1 Fat Tree 整个拓扑网络分为三个层次：自下而上分别为边缘层（Edge）、汇聚层（Aggregate）及核心层（Core），其中汇聚层交换机与边缘层交换机构成一个 Pod，交换设备均是采用商用交换设备。Fat-Tree 构建拓扑规则如下：Fat-Tree 拓扑中包含的 Pod 数目为 k，每一 Pod 连接的 server 数目为(k/2)^2，每一 Pod 内的边缘交换机及聚合交换机数量均为 k/2，核心交换机的数量为(k/2)^2，网络中每一交换机的端口数目为 k，网络所能支持的服务器总数为(k^3)/4。Fat-Tree 结构采用水平扩展（scale-up）的方式，当拓扑中包含的 Pod 数目增加，交换机的端口数目增加时，Fat-Tree 拓扑能够支持更多的服务器，满足数据中心的扩展需求，如 k=48 时，Fat-Tree 能够支持的服务器数目为 27648；Fat-Tree 结构通过在核心层多条链路实现负载的及时处理，避免网络热点；通过在 pod 内合理分流，避免过载问题。

Fat-Tree 拓扑结构网络性能参数如表 1 所示（对于 Pod 数目为 k 的拓扑， 服务器数量为 N）：

网络直径	对分带宽	并行链路	扩展能力
2logN(log 以 2 为底)	N/2	(k/2)^2	(K^3)/4

Fat-Tree 对分带宽随着网络规模的扩展而增大，因此能够为数据中心提供高吞吐传输服务；不同 Pod 之间的服务器间通信，源、目的节点对之间具有多条并行路径，因此网络的容错性能良好，一般不会出现单点故障；采用商用设备取代高性能交换设备，大幅度降低网络设备开销；网络直径小，能够保证视频、在线会与等服务对网络实时性的要求；拓扑结构规则、对称，利于网络布线及自动化配置、优化升级等。Fat-Tree 结构也存在一定的缺陷：Fat-Tree 结构的扩展规模在理论上受限于核心交换机的端口数目，不利于数据中心的长期发展要求；对于 Pod 内部，Fat-Tree 容错性能差，对底层交换设备故障非常敏感，当底层交换设备故障时，难以保证服务质量；拓扑结构的特点决定了网络不能很好的支持 one-to-all 及 all-to-all 网络通信模式，不利于部署 MapReduce、Dryad 等现代高性能应用；网络中交换机与服务器的比值较大，在一定程度上使得网络设备成本依然很高，不利于企业的经济发展。

2 DCell

DCell 结构[3]是由微软研究人员提出的拓扑结构，该拓扑采用递归方式以低端交换设备取代高性能设备实现数据中心互连。在 DCell 结构中，交换机与服务器都具有数据转发的功能，因此 DCell 拓扑属于混合型拓扑。

DCell 拓扑通过低端口 mini-switch 与多端口服务器以递归方式构建大规模网络。在 DCell 结构中，存在两种连线方式，即服务器与交换机相连，服务器与服务器相连，不存在交换机与交换机相连的情况。$DCell_0$ 结构是构建拓扑的基本单元，n 代表 $DCell_0$ 中交换机的端口数目，k 代表 DCell 结构的层数，图 4 所示即为 n=4，k=1 的 $DCell_1$ 拓扑互连结构。若在 $DCell_{k-1}$ 中包含 t_{k-1} 个服务器，则 $DCell_k$ 将由 $t_{k-1}+1$ 个 $DCell_{k-1}$ 构成，这就意味着很小的 n，k 即可容纳很多的服务器，且随着节点度的增加，服务器的数目呈 e^2 增长。如 n=4，k=3，则 $DCell_3$ 可以容纳 176820 个服务器，从而保证网络的高度可扩展性要求。DCell 结构中每一层次以全连通方式互连，因此可以提供高对分带宽传输及良好的容错性能。 n 端口 mini-switch 与多端口 server 互连的 $DCell_k$ 结构具体参数如表 2 所示：

表2　DCell 拓扑网络参数

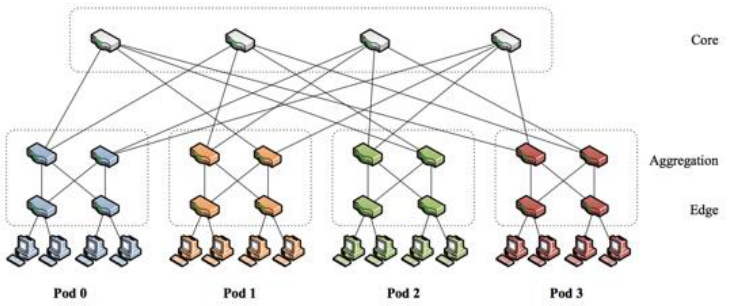
网络直径（上限）	对分带宽	节点度	服务器数量
$2^{k+1}-1$	$\frac{t_k}{4\log_n t_k}$	$k+1$	$\left(n+\frac{1}{2}\right)^{2^k}-\frac{1}{2}<t_k<(n+1)^{2^k}-1$

从上表可知，DCell 结构具有显著特点：对于很小的 n、k，DCell 结构能够满足数据中心网络的高可扩展性、高对分带宽要求；DCell 结构每一层是采用全连通方式，因此网络具有良好容错特性；以 mini-switch 取代高性能互连设备，实现低成本互连；另外，DCell 结构能够很好的支持 one-to-all 及 all-to-all 通信服务模式。

但 DCell 也因其结构特点，导致拓扑存在一定缺陷：每层结构之间以全连通方式互连，网络拓扑不规整，使得布线复杂度较高，不利于工程实施及自动化配置、管理等；当网络链路故障率超过一定门限时，网络将会被分离成不同孤立的子网，导致网络瘫痪；在 all-to-all 通信模式中，网络流量分布不均匀，低层流量比较集中，容易导致网络拥塞；以长链路取代高性能交换机，导致链路开销增加；服务器节点度的增加，导致 NIC 数量显著提高，网络成本开销进一步提升。

3　PortLand 是 2009 年提出的一种数据中心架构模式，它解决了传统架构的几种不足，具有以下几个优点：虚机迁移 IP 不变且原连接正常；部署前不需要人工配置交换机；任意主机可达；无环；链路失败检测。PortLand 采用的是二层架构模式，方便扩容、迁移，同时解决了二层 ARP，广播等带来的问题。

下图是胖树拓扑（Fat tree）的例子，也是 PortLand 采用的拓扑。拓扑为三层拓扑，从上而下分别是核心层、汇聚层和边缘层。边缘交换机下连主机，上连汇聚层，一个主机连接一个边缘交换机。几个汇聚层交换机（文中为 2 个）和几个边缘交换机（文中为 2 个）互连组成一个 Pod。每个汇聚层上连 2 条链路到核心层，注意一个 Pod 上连的核心交换机各不相同，这是为了充分负载和容错的考虑。



3.1　Fabric Manager

Fabric Manager（以下简称 FM）为集中化管理的软件，主要用于处理 ARP，链路容错以及多播报文等问题。为了保证健壮性，同样对 FM 做主备，防止单个 FM 失败带来的全网不通的问题。

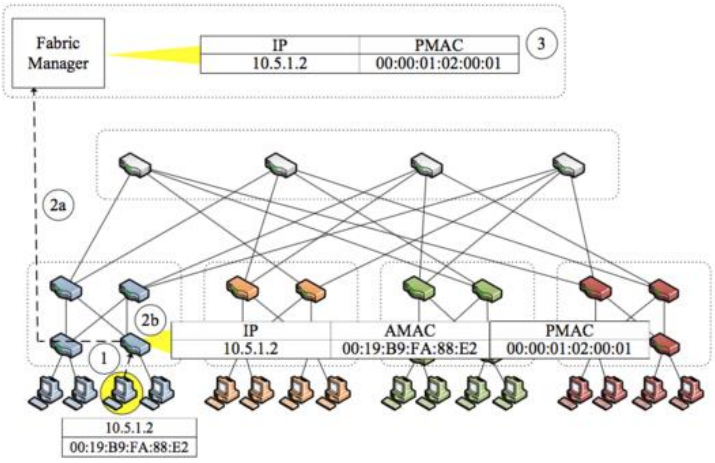
PortLand 采用逻辑集中的架构管理器，可以维护关于网络配置信息（如拓扑）的软状态。在 PortLand 中，我们限制集中知识的数量，并将其限制为软状态。 以这种方式，我们消除了对架构管理器的任何管理员配置的需要(例如，交换机的数量，它们的位置，其标识符)。在一个或多个备份上以主异步更新状态进行复制。

3.2 MAC 地址

二层架构导致了 MAC 地址成为本文问题的关键。PortLand 提出了 PMAC 的概念（Pseudo MAC），也就是一个层次化的假的 MAC 地址，每台主机分配一个 PMAC 地址，同时主机还有一个 AMAC（Actual MAC），即原 MAC 地址。主机在访问别的主机时会修改目的和源主机的 AMAC 地址为 PMAC 地址，这部分工作由源主机上联的边缘交换机完成，然后，在目的主机上联交换机处将目的 PMAC 改为 AMAC，使得报文可以送达目的主机。

PMAC 地址的格式如下：pod.position.port.vmid。其中，pod(16 bits)代表边缘交换机所在的 pod 号码；position(8 bits)表示 pod 内部的位置；port(8 bits)表示主机所上连的边缘交换机的端口号；vmid(16 bits)为主机标识号，递增序列，且存在超时机制，超时则回收并重新使用。

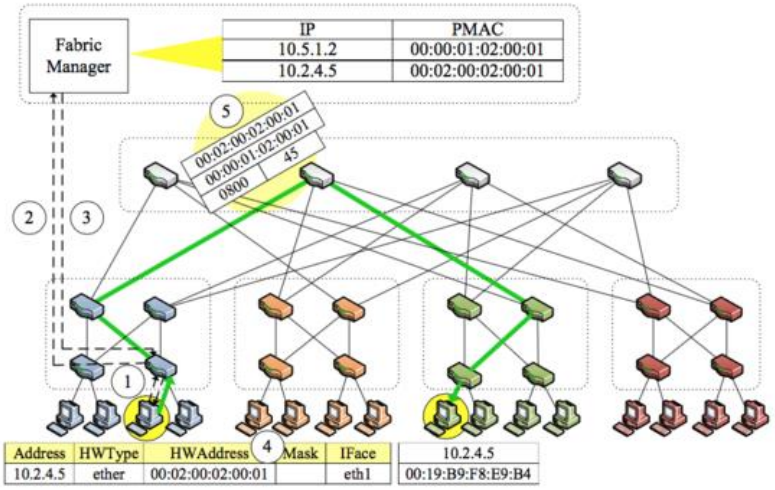
下图给出了一个 AMAC 到 PMAC 映射的例子。当一个边缘交换机发现某个 MAC 地址（AMAC）未被学习过，则报文被发送至 FM，FM 创建 AMAC+IP 到 PMAC 的映射，该条目下次用于回复 ARP。



