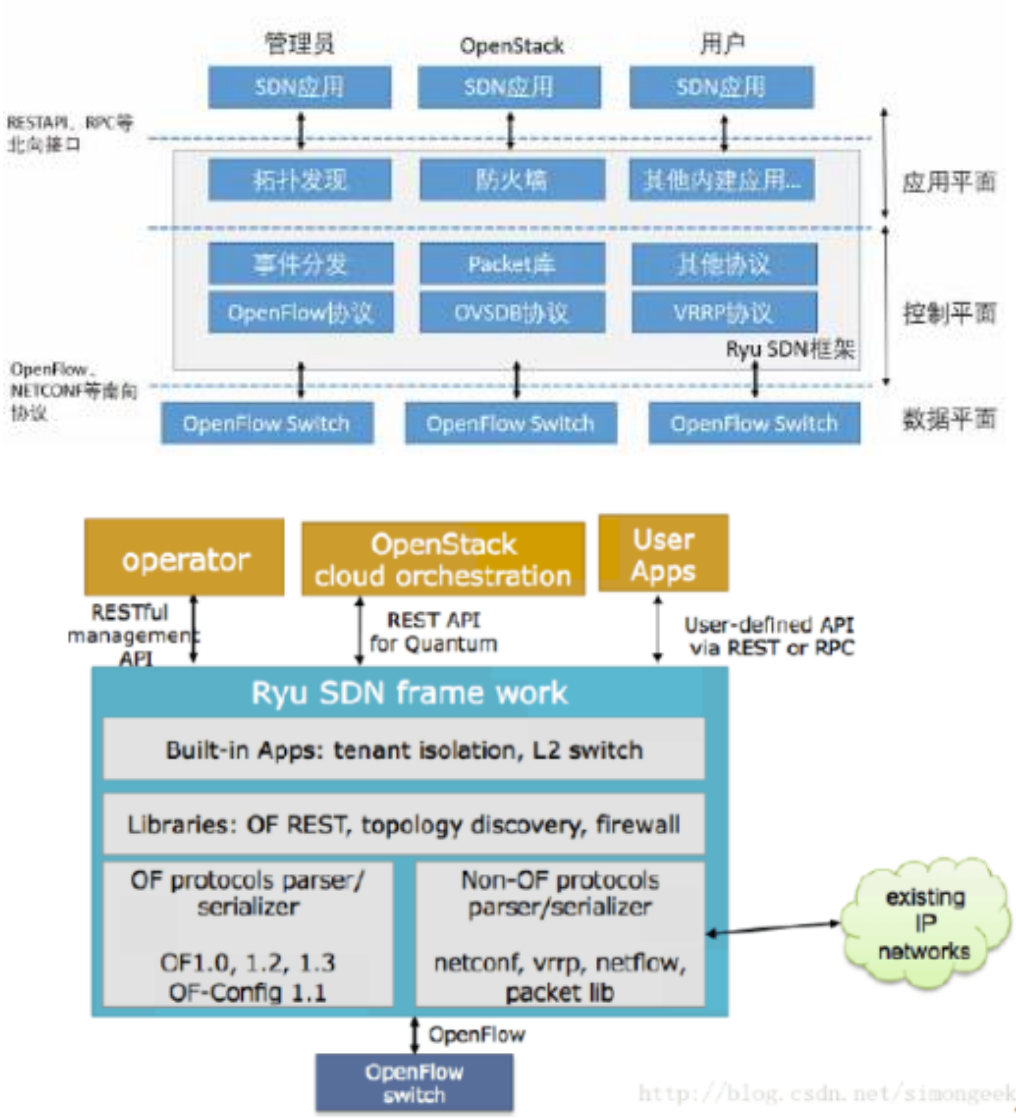
- 开放的北向API：包括REST/RESTCONF/NETCONF/AMQP
- 控制器平面：包括ODL root Parent、Controller等几十个工程
- 南向接口和协议插件：包括OpenFlow、NETCONF等

