新型网络技术

重要缩略语(背)

缩略	英文全称	中文解释
IoT	Internet of Things	物联网
SDN	Software Defined Networking	软件定义网络
P4	Programming Protocol-independent Packet Processors	协议无关的包处理器编程
PISA	Protocol Independent Switch Architecture	协议无关的交换机架构
TTL	Time to Live	生存周期
RTT	Round Trip Time	往返时间
QoS	Quality of Service	服务质量
QoE	Quality of Experience	体验质量
PHB	Per Hop Behavior	每跳行为
SLA	Service Level Agreement	服务等级约定
NFV	Network Function Virtualization	网络功能虚拟化
VMM	Virtual Machine Monitor	虚拟网络监视器

现代网络的组成

从参与者角度

名称	简介
端用户	
网络提供商	
内容提供商	
应用提供商	
应用服务提供商	

主要组件：

- 数据中心网络
- 物联网（IoT）

从网络元素角度

名称	简介
IP核心网	核心路由器（1Tbps，光链路）/边缘路由器（400Gbps，光链路）
局域网	
广域网	企业网/校园网；居民小区网；中小型商务网
公共蜂窝网	

从运营商运维角度

名称	简介
接入网	直接与端用户连接的网络
分发网	将接入网连接到核心网的网络
核心/主干网	提供网络服务的核心网络，互连分发网和接入网

现代网络的需求

流量类型(背书)

弹性流量

- 能够在较宽范围内调整以改变跨越互联网网络的时延和吞吐量，并且能够满足应用程序的需求；
- 应用于基于TCP/IP的互联网支持的传统类型流量；

非弹性流量

- 对网络延时和吞吐量的变化**缺乏适应能力**，有如下需求：吞吐量、时延、时延抖动、丢包；
- 应用于高质量音频，视频等多媒体传输，交互式仿真应用

大数据

- 从**网络涌入**处理器和存储设备的**大量、多样、高速率**的结构化和非结构化数据，这些数据经过转换后**存储**到企业的商用设备中
- **结构化数据**：数据库、数据仓库、电子表格
- **非结构化数据**：文档、通话记录、电子邮件、照片、图片、视频

QoS和QoE

QoS 服务质量

- 通过网络服务可测的 **端到端性能特性**，通过用户与服务提供商之间的服务等级预定 **SLA（Service Level Agreement）**得到 **事先保障**
- 主要性能特性：吞吐量、时延、时延抖动、误码率、丢包率、优先级、可用性、安全性

QoS的的典型应用

- 应用能够描述自身需求
- 网络能够标识出**不同应用的需求**（IP首部字段）
- 网络能够有**差别的优先处理**（DiffServ）
- 网络能够为特定的需求**预留足够的资源**（IntServ-RSVP）
- QoS控制面应当满足的控制功能

QoS控制面的功能（与别的控制面不要混淆，如SDN）

名称	解释
准入控制	判断用户的数据**是否能够进入网络**
QoS路由	确定能够满足流QoS需求的**网络路径**
资源预留	对网络性能有要求的流按需**预留网络资源的机制**

DiffServ Field (差别处理)

- IPv4/6 分组首部DS字段对应用的QoS等级进行标记，用于区分不同等级
- DS字段的值成为码点 DSCP（Code Point）
- 6bit 码点可定义64种不同类别的流量
- 3类：xxxxx0用于标准方式来使用；xxxx11用于实验或本地使用；xxxx01用于实验或本地使用 ，但是可以根据需求分配

IP Performance Metrics（IP性能测度）

- 定义：一组标准而有效的测度能够让**用户和服务提供商**在因特网和私有互联网的**性能方面**达成明确共识
- 用途：
  - 支持大规模复杂互联网的**容量规划**和**故障定位**
  - 通过统一的测度对比来**促进**服务提供商之间的**竞争**
  - 支持在 **协议设计**、**拥塞控制**、**QoS** 等领域的因特网研究
  - 可以验证 **SLA**
- 测量方式
  - 独立测度：可测量的最基本或者最小的性能**测度数值**（如；单个分组经历的时延）
  - 抽样测度：在给定的时间周期内独立测量的**集合**（如；一个周期内所有独立测度的时延集合）
  - 统计测度：从抽样测度通过计算得到的**统计特征值**（如；一个周期内抽样测度的时延平均值）
- 测量技术
  - 主动技术：**主动注入**流量进行测量
  - 被动技术：对**现有的流量进行观测**