设置 PMAC 的本质是为了分离位置信息和名字信息，使得主机同时具有位置（PMAC）和名字（AMAC）两大信息，方便了主机的迁移，即只需要变更 PMAC。我们可以用 OpenFlow 协议很好的实现本节的内容。

3.3 ARP

下图展示了 FM 处理 ARP 报文的过程，首先是边缘交换机收到 ARP 报文请求，然后送至 FM，FM 查表目的 PMAC，若有则返回消息，最后边缘交换机发回 ARP reply。当然，也可能存在 FM 不存在目的 PMAC 的情况，这时候，FM 将会发送广播请求给所有主机，然后获取映射关系：ARP 先发给核心交换机，然后转发至各个 Pod，最后到边缘交换机，再到主机，主机再返回 reply 给边缘，此时对于之前不存在的主机，将建立映射关系，最后 FM 将会得到这个映射告知源主机。



需要特殊处理的情况是虚机迁移，一旦一个虚机从一个物理机迁移到另外一个物理机，该虚机将会发送新的 IP 到 MAC 的映射给边缘层再给 FM。FM 发送一条无效消息给虚机之前上联的边缘交换机，告知原映射缓存失效。被动触发更新：若有新连接仍然来到原边缘交换机，则交换机回复新的虚机位置（即新的映射关系），并可以重新转发流量给新的虚机（可选）以保证报文不丢失。

3.4　位置探测协议

PortLand 推出了 LDP(Location Discovery Protocol)协议确定各个交换机的位置（核心、汇聚、边缘），PortLand 交换机定期从每个端口发送 LDM(Location Discovery Message)，用于保活和更新交换机位置。LDM 报文包括以下信息：

Switch identifier(switch_id)：交换机的唯一 id。

Pod number(pod)：Pod 号码。

Postition(pos)：每个 pod 内的边缘交换机都有唯一的 pos 号码。

Tree level(level)：0，1，2，分别标识核心、汇聚、边缘。

Up/down(dir)：标识交换机的端口是上联还是下联。

LDM 报文的传播可以确定层级关系。边缘交换机只会从和汇聚交换机相连的端口才会收到 LDM 报文，边缘交换机可以通过连接主机来确定自身关系（只有边缘交换机才会与主机相连接），汇聚交换机可以通过连接边缘交换机来确定自身关系（只有汇聚交换机才会连接边缘交换机），核心交换机可以通过其所有端口都连接汇聚交换机来确定自身关系（核心交换机只与汇聚交换机相连接）。具体执行算法见 ppt algorithm1。

所有交换机都将定期发送 ping（我的理解应该类似是 Hello 报文），所有连接的交换机和主机都将回复 ping 但只有交换机才会传送 LDM 而主机不会。

ppt（边缘交换机：如果一个交换机没有连接到超过 k / 2 个邻居交换机足够长的时间，则断定它是一个边缘交换机。在接收到的任何后续的 LDM 上，边缘交换机推断相应的进入端口是向上的。

聚合交换机：从向上端口的边缘交换机接收 LDM 的交换机的结论是，它必须是聚合交换机，相应的进入端口是向下的端口。

核心交换机：尚未建立其级别的交换机首先验证

- 所有活动端口都连接到其他 PortLand 交换机。
- 所有邻居都是聚合交换机，尚未设置其链路的方向

如果这些条件成立，交换机可以断定它是核心交换机，并将其所有端口设置为向下。）

刚开始的时候，每个边缘交换机都将根据以下算法申请一个 pod 内的唯一 pos 号

码（ppt algorithm2）。ppt（边缘交换机必须在 0 .. k / 2-1 的范围内获取每个 pod 中的唯一位置号码。每个边缘交换机在相同的 pod 中为所有聚合交换机提供在适当范围内的随机选择的数字。 如果提案被大多数这些交换机验证为未使用，而不是暂时保留，则建议将被定稿，并且此值将包含在来自边缘交换机的将来的 LDM 中。采用位置 0 的边缘交换机从 Fabric 管理器请求一个 pod 号码。）

3.5 无环链路

注意，在三层架构中，从边缘到汇聚，和从汇聚到核心都可以采用 ECMP 路径。层级递送报文保证了一旦报文往下传送（从高层到底层），报文不会回传（从底层到高层），也就是不会导致环路。

3.6 容错路由

3.6.1 Fault Tolerant Routing

1：在某些可配置的时间段内没有收到 LDM（在此上下文中也称 keepalive），交换机将承担链路故障。

2：检测交换机通知 Fabric Manager 失败。

3：Fabric Manager 为整个拓扑结构维护具有每条链路连接信息的逻辑故障矩阵，并在步骤 3 中使用新信息进行更新。

4：Fabric Manager 通知所有受影响的交换机故障，然后根据拓扑的新版本单独重新计算其转发表。

3.6.2 Fault Tolerant Routing for Multicast

1：pod 0 中两个突出显示的链接同时失败。

2：两台聚合交换机检测到故障。

3：交换机通知 Fabric Manager，从而更新其故障矩阵。

4：Fabric Manager 计算所有受影响的组播组的转发表项。

5：从故障中恢复需要通过 pod 0 中的两个单独的聚合交换机进行转发。但是，没有单个核心交换机同时连接到两个汇聚交换机。集合覆盖问题，可以贪心解决。

4 Ethane

4.1 Motivation

- 企业网络：

运行各种各样的应用程序和协议；在严格的可靠性和安全性限制下运行

- 需要手动配置：

昂贵且容易出错；多供应商网络中网络停机时间的 62％来自人为错误；80％的 IT 预算用于维护和运营

- 网络管理方法:

推出专有的中间箱，放置在网络阻塞点,如防火墙；向现有网络添加功能，如在交换机上添加 ACL

- 只隐藏复杂性，而不是减少它。
- 我们如何改变企业网络架构，使其更易于管理？
- 三个基本原则：
  - 网络应由在高级别名称上声明的策略来管理。
  - 策略应该确定数据包遵循的路径。
    - 策略可能需要数据包通过中间的中间框;
    - 如果路径受到控制，流量可以获得更合适的服务;
  - 网络应该强制绑定一个数据包和它的来源。

4.2 Overview of Ethane Design

4.2.1 Overview

- Ethane 通过不明确许可，不允许终端主机之间的任何通信来控制网络。
- 两个组件
  - 中央控制器
    - 包含全网络策略和拓扑
    - 对允许的流量执行路由计算。
  - 一套 Ethane switches
    - 简洁
    - 由简单的流程表和控制器的安全通道组成
    - 在控制器的指导下转发数据包

4.2.2 Names, Bindings, and Policy Language

- 当组件加入，离开和移动网络时保持命名空间一致
- 机器 - 地址 - 用户
  - Ethane 接管地址的所有绑定，表现为 DHCP 服务器
  - 机器在网络上注册
  - 用户需要与网络进行身份验证，比如 WiFi 热点
- 优点：
  - 控制器可以跟踪任何实体所在的位置
  - 控制器可以记录日志中的所有绑定和流入条目以进行网络事件重建

4.2.3 Ethane in Use

- 注册：所有交换机，用户和主机都在控制器上注册，以便进行验证
- 引导
  - 通过创建一个基于控制器的生成树来切换引导连接
  - 每个交换机与控制器进行认证并创建一个安全通道
- 认证
  - UserA 将网络与 HostA 加入，交换机 1 首先将所有 hostA 的数据包转发给控制器
  - HostA 向控制器发送 DHCP 请求，控制器将 HostA 到 IPA，IPA 到 MAC 和 MACA 到物理端口绑定到交换机 1
  - UserA 打开一个 Web 浏览器，其流量指向控制器，并通过 Web 表单进行身份验证
- 流设置
  - 在确定数据包与其流表中的任何活动条目不匹配后，交换机 1 将数据包转发给控制器
  - 控制器决定是允许还是拒绝流，或者要求它遍历一组 waypoints。
  - 控制器计算流，向路径中所有交换机的流表添加一个新条目
- 什么是流？源计算机到目的地的分组的序列，目的地可以是另一主机，多播组或广播域
- 转发
  - 如果路径允许，则控制器将数据包发送回交换机 1，交换机 1 根据新的流条目进行转发
  - 来自流的后续数据包由交换机直接转发，不会发送到控制器
  - 流入口保留在交换机中，直到超时

4.3 Ethane in More Detail

4.3.1 Ethane Switch

- Ethane 交换机比传统以太网交换机简单得多
  - 不需要知道地址，支持 VLANs，检查源地址欺骗或保持流量统计。
  - 如果 layer3，则不需要运行 OSPF，ISIS，RIP 等路由协议。
- Ethane 交换机的流表可以比等同的以太网交换机中的转发表小得多。
  - 以太网交换机需要记住它可能遇到的所有地址。
  - Ethane 交换机只需要跟踪进行中的流量

4.3.2 Flow Table and Flow Entries

- 流表中的两种常见类型的条目：
  - 应转发的流的每个流条目
  - 每个主机条目，用于数据包应该被丢弃的行为不当的主机
- 条目被删除是因为由于不活动而超时或者由控制器撤销

4.3.3 Local Switch Manager

- 建立和维护到控制器的安全通道
- 两种方式：
  - 对于与 Controller 相同物理网络一部分的交换机，使用最小生成树协议
  - 对于与 Controller 不在同一广播域内的交换机，创建一个 IP 通道
- 交换机通过广播和接收邻居发现消息来维护相邻交换机的列表
- 邻居列表每 15 秒定期发送给控制器

4.3.4 Controller

- 控制器保存策略文件，将其编译为快速查找表
- 路由计算使用网络拓扑来选择流的路由
- 拓扑由交换机管理器维护，交换机管理器从交换机接收链路更新
- 注册：所有由网络命名的实体（即主机，协议，交换机，用户和接入点）必须注册。它们组成策略命名空间，用于静态检查策略。
- 验证：不指定特定的主机认证机制，如 802.1X
- 跟踪绑定：当交换机，主机和用户加入，离开和移动网络时，跟踪网络事件的名称，地址和物理端口之间的所有绑定
- 命名空间接口
  - 在目前的网络中，几乎不可能很快地弄清楚用户的活动
  - Ethane 控制器可以记录所有的认证和绑定信息，可以准确地确定哪个用户发送数据包，发送时间，所用的路径及其目的地。
- 许可检查和访问授予：控制器收到数据包后，会检查该策略以查看哪些操作适用于该策略
- 执行资源限制：控制器可以限制流量的速率，限制新流设置的速率，或限制分配的 IP 地址数量