ONOS



- 分布式核心平台，提供高可扩展性、高可靠性以及高稳性能，实现运营商级SDN控制器平台特征。ONOS像集群一样运行，使SDN控制平台和服务提供商网络具有网页式敏捷度。
- 北向接口抽象层/APIs，图像化界面和应用提供更加友好的控制、管理和配置服务，抽象层也是实现网页式敏捷度的重要因素。
- 南向接口抽象层/APIs，可插拔式南向接口协议可以控制OpenFlow设备和传统设备。南向接口抽象层隔离ONOS核心平台和底层设备，屏蔽底层设备和协议的差异性。且南向接口是从传统设备向OpenFlow白牌设备迁移的关键。
- 软件模块化，让ONOS像软件操作系统一样，便于社区开发者和服务提供商开发、调试、维护和升级。

RYU



- 网络拓扑发现和管理：RYU可以发现和管理SDN网络中的拓扑结构，包括交换机、主机、链路和路径等信息。
- 网络流量控制和管理：RYU可以控制网络流量的转发和策略，支持流表、QoS、ACL等功能，可以实现网络流量的精确控制和管理。
- 网络安全监控：RYU可以对网络流量进行监控和分析，能够识别和处理恶意流量和攻击行为，提高网络安全性。
- 网络服务质量（QoS）保障：RYU可以实现基于流量的服务质量保障，包括带宽限制、拥塞控制、流量分类等功能。
- 网络编程和应用开发：RYU提供了丰富的API和SDK，支持Python编程语言，可以方便地开发和部署SDN应用程序，如基于SDN的网络监控、负载均衡、流量控制和优化等。

来源：DeepSeek

控制器	架构类型	核心特点
RYU	**集中式**	单实例架构，基于Python开发，代码简洁且模块化清晰。适合小型网络或实验环境，轻量级但缺乏分布式扩展能力 1 3 。
OpenDaylight	**集中式/可扩展分布式**	基于Java开发，采用OSGi框架实现模块化动态加载，核心架构包含模型驱动服务抽象层（MD-SAL），支持通过集群扩展实现分布式部署。适合企业级复杂网络 6 9 12 。
ONOS	**分布式**	专为大规模网络设计，采用Java开发，基于Raft协议实现分布式集群，支持横向扩展和高可用性。核心层（ONOS Core）管理全局网络状态，适配层支持多协议。适用于运营商级广域网 9 13 11 。

维度	RYU	OpenDaylight	ONOS
开发语言	Python	Java	Java
适用场景	实验环境、学术研究、小型网络	企业网络、混合设备管理	运营商广域网、大规模SDN
扩展性	低（单实例）	中（模块化扩展）	高（分布式集群）
安全性	基础TLS支持	支持RBAC、加密通信	分布式容错、安全策略管理
典型应用案例	校园网实验、OpenStack集成	谷歌B4网络（早期版本）	中国联通敏捷VPN、AT&T Domain 2.0

南向接口

控制器	主要协议支持	特点
RYU	**OpenFlow（全版本）**	专注于OpenFlow协议，兼容性强，但缺乏对其他南向协议的支持 1 9 。
OpenDaylight	**OpenFlow、NETCONF、OVSDB、BGP-LS等**	支持多协议适配，MD-SAL层将不同协议抽象为统一模型，适用于异构网络设备管理 6 12 13 。
ONOS	**OpenFlow、NETCONF、PCEP、SNMP等**	南向接口分三类：拓扑收集（如BGP-LS）、配置管理（如OVSDB）、流量控制（如OpenFlow），适配运营商复杂需求 9 11 。

北向接口

控制器	接口类型与特点	应用开发支持
RYU	**Python API**	通过Python直接调用控制器功能，开发灵活但依赖编程能力。提供REST API但功能有限 1 3 。
OpenDaylight	**REST API（RESTCONF）、Java API**	提供标准化REST接口和YANG模型驱动API，支持与OpenStack等云平台集成，适合企业级应用开发 6 12 。
ONOS	**REST API、Intent Framework（声明式API）**	Intent Framework允许开发者通过声明式指令（如"连接A到B"）定义网络策略，简化应用开发流程 9 13 。

开放课题

开放课题

- SDN与SR协同的5G承载网架构设计
- SDN驱动的5G网络切片动态资源分配
- SDN与NFV融合的5G移动核心网重构

- 基于SDN的5G边缘计算与核心网协同优化
- SDN在垂直行业的定制化网络服务设计

- 运营商网络中SDN部署的平滑演进路径
- SDN控制器性能优化与安全性增强

- SDN驱动的网络自动化运维与智能分析