QoE 体验质量

- 用户主观性能评价，对多媒体应用和多媒体内容传输特别重要

分类	细节
感知方面	清晰度、音色
心理方面	使用难易度、兴奋度、厌倦度
交互方面	响应性，自然性

拥塞控制

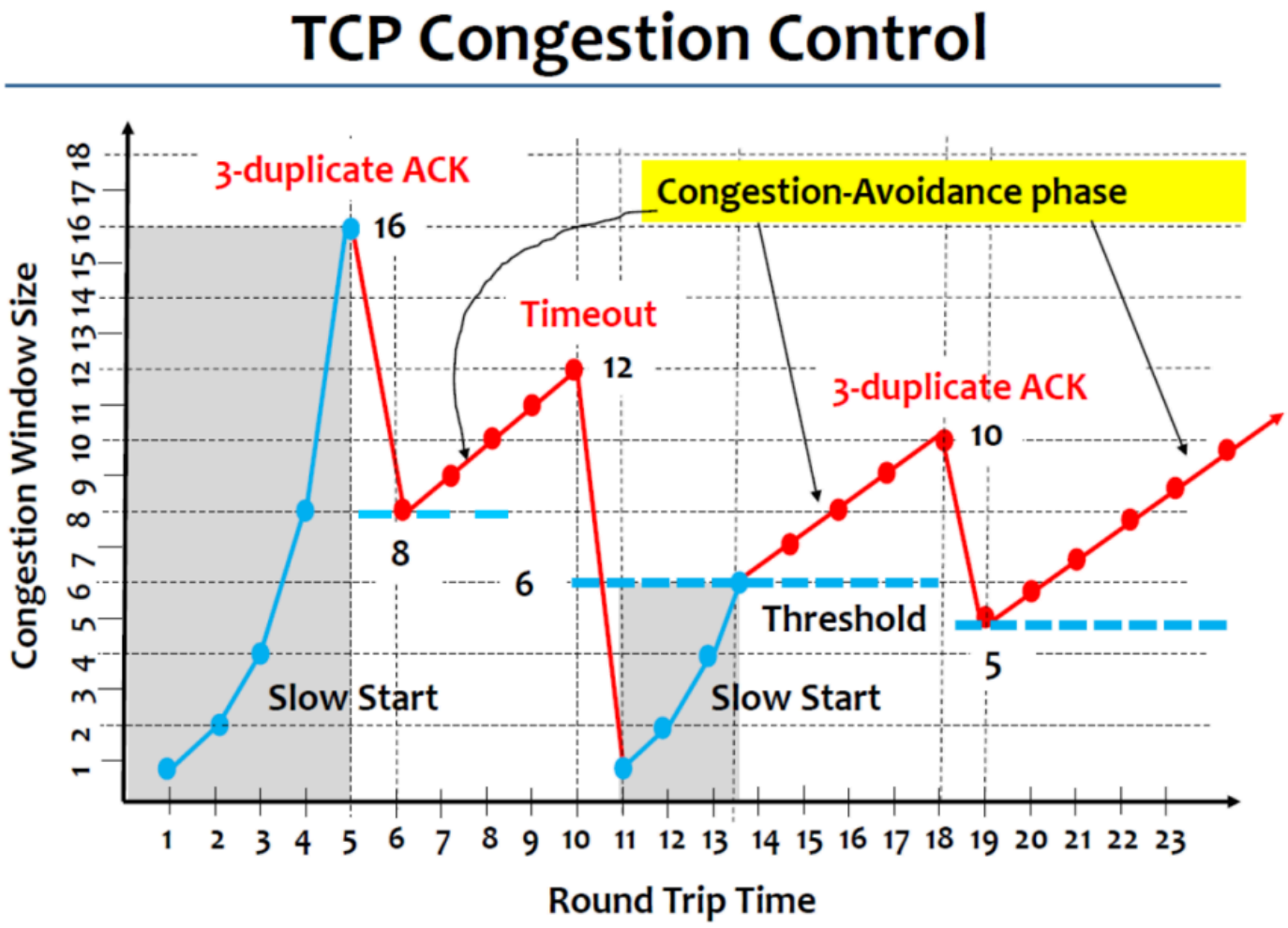
拥塞控制策略分类：

- 准入控制策略
- 反向压力
- **隐式阻塞信令**：根据隐式的拥塞信号进行拥塞控制，不需要对网络结点进行操作，例如 **TCP-ACK**
- **抑制分组**：抑制分组是一个**控制分组**；由出现了**拥塞的结点产生**；发送给**源结点**来**限制流量**；路由器和目的端系统能够将**抑制分组**发送给**源端系统**；接收到抑制分组后，源主机会立即减少发送给目的主机的流速，直至不再接收到抑制分组为止，例如，**ICMP源抑制报文**
- **显示拥塞通知**：
  - 方向：后向通知源端；前项通知目的端
  - 内容：标志位，信用，速率
  - 例如：P4实现ECN（Explicit Congestion Notification）

TCP拥塞控制机制：

cwnd：一次发送的包数量；ssthresh：慢启动门限（ssthresh大于等于2）

- 慢启动：cwnd<ssthresh时，cwnd以2的指数次方增长，知道cwnd等于ssthresh
- 拥塞避免：当cwnd大于等于ssthresh时，每发送cwnd个包，且不存在阻塞，下一次cwnd加1；若发生阻塞，sshresh变成cwnd/2，cwnd变成1
- 快恢复和快回复：收到三个重复的ACK，立即重传，并且更新sshresh为cwnd/2，更新cwnd为cwnd/2



ICMP源抑制报文

IP协议中，ICMP源报文抑制（Source Quench）报文来反馈抑制分组，实现拥塞控制

- 当路由器或目标主机因**拥塞而丢弃IP数据报时**，它会向数据报的源节点发送一个ICMP抑制报文，通知源节点数据报已经被丢弃