4.3.5 Broadcast and Multicast

- 处理组播：交换机保留每个流的位图，以指示要沿路径发送数据包的哪些端口。 控制器可以计算广播或多播树，并在路径设置期间分配相应的位
- 处理广播发现协议：
  - 主机试图找到服务器或地址; 例如 ARP，DHCP
  - 鉴于控制器知道所有，它可以回复请求而不创建新的流并广播流量

4.3.6 Replicating the Controller: Fault-Tolerance and Scalability（三种复制技术）

- 冷备用：备份控制器坐视不理，等待接管

- 如果失败，网络将收敛到新的根，for MST（最小生成树）
- 备份只需要包括注册状态和网络策略
- 主要优点是简单; 缺点是主机，交换机和用户需要重新验证并重新绑定主服务器的故障

➢ 热备份：为每个控制器创建单独的 MST
- 控制器监视彼此的活动，并且在检测到主要故障时，辅助控制器将根据静态排序进行接管
- 需要跨控制器复制绑定
- 一些新用户和主机需要重新认证

➢ 全复制：两个或多个活动控制器
- 交换机只需要对一个控制器进行身份验证，然后可以将其流请求扩展到控制器（例如散列或循环）
- Gossip 为事件提供弱一致的排序。

4.3.7 Others

➢ 链路故障
- 交换机删除与故障端口绑定的所有流表项，并将其新的链路状态信息发送到控制器
- 控制器学习新的拓扑

➢ 引导
- 在启动时，网络创建一个最小的生成树，控制台将自己本身作为根
- 如果交换机找到一个到控制器更短的路径，它会尝试双向身份验证，然后将该路径作为有效的路由

4.4 The POL-ETH Policy Language

➢ Ethane 网络策略被声明为一组规则，每个规则由条件和相应的操作组成

[(usrc="bob")∧(protocol="http")∧(hdst="websrv")]:allow;

- 条件：如果发起流的用户为“bob”，流协议为“HTTP”，流目的地为主机“websrv”
- 操作：操作包括允许，拒绝，航点和仅出站

➢ 规则是独立的，不包含固有的顺序

4.5 Ethane’s shortcoming

➢ 广播和服务发现协议
- 创建大量的流条目，导致大量流量
- 除非 Ethane 可以解释协议并代表其做出回应。

➢ 应用层路由
- Ethane 的策略可能会受到较高层的沟通的影响
- 例如，如果 A 允许与 B 通话而不能和 C 通话，如果 B 可以与 C 通话，则 B 可以将消息从 A 转发到 C.

➢ 知道用户在做什么
- Ethane 的策略假设传输端口号表示用户在做什么
- 恶意用户或应用程序可能会通过同意使用非标准端口号码来愚弄 Ethane

➢ 以太网地址欺骗
- Ethane 交换机依靠用户和以太网地址之间的绑定来识别流
- 如果用户欺骗 MAC 地址，可能会愚弄 Ethane 向终端主机发送数据包

- 拒绝服务攻击
  - 在交换机上：刷新流程表
  - 在控制器上

5 OpenFlow

OpenFlow 是一种在软件定义的网络环境下构建控制和数据平面之间的通信的协议。

5.1 Background

5.1.1 New Computing Trends

- 改变交通模式：
  - 在经典的“南北”交通模式返回数据到最终用户设备之前，一直处于“东西向”的机对机交通状态
  - 私有/公共云，导致广域网上的额外流量
- “消费化 IT”：IT 需要适应各种个人设备，同时保护企业数据和知识产权并满足合规性要求
- 云服务的兴起：计算，存储和网络资源的弹性缩放，理想地是从共同的观点和一套通用的工具。
- “大数据”意味着更多带宽：大数据集的兴起正在加剧数据中心对额外网络容量的不断需求。

5.1.2 The Conventional Network

- Hierarchical with tires of Ethernet switches
- Tree structure

5.1.3 Limitations of Current Networking Technologies

- 导致停滞的复杂性
  - 协议往往被隔离定义，每个解决一个具体的问题，没有任何基本的抽象的好处。这导致了当今网络的主要限制之一：复杂性。
  - 网络的静态本质与当今服务器环境的动态性质形成鲜明对比。应用程序分布在虚拟机之间。
  - 许多操作 IP 语音，数据和视频流量的 IP 融合网络，虽然现有网络可以为不同的应用程序提供差异化的 QoS，但这些资源的配置是高度手动的
- 不一致的策略：为了实施全网络策略，IT 可能需要配置数千个设备和机制
  - 费时
  - 由于复杂性难以应用一套一致的策略
- 无法缩放：
  - 网络变得非常复杂，增加了数百或数千个必须配置和管理的网络设备。
  - 大型运营商，如 Google，Yahoo！和 Facebook，需要所谓的超大规模网络，可以在数十万甚至数百万的物理服务器之间提供高性能，低成本的连接。这种缩放不能用手动配置完成。
- 供应商依赖：
  - 运营商和企业寻求部署新的功能和服务，以快速响应不断变化的业务需求或用户需求。
  - 供应商设备产品周期：3 年以上。
  - 缺乏标准的开放接口限制了网络运营商将网络定制到各自环境的能力。

5.2 Software Defined Networking

- 网络控制与转发分离，可直接编程。
- 网络运营商和管理员通过编程方式配置这种简化的网络抽象，他们可以自己

编写这些程序，而不是等待功能嵌入在供应商的专有和封闭的软件环境中

- SDN 架构支持一组可以实现通用网络服务的 API，路由，组播，安全，访问控制，带宽管理，流量工程，服务质量，...，定制满足业务目标

5.3 The OpenFlow Protocol

5.3.1 OpenFlow Switches

- OpenFlow 提供了一个开放式协议，用于在不同的交换机和路由器中对流表进行编程。
- OpenFlow 交换机由至少三部分组成
  - 流表，具有与每个流条目相关联的动作，以告知交换机如何处理流
  - 安全通道，将交换机连接到远程控制进程（称为控制器），允许在控制器和交换机之间发送命令和数据包
  - OpenFlow 协议，为控制器与交换机通信提供了开放和标准的方式
- 什么是流？流可以是 TCP 连接，或来自特定 MAC 或 IP 地址的所有数据包，或具有相同 VLAN 标签的所有数据包，或来自同一交换机端口的所有数据包。
- 每个流条目都有一个与之相关联的简单操作。至少有三个基本动作
  - 将此流的数据包转发到给定的端口（或端口）
  - 将此流包封装并转发给控制器
  - 删除此流的数据包
- 流表中的一个条目有三个字段：
  - 数据包头，定义流
  - 动作，定义了如何处理数据包，
  - 统计数据，保留每个流的数据包和字节数，以及自上一个数据包匹配以来的时间
- 一个 10 元组包头

In	VLAN	Ethernet			IP			TCP	
Port	ID	SA	DA	Type	SA	DA	Proto	Src	Dst

5.3.2 Benefit of Openflow-based SDN

- 集中控制多供应商环境，无需思科认证
- 通过自动化降低复杂性
- 更高的创新率
- 提高网络可靠性和安全性，可以确保有线和无线网络基础设施一致地执行访问控制，流量工程，服务质量，安全性和其他策略
- 更细粒度的网络控制，每个地址块->每个流
- 更好的用户体验，例如，自动视频分辨率适应

5.4 Using OpenFlow

- 示例 1：网络管理和访问控制
  - Ethane：基本思想是允许网络管理员在中央控制器中定义一个全网络的策略，通过对每个新流程进行准入控制决定直接执行。
  - 控制器通过管理名称和地址之间的所有绑定来将数据包与其发件人相关联 - 它实质上接管 DNS，DHCP 并在所有用户加入时进行身份验证，跟踪他们连接到哪个交换机端口（或接入点）。
- 示例 2：VLAN
  - 最简单的方法是静态地声明一组流，这些流指定给定 VLAN ID 上的流量

可访问的端口。
    - 更动态的方法可能会使用控制器来管理用户的身份验证，并使用用户位置的知识在运行时标记流量。
- 移动无线 VoIP 客户端
    - 支持 WiFi 功能手机的呼叫切换机制。
    - 实施一个控制器来跟踪客户端的位置，重新路由连接 - 重新编程流表 - 用户通过网络移动，允许从一个接入点到另一个接入点的无缝切换。
- 非 IP 网络，启用 OpenFlow 的交换机可以支持非 IP 流量的几种方式。 例如，可以使用它们的以太网头标识流
- 示例 5：处理数据包而不是流。
    - 方法 1：强制所有流的数据包默认通过控制器。 更灵活，以性能为代价
    - 方法 2：将它们路由到可执行数据包处理的可编程交换机

6 NOX

6.1 Motivation

- 网络通过各个组件的低级配置进行管理
    - 用 ACL 阻止用户需要知道用户的当前 IP 地址
    - 强制端口 80 流量到 HTTP 代理需要广泛了解当前的网络拓扑
- 企业网络类似于没有操作系统的计算机，与网络相关的组件配置与硬件相关的机器语言编程类似
- 需要网络操作系统提供观察和控制网络的能力
    - 不管理网络本身，而是提供一个编程接口
    - 在网络操作系统之上实现的应用程序执行实际的管理任务
- 不同
    - 呈现具有集中编程模式的程序; 程序的写作就好像整个网络都存在于一台机器上一样
    - 程序是根据高级抽象（例如，用户和主机名）而不是低级配置参数（例如 IP 和 MAC 地址）来编写的

6.2 NOX Overview

6.2.1 Components

- 一组交换机：支持 OpenFlow 抽象
- 一个或多个网络连接的服务器：运行 NOX 软件; 可以被认为涉及到几个不同的控制器进程和单个网络视图

6.2.2 Granularity

- NOX 的网络视图：
    - 交换机级拓扑; 用户，主机，中间框和其他网元的位置; 并提供服务（例如，HTTP 或 NFS）。
    - 名称和地址之间的所有绑定。
    - 不包括当前的网络流量状态。
- 选择粒度涉及可扩展性与灵活性之间的折衷
    - 集中的每个数据包控制：不可行实现跨任何相当大的网络。**无可扩展性**
    - 基于前缀的路由表：不允许充分的控制，因为两个主机之间的所有数据包都必须遵循相同的路径。 **没有灵活性**
- 基于中间流的粒度，一旦对某个数据包进行了控制，相同标题的后续数据包将以相同的方式进行处理

6.2.3 Switch Abstraction

- 独立于特定的交换机硬件。
- 支持流级别控制粒度。
- 交换机由流表来表示，其条目如下：<header : counters; actions>
- 对于与指定标题匹配的每个数据包，计数器将被更新并执行相应的操作。如果数据包匹配多个流条目，则选择具有最高优先级的条目。
- OpenFlow 的基本操作
  - 作为默认转发（即转发，如果 NOX 不存在）
  - 转发指定的接口
  - 拒绝
  - 转到控制器进程
  - 修改各种数据包标题字段（例如，VLAN 标签，源和目标 IP 地址和端口）。

6.2.4 Operation

- 在交换机上，如果传入的数据包与交换机上的流条目匹配，更新相应的计数器并应用相应的操作。
- 如果不匹配，转发到控制器进程。两例：
  - 这些数据包通常是流的第一个数据包
  - 控制器进程可以选择从某些协议接收所有数据包。（例如，DNS）
- NOX 应用程序使用流启动和其他转发流量
  - **观察**：构建网络视图
    使用 DNS，DHCP，LLDP 和流程启动来构建网络视图，包括网络拓扑和名称地址绑定集合
  - **控制**：确定是否转发流量，如果是，则确定哪条路由
    开发访问控制和路由应用程序，以确定是否应允许流程，计算适当的 L2 路由，沿着路径在所有交换机中安装流条目，然后将数据包返回到始发交换机

6.3 Programming Interface

- 活动
  - 使用一组在特定事件发生时被注册执行的处理程序。
  - 某些事件由 OpenFlow 消息直接生成。
  - 由于处理这些低级事件和/或其他应用程序生成的事件，NOX 应用程序会生成其他事件。
- 网络视图和命名空间
  - NOX 包括许多构建网络视图的“基础”应用程序，并维护可被其他应用程序使用的高级命名空间。
  - 允许以拓扑独立的方式写入应用程序。
  - 高级声明可以针对网络视图进行“编译”，以产生每个数据包执行的低级查找功能。
  - 应用程序不良的最糟糕的结果是性能下降，功能不正确。
- 控制
  - 管理应用程序通过 OpenFlow 进行网络控制。
  - OpenFlow 不会包括自定义的每个数据包处理（例如深度数据包检测）。
  - NOX 应用程序可以通过专门的中间箱来引导流量
- 高级服务，一套“系统库”来提供许多网络应用程序通用功能的高效实现。

7 FlowVisor

7.1 FLowVisor 为一个网络抽象层，包含了一种新的交换机层网络虚拟化技术，同一个硬件转发平面可以被多个逻辑网络共享。类似于计算机的虚拟层，FLowVisor 位于底层硬件和上层软件之间（位于 OpenFlow 控制器和 OpenFlow 交换机之间的虚拟层）。操作系统利用指令集控制底层硬件，FLowVisor 利用的是 OpenFlow 协议，有如下如特性：

- FLowVisor 定义切片为由 OpenFlow 交换机组成的拓扑上的一组流
- FLowVisor 位于每个 OpenFlow 控制器和 OpenFlow 交换机之间，确保每个控制器只能看到和控制他应该管理的 OpenFlow 交换机
- FLowVisor 通过给组成切片的流一个最小的数据速率来划分带宽
- FLowVisor 通过追踪流表项属于哪个控制器来划分交换机的流表

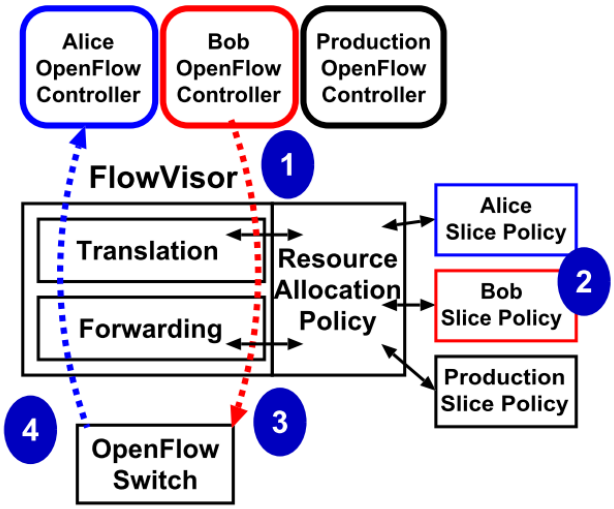
7.2 FLowVisor 定义切片为一组流，因此我们可以将切片当作一组域，给定一个数据包包头，我们可以判断出这个数据包包含在哪个流空间中，即可判断出这个数据包属于哪个虚拟网络，所以可以将 FLowVisor 的一个实例等价于一个流空间。

FLowVisor 的设计目标有三点：

- 这个虚拟化对控制器来说应该是透明的
- 不同的切片之间是完全独立的
- 切片定义是可扩展的

7.3 工作流程

FLowVisor 类似于一个 OpenFlow 代理，对 OpenFlow 交换机和控制器之间的消息进行拦截。所有的 OpenFlow 消息，不论从交换机到用户还是用户到交换机，都会经过 FLowVisor，FLowVisor 利用 OpenFlow 协议和用户、交换机进行通信，FLowVisor 对交换机和用户来说是透明的，所以在用户看来，他们是直接和交换机通信的。接下来以下图简单的例子来阐述 FLowVisor 的操作流程。



Bob 在控制器上面运行了一个 HTTP 负载均衡应用，将所有的 HTTP 流散播到一组服务器上，Bob 的 FLowVisor 的策略是切片网络从而让自己只处理所有源 IP 为一个固定值的 HTTP 流，而控制器上可以看到所有的 HTTP 流，控制器自信可以实现负载均衡的任务，他会下发流表，对所有的 HTTP 流负载均衡。当 Bob 的控制器下发一条流表（如将某个流指引到某个服务器），FLowVisor 会拦截他（上图 1），和 Bob's 的切片策略进行对比（上图 2），重写流表项只对固定源 IP 的 HTTP

流有效，这样控制器就实现了只控制固定源 IP 的流，但是他觉得自己控制了所有的流。同样的，对于从交换机向控制器的消息，FLowVisor 只允许满足对应切片流空间的消息上传。FLowVisor 不需要 FLowVisor 实例和物理交换机一对一存在，一个 FLowVisor 实例可以控制多个物理交换机，甚至可以虚拟化另外一个虚拟网路。

7.4 切片定义策略

FLowVisor 中切片被定义为可插入模块，每个策略由文本配置文件来描述的，一个切片一个。对于带宽分配，一个切片的所有流都会被映射到一个 Qos 组，每个切片有固定数量的交换机 CPU 和转发流表的预算，网络拓扑被指定为网络结点和端口的列表。

用一个有序的元素列表，类似于防火墙规则来定义的每个切片的流空间，每个规则描述有一个相关的操作，比如，允许，只读或者拒绝，这些被按照特定顺序来进行解析，执行第一个匹配规则的操作。将所有规则组合起来作为流空间的一部分，基本控制了整个切片。只读规则只允许切片接收 OpenFlow 控制消息，查询交换机的统计信息，不允许在转发表中插入流表。规则是允许重叠的。

我们接着复杂化前面的场景：Bob 邀请了他的一些小伙伴和他合作来做 HTTP 负载均衡实验，网络管理员 Alice 准许 Bob 进行这些合作性实验，Alice 将参与这个合作性 HTTP 实验。实验的人员的 HTTP 流交由 Bob 控制，她继续负责剩余的 HTTP 流，另外，Alice 还想要运行一个被动的切片来监测整个网络的性能，为了实现这样的功能，我们可以用下面的流空间规则。

- Bob'的试验网络：被定义为所有参与实验的人员所需要的 HTTP 流，他的网络描述文件对每个用户只有一个规则，格式如下：
  Allow: tcp_port: 80 and ip = user_ip
  凡是来自于交换机中可以匹配这条规则的 OpenFlow 消息被转发到 Bob 的控制器，所有 Bob 意图插入的流表项都会满足这些规则
- Alice 的生产网络：是 Bob 网络的补集，所以规则如下：
  Deny: tcp_port:80 and ip=user_ip
  Allow: all
  凡是不进入 Bob 虚拟网络的交换机的 OpenFlow 消息都会被进入生产网络的控制器，对于不满足 Bob 流空间的流生产控制器就可以随意对它们进行控制
- Alice 的监测网络：需要看到所有网络中的流，他的规则如下：
  Read-only: all
  这个规则确保 Alice 的检测网络是完全被动的，不与她的生产网、Bob 的试验网产生任何冲突

7.5 隔离机制

7.5.1 带宽隔离

FLowVisor 可以通过标记数据包的 VLAN 优先级比特位来利用现存的交换机带宽隔离特性。VLAN 标签拥有三个比特域，PCP（VLAN Priority Code Point）有一个将数据包映射到八个优先级队列中的其中一个的标准机制，OpenFlow 协议里面有对 VLAN 标签和优先级比特的管理，所以可以给流中的所有数据包一个优先级。

因为，为了增强带宽隔离，FLowVisor 重写了所有切片的转发表，加了一个“set VLAN priority”的操作，将优先级置为八个优先级队列中的某一个，一个虚拟网络实例中的所有流根据资源分配策略映射到相应的流组（流组需要

网络管理员通过 CLI 定义）

使用 VLAN PCP 比特位并不是 FLowVisor 所固有的，而是为了和商用硬件协调工作所选择的短期变通方案。

7.5.2 拓扑隔离

控制器通过 OpenFlow 协议发现网络中的结点和链路，在非虚拟网络中，当一个网络设备连接到控制器的 TCP 监听端口的时候，控制器就可以发现这个设备。FLowVisor 作为交换机和控制器中间的一个代理，他只代理用户虚拟网络中的交换机和控制器的连接，同理，用于罗列交换机所有物理端口的 OpenFlow 消息，也会被 FLowVisor 拦截，修改为只有虚拟网络中有的端口。

7.5.3 交换机 CPU 隔离

商用交换机上的 CPU 都是典型的低功耗嵌入式处理器，因此很容易过载，在大多数交换机硬件中，一个高负载的 CPU 会导致严重的网络破坏。比如，一个 CPU 高负载的情况下，硬件转发数据会继续进行，但是交换机会停止响应控制器的 OpenFlow 请求，导致 LLDP 链路发现协议超时，让交换机误以为网络连接非常不稳定，网络变得不可用。

交换机的 CPU 负载主要被下面四种程序瓜分，这四个负载来源需要不同的隔离机制：

- 产生新流到达的消息：在 OpenFlow 交换机中，如果一个数据包无法匹配流表，就会向控制器发送 PacketIn 消息，这个过程消耗了一部分处理资源，如果新流比较多，消息产生很频繁，CPU 资源就会耗尽。为了防止饿死，FLowVisor 会追踪新流产生的消息到达速率，如果速率超过了阈值，FLowVisor 插入一条转发流表规则在一个时间段内丢弃所有的数据包，就这样，FLowVisor 利用 OpenFlow 协议限制了新流的速度。
- 处理控制器的请求：对于每个虚拟网络实例，FLowVisor 通过限制 OpenFlow 消息速率来限制 CPU 消耗，因为不同类型的消息消耗不同的 CPU 资源，因此这个工作现在还略显不足。
- “慢路径”转发：除了很快的专用的硬件转发路径，数据包通过“慢路径”转发，这当然会消耗一部分 CPU 资源，而这个消耗显然是不确定的。FLowVisor 通过重写多个转发规则，将他们分开，分别为一次数据包转发事件来防止控制器一次插入多个“慢路径”转发规则，比如 OpenFlow 的 PacketOut 消息，他可以一次下发很多转发规则，但是 FLowVisor 可以拦截，将其分为多个。这样，“慢路径”数据包会通过上面提到的两条机制：新流产生的消息和控制器的请求来限速。
- 维持内部状态：所有的网络设备会使用 CPU 去更新他们的内部计算器、处理事件等等，所以必须留有足够的 CPU 给交换机“记账”。FLowVisor 通过限制上面三点的 CPU 消耗来保障交换机有足够的资源执行内部功能。

和带宽隔离一样，CPU 隔离机制不是 FLowVisor 固有的设计，更像是一个解决现在硬件抽象问题的应急措施。

7.5.4 流空间隔离

每个虚拟网络只能处理他们流空间中的数据流，FLowVisor 对 OpenFlow 消息进行重写，确保每个切片只能控制他自己的流，不能影响别的切片的流。当然，不是所有的规则都能够被重写来满足某个切片的流空间，FLowVisor 只能让规则更加详细精确，比如如果 Bob 的控制器想创建一个规则去影响所有

的流，FLowVisor 应该重写规则让其只影响 TCP 数据流。但是 FLowVisor 不会重写影响 22 端口的数据流的规则，让其只影响 80 端口（TCP 数据流），在这种情况下，FLowVisor 会给控制器发送一个错误消息。

7.5.5 流表项隔离

FLowVisor 会统计每个虚拟网络使用的流表项数目，确保其不会超过一个预先给定的值。对于控制器插入到交换机的每个规则，FLowVisor 会给其计数器加一，如果某个流表到期失效，计数器会减一。由于硬件约束，一些交换机会内部扩展规则来满足多个输入端口，FLowVisor 需要处理这个情况。OpenFlow 协议为 FLowVisor 协议提供了一个机制来精确的列出交换机中的所有流表项，当控制器超出了流表项预先给定的值，任何新的规则插入都只会收到“table full”的错误提醒消息。

7.5.6 OpenFlow 控制通道隔离

除了上面提到的所有物理资源外，OpenFlow 控制器通道本身也必须虚拟化和隔离。比如，OpenFlow 协议中的所有消息包含一个独一无二的传输 ID，FLowVisor 必须重写这些传输 ID 以确保不同控制器的消息不会使用同一个 ID，类似的，如果一个数据包的处理需要交换机做决定，需要先将数据包存储在交换机队列中，OpenFlow 协议使用一个 32 位的整数去标识这个队列中的不同数据包。FLowVisor 需要确保每个交换机只能访问交换机队列中他自己的数据包。

8 P4

8.1 Motivation

8.1.1 Motivation

- 在过去的几年中，OpenFlow 协议标准已经演进得越来越复杂
- 新首部区域的增多过程没有表露出任何即将结束的迹象
- 相比重复地扩展 OpenFlow 的协议标准，我们认为未来的交换机应该为包解析和首部域匹配支持灵活的机制，允许控制器应用通过一个通用的开放的接口利用交换机的这些能力（例如新的“OpenFlow 2.0”接口）。比起如今的 OpenFlow 1.x 标准，上述这样一个通用的可扩展的方法将是更简单、更优雅也更不会过时的。

8.1.2 Three goals

- 重配置能力：控制器应该能够重新定义数据包的包解析过程和对首部区域的处理过程；
- 协议无关性：交换机不应该与特定的包格式绑定。相反地，控制器应该能够指定：
  - 1)一个能提取出特定名称和类型的首部区域的包解析器
  - 2)一个类型化的用于处理这些首部区域的“匹配 – 动作”表的集合
- 目标无关性：正如 C 开发者不需要知道底层具体是什么 CPU 在工作一样，控制器的开发者不必知道底层交换机的细节。而是 P4 编译器在将目标无关的 P4 描述转换成目标相关的用来配置交换机的程序时，才应该去考虑交换机的能力。

8.2 Abstract Forwarding Model

- 源于对 OpenFlow 的探索，我们的模型包含了三个一般化。首先，OpenFlow 假设有一个固定的包解析器，在这一点上我们的模型能够支持可编程的解析器，允许定义新的协议首部。第二，OpenFlow 假设“匹配 – 执行动作”的各个阶段是串行的，在这一点上我们的模型允许它们是并行的或串行的。第三，我

们的模型假设“动作”是使用交换机所支持的协议无关的原语编写而成的。

- 这个转发模型受两类操作控制：配置和下发。配置操作编写了包解析器，设置了“匹配 – 执行动作”各阶段的顺序，指定了每个阶段要处理的协议首部区域。配置操作决定了支持哪种网络协议，也决定了交换机可能会如何处理数据包。下发操作将表项添加到“匹配 – 动作”表中，或从其中移除。其中，表本身是在配置操作的时候指定好的。下发操作决定了在任意给定时刻应用到数据包上的执行策略。
- 到达的数据包首先被包解析器处理。我们假设数据包的数据内容是与包首部分开缓存的，并且不能够用来进行匹配。解析器从包首部中找出并提取某些区域，这也即定义了交换机所支持的协议。这个模型没有对协议首部的含义做任何假设，仅仅是解析后的数据包的表现形式定义了一个首部区域的集合，“匹配 – 执行动作”的过程就在这个集合上进行。

  提取出来的首部区域接下来会被传递到“匹配 – 动作”表。这个表被分为入口表和出口表两部分。虽然两个表都可能会修改包首部，但是入口的“匹配 – 动作”决定了是数据包将去往哪一个出口，也决定了数据包将被放入哪一个队列。基于入口的处理结果，数据包可能会被转发、复制（为了组播、SPAN 端口监控或发往控制平面）、丢弃或触发流量控制。出口的“匹配 – 动作”在包首部上为每一个动作目标单独做一轮修改，比如在组播复制数据包的时候所做的。动作表（计数器、流量监管器 policer 等）可以与一条流关联起来，追踪其每一帧的状态。
- 数据包可以在其被处理的不同阶段之间携带额外的信息，称作 metadata 元数据。元数据可以同等地被当成一个数据包首部区域。元数据的一些例子有入端口号、传输目的地和队列、用于数据包调度的时间戳，以及在表与表之间传递的数据，这些数据不涉及改变数据包解析后的表现，比如虚网标识号。

  排队策略的处理方式同当前 OpenFlow 的一样：通过一个动作将一个数据包映射到一个队列中，这个队列是为了接收特定服务策略而配置的。服务策略（例如最低速率、DRR 轮询）的选择是交换机配置的一部分。

8.3 A Programming Language

- 一门数据包处理语言必须允许开发者暗示或明示首部区域之间任何串行化的依赖。这种依赖决定了哪些表可以并行执行。例如，由于 IP 路由表和 ARP 表之间的数据依赖，它们是需要串行执行的。依赖可以通过分析 TDGs 表依赖图而识别；这些图描述了输入的首部区域、动作和表之间的控制流程。
- 两步编译过程。在顶层，开发者使用必要的能描绘 P4 控制流的语言来编写数据包的处理程序；在这一层之下，编译器将对 P4 的描述翻译成 TDG 表依赖图，以促进依赖的分析，然后编译器将会把 TDG 映射到具体的交换机对象上。

8.4 An Example

- P4 概念
  - 首部 Headers：首部的定义描述了一系列首部区域的顺序和结构。它包含区域长度的规范，约束了区域数据的取值。
  - 解析器 Parsers：解析器的定义描述了如何识别数据包内的首部和有效的首部顺序。
  - 表 Tables：“匹配 – 动作”表是执行数据包处理的机制。P4 程序定义的首部区域可能会用于匹配，或者在其上执行特定的动作。

- 动作 Actions：P4 支持通过更简单的协议无关的原语构造复杂的执行动作。这些复杂的动作可以在“匹配 – 动作”表中使用。
- 控制程序 Control Programs：控制程序决定了“匹配 – 动作”表处理数据包的顺序。一个简单又必要的程序描述了“匹配 – 动作”表之间的控制流。

➢ 首部格式
- 每一个首部的详细阐述都是通过声明一个各首部区域名称和长度的有序列表来完成
- mTag 首部可以直接添加，而不必替换已有的这些声明。首部区域名称表明核心层有两层的汇聚。每一个核心交换机都被编写了一些规则来检查这些字节中的某一个。具体检查哪一个字节，是由交换机在层次中所处的位置和数据流的方向（上或下）决定的。

➢ 数据包解析器
- P4 假设底层交换机可以实现一个状态机，这个状态机能够自头至尾横贯数据包的各个首部，随着状态机的行进提取首部区域的值。提取出来的首部区域值被送入“匹配 – 动作”表进行处理。
- P4 把状态机直接描述成从一个首部到下一个首部的过渡转移的集合。每一个过渡转移可能会被当前首部中的值触发。
- 数据包的解析从 start 状态开始，一直行进直到到达了明确的 stop 状态或是遭遇到无法处理的情况（这可能被标记成错误）。在到达了对应下一个首部的状态时，状态机根据首部的规范描述提取出首部，然后根据状态机的下一个过渡转移继续向前行进。提取出来的首部被送往交换机流水线后半部分的“匹配 – 执行动作”的处理过程。mTag 的解析器是非常简单的：它只有四种状态。

➢ 表的规范
- 边缘交换机匹配二层目的地和 VLAN ID，然后选择一个 mTag 添加到首部中。开发者定义一张表来匹配这两个首部区域，以及执行一个添加 mTag 首部的动作（见后文）。其中的 reads 属性声明了要匹配哪些首部区域，同时限定了匹配类型（精确匹配、三重匹配等）。actions 属性列出了“匹配 – 动作”表可能会对数据包执行的动作。动作将会在本文后续部分讲解。max_size 属性指明了“匹配 – 动作”表需要能够支持多少条表项。
- 表的规范允许 P4 编译器决定存储表需要多大的存储空间，以及在什么样的存储器（比如 TCAM 或 SRAM）上实现这个表。

➢ 动作的规范
- P4 定义了一个基本动作的集合，可以利用它们构造复杂的动作。
- 如果某个动作需要有输入参数（例如 mTag 中的 up1 值），参数将会在运行时由匹配表提供。
- 在这个例子中，交换机将 mTag 标签插入在 VLAN 标签之后，复制 VLAN 标签的 Ethertype 字段到 mTag 中，以指明后续载荷是什么协议的封包，然后设置 VLAN 标签的 Ethertype 字段为 0xaaaa，表明其后跟随的是 mTag 标签。在边缘交换机上执行的相反动作没有展示出来，这些动作将会从数据包中剥去 mTag 标签。
- P4 的基本动作包括：

  1.Set_field：将首部中的某一特定区域设置为特定的值，支持带掩码的设

置；

2.Copy_field：将一个首部区域的值拷贝到另一首部区域中；

3.Add_header：添加一个有效的特定的首部（以及它所有的首部区域）；

4.Remove_header：从数据包中删除（pop 取出）一个首部（以及它所有的首部区域）；

5.Increment：递增或递减一个首部区域的值；

6.Checksum：计算首部区域的一些集合的校验和（比如 IPv4 校验和）。

- 控制程序
  - 指定从一个表转移到下一个表的控制流
  - 在包解析之后，source_check 表确认接收到的数据包和入端口是否一致。例如，mTag 只应该存在于连接到核心交换机的端口上。source_check 也会从数据包中剥去 mTag 标签，同时在元数据中记录数据包是否拥有 mTag 标签。流水线中后续的表可能会匹配这个元数据以避免再次往数据包中添加标签。
  - local_switching 表稍后将会被运行。如果没有匹配上，就意味着这个数据包的目的地不是连接在同一个交换机上的主机。在这种情况下，mTag_table 表（上述定义的）将会用来匹配这个数据包。本地和送往核心层的转发控制都可以被 egress_check 表处理。这个表将会在转发目的地未知的情况下，上送一个通知到 SDN 控制层。
- 编译包解析器，对于有可编程包解析器的设备，编译器将解析器描述翻译成解析状态机。对于固定的解析器，编译器仅仅确认解析器描述与目标设备的解析器是一致的。
- 编译控制程序
  - 必要的控制流描述是一种方便的指定交换机的逻辑转发行为的方法，但它不能明确地表示出表之间的依赖和并发执行的机会。因此我们部署一个编译器来分析控制程序，帮助我们识别依赖以及寻找能够并发处理首部区域的机会。最终，编译器为交换设备生成目标配置。
  - mTag 这个例子如何被映射到不同种类的交换设备中

1.软件交换机：

软件交换机提供了完整的灵活性：表的数量、表的配置和解析都是在软件的控制之下。编译器直接将 mTag 表图映射到交换机的表中。编译器使用表类型信息来限制表的宽度、长度和每张表的匹配准则（例如精确匹配，范围匹配或通配符匹配）。编译器也可能通过软件数据结构来优化三重匹配或者前缀匹配。

2.内含 RAM 和 TCAM 的硬件交换机：

编译器可以配置哈希散列，使用 RAM 来对边缘交换机的 mTag_table 表执行有效的精确匹配。而核心层的 mTag 转发表是要匹配标签比特位的一个子集，它将被映射到 TCAM 中去执行匹配。

3.支持并行的规则表的交换机：

编译器能够检测数据依赖，然后安排各个表是串行执行还是并行执行。在 mTag 例子中，mTag_table 表和 local_switching 表可以并行执行直到设置 mTag 动作的执行。

4.在流水线的末端才执行动作的交换机：

对于只在流水线的末端才执行动作的交换机。编译器可以告诉中间的步骤生成元数据，在最终的执行中使用。在 mTag 例子中，无论是 mTag 标签的添加还是移除，都能在元数据中表现出来。

5.只能容纳少量表的交换机：

编译器可以将大量的 P4 表映射到较少量的物理表中。在 mTag 例子中，本地交换表可以与 mTag 表结合起来。当控制器在运行时安装了新的规则，编译器的规则翻译器可以将原本应写入两个 P4 表中的规则重新编写，生成在单个物理表中的规则。

9 Protocol Oblivious Forwarding

9.1 Challenges of OpenFlow

- 挑战
  - 遵循一个反应性而不是主动的演进路径。
  - 转发平面几乎是无状态的。缺乏主动监控流量状态和改变流量行为的能力，而不需要控制器的参与。
- 后果
  - 控制平面和转发平面没有足够的分离，FE 需要知道协议
  - 没有简单的方法来修改数据包格式，或者添加辅助数据到网络中传送的数据包，更不用说应用用户定义的协议。
  - 添加新的转发协议功能需要对 FE 和控制器进行大修，在最坏的情况下，对 FE 芯片组和硬件进行全面的重新设计。新协议不断涌现，例如 VXLAN，NVGRE
  - 有限的表现力会严重影响转发平面的可编程性。

9.2 Make SDN FE a White Box

- 当前的网络设备是黑盒子。
- OpenFlow 和其他一些 SDN 的风格，使得网络设备灰盒子具有有限的可编程性，FE 保持相当多的智能，控制器可以做任何事情。
- SDN 真正需要的是一个完全可编程的转发平面，其中 FE 都是白盒。
- FE 在 SDN 的框架中应该具有类似 CPU 的作用，以确保其灵活性和可扩展性。 也就是说，FE 不应该要求任何先验知识和理解应用程序语义。
- 一组简单的协议无关指令足以构成来自控制平面的任何网络服务。我们将这样的指令命名为流指令集（FIS）。
- 用户可以自由使用网络协议和数据包格式，从而进一步脱离现状。

9.3 POF Overview

- 在 POF 中，FE 不了解任何转发协议。
- 分组解析由控制器通过一系列通用密钥组合和表查找指令来指导。
- 搜索键简单地由一个或多个{offset，length}元组定义
  - 偏移指示来自分组（或元数据）中的当前光标的跳过比特
  - 长度表示从偏移位置开始应包含在键中的位数。
- 所有转发指令都是故意使协议无关的。
- 相反，在 FIS 中，对于操纵数据包或元数据（例如插入，删除和修改）的指令，使用{offset，length}来定位目标数据。
- 核心概念
  - 表搜索键定义为{offset，length}元组
  - 指令/操作使用{offset，length}元组访问数据包数据或元数据
  - 包括其他数学，逻辑，移动，分支和跳转指令

- 分组字段解析和处理被抽象为通用指令，以实现灵活和未来的证明转发元素

10 Networking Named Content

10.1 CCN Node Model

- 有两个 CCN 包类型：兴趣包和数据包。消费者在所有可用的连接上通过广播兴趣包请求内容。任何听到兴趣包的结点，如果有可以满足兴趣包的数据，都能够用数据包来响应。数据包仅在对兴趣包的响应中传输，数据包消费请求包。兴趣包和数据包都通过名字识别正在交换的内容，多个对同样内容感兴趣的结点使用标准多播抑制技术通过广播介质共享传输。如果兴趣包中的内容名是数据包中内容名的前缀，则数据包满足兴趣包。
- 核心 CCN 包转发引擎有三个主要的数据结构：FIB（转发信息库）、CS（内容缓存，缓冲存储器）和 PIT（待定兴趣包表）。
  - FIB 用来向匹配数据的可能源转发兴趣包。除了他列出输出接口的一个列表，而不是一个单独的输出接口之外，几乎和 IP 中的 FIB 表相同。这反映了 CCN 没有限制在一个生成树上转发的事实。它允许多个数据源，能够并行地查询他们。
  - 内容缓存与 IP 路由器的缓冲存储器相同，但有一个不同的替换策略。由于每个 IP 包属于一个单独的点到点会话，在被向下游转发后没有更多价值。因此 IP“忘记”了一个包，并当转发完成后立即回收缓存（MRU 替换）。CCN 包是幂等的，自我识别的和自我认证的，因此每个包对许多消费者都可能是有用的。为了最大化共享的可能性，最小化上游带宽需求和下游延迟，CCN 尽可能长时间地记住到达的数据包（LRU 或者 LFU）
  - PIT 记录转发到朝向内容源的上游的兴趣包，以至于返回数据包能够向下游请求者发送。在 CCN 中，仅仅兴趣包被路由，因为他们向上游可能的数据源传播，他们留下一个“面包屑”的痕迹为一个可以匹配的数据包遵循到源请求者的返回路径。每个 PIT 条目是一个面包屑。一旦 PIT 表项被用来转发一个匹配的数据包后，PIT 表项就被擦除（数据包消费兴趣包）。从来没有找到匹配数据包的兴趣包的 PIT 条目最终会超时。
- 处理兴趣包

  当兴趣包到达某一接口，就执行内容名字的最长匹配查询。用来查询的索引结构是排好序的，以至于内容缓存匹配将比 PIT 匹配优先，PIT 匹配优先于 FIB 匹配。

  因此如果内容缓存中已经有可以匹配兴趣包的数据包，他将从兴趣包到达的接口发送出去，兴趣包将会被丢弃（由于它已经被满足了）。

  另外，如果有确切匹配的 PIT 条目，兴趣包的到达接口将会被添加到 PIT 表项的请求接口列表中，兴趣包将会被丢弃。（兴趣包中的数据已经被向上游发送，因此所需要做的事是确保当请求的数据包到达时，包的一个副本将会被从新的兴趣包到来的接口发送出去。）

  另外，如果有一个匹配的 FIB 条目，那么兴趣包需要被朝着数据源向上游发送。如果查询 FIB 的结果列表不空，到达接口从 FIB 条目的接口列表中移除，然后，兴趣包被从所有保留的接口发出，一个新的 PIT 条目被从兴趣包和它的到达接口创建。

  如果兴趣包没有匹配，则它将被丢弃（这个结点没有任何匹配数据，而且不知道怎样找到匹配数据）。
- 处理数据包

由于数据包没有被路由，而仅仅遵循 PIT 表项链返回到请求源，所以数据包的处理相对简单。数据包一旦到达，就执行数据包的内容名字的最长匹配查询。内容存储如果匹配，意味着数据包是一个副本，因此被丢弃。FIB 匹配意味着没有匹配的 PIT 条目，因此数据包是未经请求的，所以被丢弃。

10.2 Transport

- CCN 传输被设计为在不可信包交付服务的上层操作，因此兴趣包，数据包，或者两者都可能在传输的过程中丢失或损坏，或者被请求的数据可能临时不可用。
- CCN 发送者是无状态的，最后的消费者（发出初始兴趣包的应用程序）如果仍然想要数据，那么他对重新发出未满足的兴趣包负有责任。接收者的策略层对在特别接口上的重传负有责任（由于他知道这个接口上的上游结点的超时时间），接收者的策略层也选择哪个可用的通信接口和多少可用的通信接口用来发送兴趣包，多少未满足的兴趣包应该被允许发送，不同兴趣包的优先级等。
- 一个兴趣包最多获取一个数据包。这个基本原则确保流平衡在网络中保持，并允许不同速率的网络上的多样的机器之间的有效的通信。然而，就像在 TCP 中，数据包和请求包可能重叠。在数据包到达消费第一个兴趣包之前，多个兴趣包可能被立即发出。
- 在 CCN 中，所有的通信都是本地的，因此没有意义讨论没有涉及到流平衡的发送者和接收者之间的会话。由于 CCN 流平衡在每一跳保持，不需要额外的技术在路径中间控制拥塞。
- CCN 名字是被层次化地组织的，以至于一个单独的名字由许多成分组成。每个成分由许多任意的八位组组成。对协议栈中的高层，名字必须是有意义的和有用的，但是除了组件架构，传输没有限制条件。为了计数的方便，我们呈现了类似 URI 的名字，用/字符分割组件，但是这些分隔符不是名字的一部分，没有包含在包的编码当中。
- 兴趣可以准确地指定内容；内容名称可以包含自动生成的结尾，如序列号；名称的最后一部分将为下一个块（例如视频帧）递增；这些名称形成一个预先遍历的树；这样，接收者可以在他的兴趣数据包中询问下一个数据包

11 Data-Oriented (and Beyond) Network Architecture

11.1 Basic Design

11.1.1 Naming

- 围绕 principals 组织的命名
  - 每个主体都与一个公私密钥对相关联，并且每个数据或服务或任何其他命名实体与主体相关联
- 名称的形式是 P：L
  - P 是主体公钥的加密散列
  - L 是 principals 选择的标签，确保这些名称是唯一的
- 命名的粒度由 principals 决定，principal 可以选择仅将她的网站命名，或者将她的网站和其中的每个页面命名，或以更细的粒度命名

11.1.2 Self-certifying names

- principals 被认为拥有自己的数据。 一个数据来自主体的公钥和主体的数据签名。客户端收到三元组<data，public key，signature>
- 如果客户端接收到一个名称为 P：L 的数据，则可以验证数据是否来自主体通过：
  - 检查公钥哈希

- 验证签名对应于公钥

- 挑战是将平面名称解析为适当的位置。

11.1.3 Name Resolution

- 名称解析是通过使用两个基本原语完成的：
  FIND（P：L）和 REGISTER（P：L）
  FIND（P：L）找到名为 P：L 的对象
  REGISTER 消息设置了 RHs 有效路由 FIND 的必要状态

12 Serval

12.1 Modern Internet Service

- 多重性
  - 多个副本
  - 多个接口
  - 多路径
- 动态性
  - 复制失败和恢复
  - 服务迁移
  - 客户流动性
- 与今天的 TCP / IP 堆栈匹配不好

12.2 Service Access Today

- DNS 将服务绑定到客户端的位置（早期绑定）
- 数据中心 LB 将单个 IP 映射到多个服务器
- 迁移虚拟机以平衡云中的负载
- 交换网络或接口时流量中断

12.3 Rethink the Network Stack

- 应用层
  - 今天的应用程序运行在两个低级标识符（IP 地址和 TCP / UDP 端口）；在启动通信之前，客户端必须使用带外查找机制（例如 DNS）或先验知识（例如，Web 在端口 80 上）来早日绑定到这些标识符；IP 地址和端口上的套接字编程
  - Serval：应用程序通过使用 serviceID 的活动套接字进行通信，可以将 serviceID 延迟绑定到服务实例。
- 传输层
  - 今天的堆栈使用五元组<remote IP，远程端口，本地 IP，本地端口，protocoli>将传入的数据包解复用到套接字；接口地址无法更改，而不会中断正在进行的连接 - 缺少移动性
  - Serval：传输协议仅处理跨越一个或多个流的数据传输，包括重传和拥塞控制; Demux 是基于 flowID 的 SAL 层
- 网络层
  - 今天的网络层使用分层 IP 寻址来高效地传送数据包；受到地址更改时上层协议失败的堆栈中终端主机移动性的需求的挑战。
  - Serval：网络层不变。

12.4 Naming

- Group-Based Service Naming
  称为服务 ID 的服务名称对应于提供相同服务的一个或多个（可能改变的）进

程的组。

- ServiceID
  - 服务名称不规定粒度
    可以是单个 SSH 守护程序，一组打印机，一组对等体分发一个公用文件
  - 格式
    提供者前缀
    基于前缀的聚合和 LPM
  - 保护沟通和注册：
    以自我认证的 bitstring 结束
- Explicit Host-Local Flow Naming
  - Serval 通过在连接设置期间分配和交换的 flowID 提供明确的主机 - 本地流命名。
  - 优点：
    - 网络层疏忽：识别流，而不知道网络层地址。
    - 移动性和多路径：FlowID 可帮助识别各种动态事件的流量，如迁移，地址更改等。

12.5 The Serval Network Stack

- 用户空间服务控制器
  - 根据政策管理服务解决方案
  - 监听与服务相关的事件，监控服务性能，并与其他控制器进行通信
- 服务访问层（SAL）提供服务级数据平面
  - 负责通过服务表转发来连接到服务。
  - 一旦连接，SAL 将新流程映射到流表中的套接字，确保数据包解复用。

12.6 Active Socket

- 保持标准的 BSD 套接字接口。
- 一个服务 ID 自动调用绑定注册，登记关，进程终止，或超时。
  - 在本地服务注册事件中，堆栈将更新本地服务表并通知服务控制器，该服务控制器可以通知上游服务控制器。
  - 本地注销事件触发删除本地规则并通知服务控制器。
- 将 serviceID 解析为网络地址被委派给 SAL 应用程序，只需使用 serviceIDs 调用套接字接口，从不看到网络地址。

12.7 A Service-Level Data Plane

- 新连接（或未连接数据报）的第一个数据包包含一个 serviceID
- 该堆栈在 serviceID 上执行最长的前缀匹配（LPM），以从服务表中选择规则。

12.8 Actions

- FORWARD：
  - 包括一组或多个 IP 地址
  - 选择所有地址或使用加权采样来选择其中一个地址。
  - 对于每个选定的目的地，堆栈将数据包传递到网络层进行传送。
- DEMUX：
  - 当应用程序正在监听服务 ID 时，将接收到的数据包传递到本地套接字。
  - 由活动套接字的绑定事件添加
- DELAY：
  - 对数据包进行排队并通知服务控制器的 serviceID。

- 允许“按需”安装
- DROP：
  - 丢弃不需要的数据包

补：

1 Case Experiment: Replicated Web Services

- Four clients running the Apache benchmark generate requests to a Serval web service with an evolving set of Mongoose service instances.
  - each client requests a 3MB file and maintains an open window of 20 HTTP requests
- At time 60, two new service instances start, register with the service router
- At time 120, we force the first two servers to gracefully shut down, which causes them to close their listening socket
- Start another server at time 180

2

- Client issues SET requests with random keys 100,000 per sec.
- Time 0, all four servers are working
- Time 10, remove one server, SR re-partitions among the three servers.
- Time 20, remove another server, SR evenly partitions among the two servers.
- Time 30 and 40, failed servers join again.

3 Case Experiment: Interface Load Balancing and Virtual Machine Migration

- Two iperf clients then connected to the server and began transfers to measure maximum throughput.
- Six seconds into the experiment, the server’s service controller signals the SAL to migrate one flow to its second interface.

1 OSI 模型把网络通信的工作分为 7 层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。

2 MAC（Media Access Control）：介质访问控制，该协议位于 OSI 七层协议中数据链路层，数据链路层分为上层 LLC（逻辑链路控制），和下层的 MAC（介质访问控制），MAC 主要负责控制与连接物理层的物理介质。在发送数据的时候，MAC 协议可以事先判断是否可以发送数据，如果可以发送将给数据加上一些控制信息，最终将数据以及控制信息以规定的格式发送到物理层；在接收数据的时候，MAC 协议首先判断输入的信息并是否发生传输错误，如果没有错误，则去掉控制信息发送至 LLC（逻辑链路控制）层。

3 ARP（Address Resolution Protocol）：地址解析协议，是根据 IP 地址获取物理地址的一个 TCP/IP 协议。主机发送信息时将包含目标 IP 地址的 ARP 请求广播到网络上的所有主机，并接收返回消息，以此确定目标的物理地址；收到返回消息后将该 IP 地址和物理地址存入本机 ARP 缓存中并保留一定时间，下次请求时直接查询 ARP 缓存以节约资源。IP 地址在 OSI 模型的第三层，MAC 地址在第二层，彼此不直接打交道。在通过以太网发送 IP 数据包时，需要先封装第三层（32 位 IP 地址）、第二层（48 位 MAC 地址）的报头，但由于发送时只知道目标 IP 地址，不知道其 MAC 地址，又不能跨第二、三层，所以需要使用地址解析协议。

4 IGP（Interior Gateway Protocol,内部网关协议）是在一个自治网络内网关（主机和路由器）间交换路由信息的协议。路由信息能用于网间协议（IP）或者其它网络协议来说明路由传送是如何进行的。

5 BGP（Border Gateway Protocol，边界网关协议）：用于在不同的自治系统（AS）之间交换路由信息。当两个 AS 需要交换路由信息时，每个 AS 都必须指定一个运行 BGP 的节点，来代表 AS 与其他的 AS 交换路由信息。

6 IP（ Internet Protocol）：网络之间互连的协议也就是为计算机网络相互连接进行通信而设计的协议。

7 DNS（Domain Name System，域名系统），因特网上作为域名和 IP 地址相互映射的一个分布式数据库，能够使用户更方便的访问互联网，而不用去记住能够被机器直接读取的 IP 数串。通过主机名，最终得到该主机名对应的 IP 地址的过程叫做域名解析。

8 TCP（Transmission Control Protocol 传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议，完成第四层传输层所指定的功能。

9 UDP（User Datagram Protocol，用户数据报协议）:是 OSI 模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，在网络中它与 TCP 协议一样用于处理数据包。UDP 有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。

10 ICMP（Internet Control Message Protocol，Internet 控制报文协议）：TCP/IP 协议族的一个子协议，属于网络层协议，是一种面向无连接的协议，用于在 IP 主机、路由器之间传递控制消息。控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。

11 HTTP 协议（HyperText Transfer Protocol，超文本传输协议）：用于从 WWW 服务器传输超文本到本地浏览器的传输协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。HTTP 是客户端浏览器或其他程序与 Web 服务器之间的应用层通信协议。在 Internet 上的 Web 服务器上存放的都是超文本信息，客户机需要通过 HTTP 协议传输所要访问的超文本信息。HTTP 包含命令和传输信息，不仅可用于 Web 访问，也可以用于其他因特网/内联网应用系统之间的通信，从而实现各类应用资

源超媒体访问的集成。

12 OSPF(Open Shortest Path First 开放式最短路径优先）是一个内部网关协议，用于在单一自治系统内决策路由。是对链路状态路由协议的一种实现，隶属内部网关协议（IGP），运作于自治系统内部。著名的迪克斯加算法(Dijkstra)被用来计算最短路径树。

13 组播（Multicast）在发送者和每一接收者之间实现点对多点网络连接。如果一台发送者同时给多个的接收者传输相同的数据，也只需复制一份的相同数据包。它提高了数据传送效率。减少了骨干网络出现拥塞的可能性。

14 SDN（Software Defined Network 软件定义网络）是 Emulex 网络一种新型网络创新架构，是网络虚拟化的一种实现方式，其核心技术 OpenFlow 通过将网络设备控制面与数据面分离开来，从而实现了网络流量的灵活控制，使网络作为管道变得更加智能。

15 DHCP（Dynamic Host Configuration Protocol 动态主机配置协议）是一个局域网的网络协议，使用 UDP 协议工作，主要有两个用途：给内部网络或网络服务供应商自动分配 IP 地址，给用户或者内部网络管理员作为对所有计算机作中央管理的手段。

16 VLAN（Virtual Local Area Network 虚拟局域网）是一组逻辑上的设备和用户，这些设备和用户并不受物理位置的限制，可以根据功能、部门及应用等因素将它们组织起来，相互之间的通信就好像它们在同一个网段中一样。VLAN 是一种比较新的技术，工作在 OSI 参考模型的第 2 层和第 3 层，一个 VLAN 就是一个广播域，VLAN 之间的通信是通过第 3 层的路由器来完成的。与传统的局域网技术相比较，VLAN 技术更加灵活，它具有以下优点： 网络设备的移动、添加和修改的管理开销减少；可以控制广播活动；可提高网络的安全性。

17 ONF (Open Networking Foundation 开放网络基金会) 2011 年由 Deutsche Telekom, Facebook, Google, Microsoft, Verizon, 和 Yahoo!创立，其共同使命是加速开放 SDN 的部署。ONF 推广开放 SDN 和 OpenFlow 技术及标准，促进产品，服务，应用，客户和用户市场的发展。

18 QoS（Quality of Service 服务质量）指一个网络能够利用各种基础技术，为指定的网络通信提供更好的服务能力, 是网络的一种安全机制，是用来解决网络延迟和阻塞等问题的一种技术。在正常情况下，如果网络只用于特定的无时间限制的应用系统，并不需要 QoS，比如 Web 应用或 E-mail 设置等。但是对关键应用和多媒体应用就十分必要。当网络过载或拥塞时，QoS 能确保重要业务量不受延迟或丢弃，同时保证网络的高效运行。

19 VoIP（Voice over Internet Protocol）将模拟信号（Voice）数字化，以数据封包的形式在 IP 网络上做实时传递。VoIP 最大的优势是能广泛地采用 Internet 和全球 IP 互连的环境，提供比传统业务更多、更好的服务。VoIP 可以在 IP 网络上便宜的传送语音、传真、视频、和数据等业务。

20 ACL（Access Control List 访问控制列表）是路由器和交换机接口的指令列表，用来控制端口进出的数据包。ACL 适用于所有的被路由协议，如 IP、IPX、AppleTalk 等。

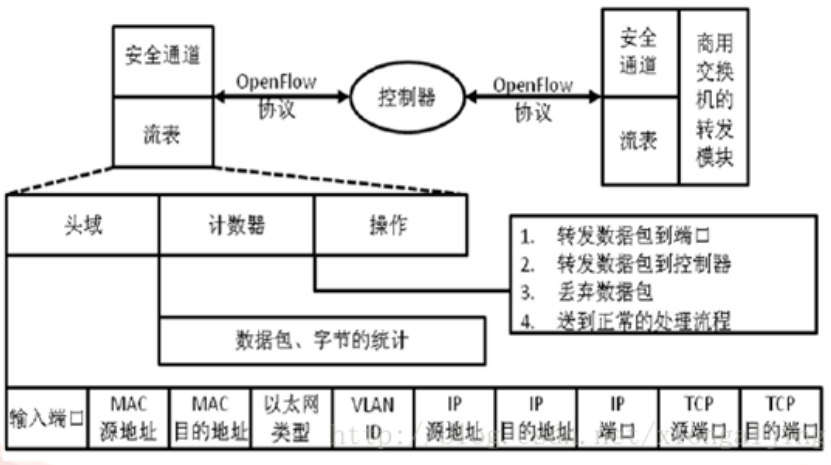
21 OpenFlow

最初出发点是用于校园内网络研究人员实验其创新网络架构、协议，考虑到实际的网络创新思想需要在实际网络上才能更好地验证，而研究人员又无法修改在网的网络设备，故而提出了 OpenFlow 的控制转发分离架构，将控制逻辑从网络设备盒子中引出来，研究者可以对其进行任意的编程从而实现新型的网络协议、拓扑架构而无需改动网络设备本身。

Openflow 的思路很简单，网络设备维护一个 FlowTable 并且只按照 FlowTable 进行转发，Flowtable 本身的生成、维护、下发完全由外置的 Controller 来实现，注意这里的 FlowTable 并非是指 IP 五元组，事实上 OpenFlow 1.0 定义的了包括端口号、VLAN、L2/L3/L4

信息的 10 个关键字，但是每个字段都是可以通配的，网络的运营商可以决定使用何种粒度的流，比如运营商只需要根据目的 IP 进行路由，那么流表中就可以只有目的 IP 字段是有效的，其它全为通配。

这种控制和转发分离的架构对于 L2 交换设备而言，意味着 MAC 地址的学习由 Controller 来实现，V-LAN 和基本的 L3 路由配置也由 Controller 下发给交换机。对于 L3 设备，各类 IGP/EGP 路由运行在 Controller 之上，Controller 根据需要下发给相应的路由器。流表的下发可以是主动的，也可以是被动的，主动模式下，Controller 将自己收集的流表信息主动下发给网络设备，随后网络设备可以直接根据流表进行转发;被动模式是指网络设备收到一个报文没有匹配的 FlowTable 记录时，将该报文转发给 Controller，由后者进行决策该如何转发，并下发相应的流表。被动模式的好处是网络设备无需维护全部的流表，只有当实际的流量产生时才向 Controller 获取流表记录并存储，当老化定时器超时后可以删除相应的流表，故可以大大节省 TCAM 空间。当一个 Controller 同时控制多个交换机/路由器设备时，它们看起来就像一个大的逻辑交换机，各个交换机/路由器硬件就如同这个逻辑网络设备的远程线卡。



OpenFlow 交换机是整个 OpenFlow 网络的核心部件，主要管理数据层的转发。OpenFlow 交换机接收到数据包后，首先在本地的流表上查找转发目标端口，如果没有匹配，则把数据包转发给 Controller，由控制层决定转发端口。

流表由很多个流表项组成，每个流表项就是一个转发规则。进入交换机的数据包通过查询流表来获得转发的目的端口。流表项由头域、计数器和操作组成；其中头域是个十元组，是流表项的标识；计数器用来计数流表项的统计数据；操作标明了与该流表项匹配的数据包应该执行的操作。

安全通道是连接 OpenFlow 交换机到控制器的接口。控制器通过这个接口控制和管理交换机，同时控制器接收来自交换机的事件并向交换机发送数据包。交换机和控制器通过安全通道进行通信，而且所有的信息必须按照 OpenFlow 协议规定的格式来执行。

OpenFlow 协议用来描述控制器和交换机之间交互所用信息的标准，以及控制器和交换机的接口标准。协议的核心部分是用于 OpenFlow 协议信息结构的集合。

OpenFlow 协议支持三种信息类型：Controller-to-Switch，Asynchronous 和 Symmetric，每一个类型都有多个子类型。Controller-to-Switch 信息由控制器发起并且直接用于检测交换机的状态。Asynchronous 信息由交换机发起并通常用于更新控制器的网络事件和改变交换机的状态。Symmetric 信息可以在没有请求的情况下由控制器或交换机发起。

22 NOX

22.1 从操作系统到网络操作系统

早期的计算机程序开发者直接用机器语言编程。因为没有各种抽象的接口来管理底层的物理资源（内存、磁盘、通信），使得程序的开发、移植、调试等费时费力。而现代的操作系统提供更高的抽象层来管理底层的各种资源，极大的改善了软件程序开发的效率。

同样的情况出现在现代的网络管理中，管理者的各种操作需要跟底层的物理资源直接打交道。例如通过 ACL 规则来管理用户，需要获取用户的实际 IP 地址。更复杂的管理操作甚至需要管理者事先获取网络拓扑结构、用户实际位置等。随着网络规模的增加和需求的提高，管理任务实际上变成巨大的挑战。

而 NOX 则试图从建立网络操作系统的层面来改变这一困境。这些操作系统实际上提供的是用户跟某些部件（例如交换机、路由器）的交互，因此称为交换机/路由器操作系统可能更贴切。而从整个网络的角度来看，网络操作系统应该是抽象网络中的各种资源，为网络管理提供易用的接口。

22.2 实现技术探讨

模型

NOX 的模型主要包括两个部分。

一是集中的编程模型。开发者不需要关心网络的实际架构，在开发者看来整个网络就好像一台单独的机器一样，有统一的资源管理和接口。

二是抽象的开发模型。应用程序开发需要面向的是 NOX 提供的高层接口，而不是底层。例如，应用面向的是用户、机器名，但不面向 IP 地址、MAC 地址等。

通用性标准

正如计算机操作系统本身并不实现复杂的各种软件功能，NOX 本身并不完成对网络管理任务，而是通过在其上运行的各种“应用”来实现具体的管理任务。管理者和开发者可以专注到这些应用的开发上，而无需花费时间在对底层细节的分析上。为了实现这一目的，NOX 需要提供尽可能通用的接口，来满足各种不同的管理需求。

架构

组件

下图给出了使用 NOX 管理网络环境的主要组件。包括交换机和控制（服务）器（其上运行 NOX 和相应的多个管理应用，以及 1 个 Network View），其中 Network View 提供了对网络物理资源的不同观测和抽象解析。注意到 NOX 通过对交换机操作来管理流量，因此，交换机需要支持相应的管理功能。此处采用支持 OpenFlow 的交换机。

操作

流量经过交换机时，如果发现没有对应的匹配表项，则转发到运行 NOX 的控制器，NOX 上的应用通过流量信息来建立 Network View 和决策流量的行为。同样的，NOX 也可以控制哪些流量需要转发给控制器。

多粒度处理

NOX 对网络中不同粒度的事件提供不同的处理。包括网包、网流、Network View 等。