- 源节点接收到后，会**降低发送往目标节点的传输速率**，以缓解拥塞
- 如果一段时间内源头节点没有接收到新的源抑制报文，源节点拥塞被认为已经解除，并**逐渐恢复**到原来的数据报流量

SDN（software defined network）

软件定义网络（软件定义指的是能够通过 SDN 控制器来定义网络）

实现目标

- 实现**数据平面**和**控制平面**的分离
  - 控制平面：负责**智能路由设计、设置优先级、路由策略参数**以**满足QoS和QoE的要求**
  - 数据平面：负责**分组转发 （数据平面仅仅负责分组转发）**
- SDN的特征
  - **控制平面与数据平面相互分离**，数据平面设备成为**只进行分组转发**的设备
  - 控制平面在**集中式控制器**或一组**协作的控制器**上实现，**SDN控制器**拥有网络或它所控制网络的**全局视图**
  - 控制平面设备（控制器）和数据平面设备（虚拟\物理交换机）之间的**开放接口**（OpenFlow）已经得到定义
  - 网络可由运行在SDN控制器之上的应用进行编程，SDN控制器向**应用**提供了**网络资源的抽象视图**

SDN体系结构

- 三个平面
  - 数据平面：物理交换机和虚拟交换机组成，负责转发分组
  - 控制平面：有SDN控制器组成
  - 应用平面：由多个与SDN控制器相交互的应用，包括**安全应用、网络应用、业务应用**，具体的应用案例包括**节能网络、安全监视、访问控制**和**网络管理**
- 四个接口
  - **南向接口**：SDN控制器和数据平面（交换机）之间的接口；如：OpenFlow 由控制平面向数据平面发布流表项
  - **北向接口**：网络应用和SDN控制器进行通信（如：RESET API）
  - **东向/西向接口**：控制器组或者联盟之间可以通信和协作，从而同步状态以达到更高的实用性

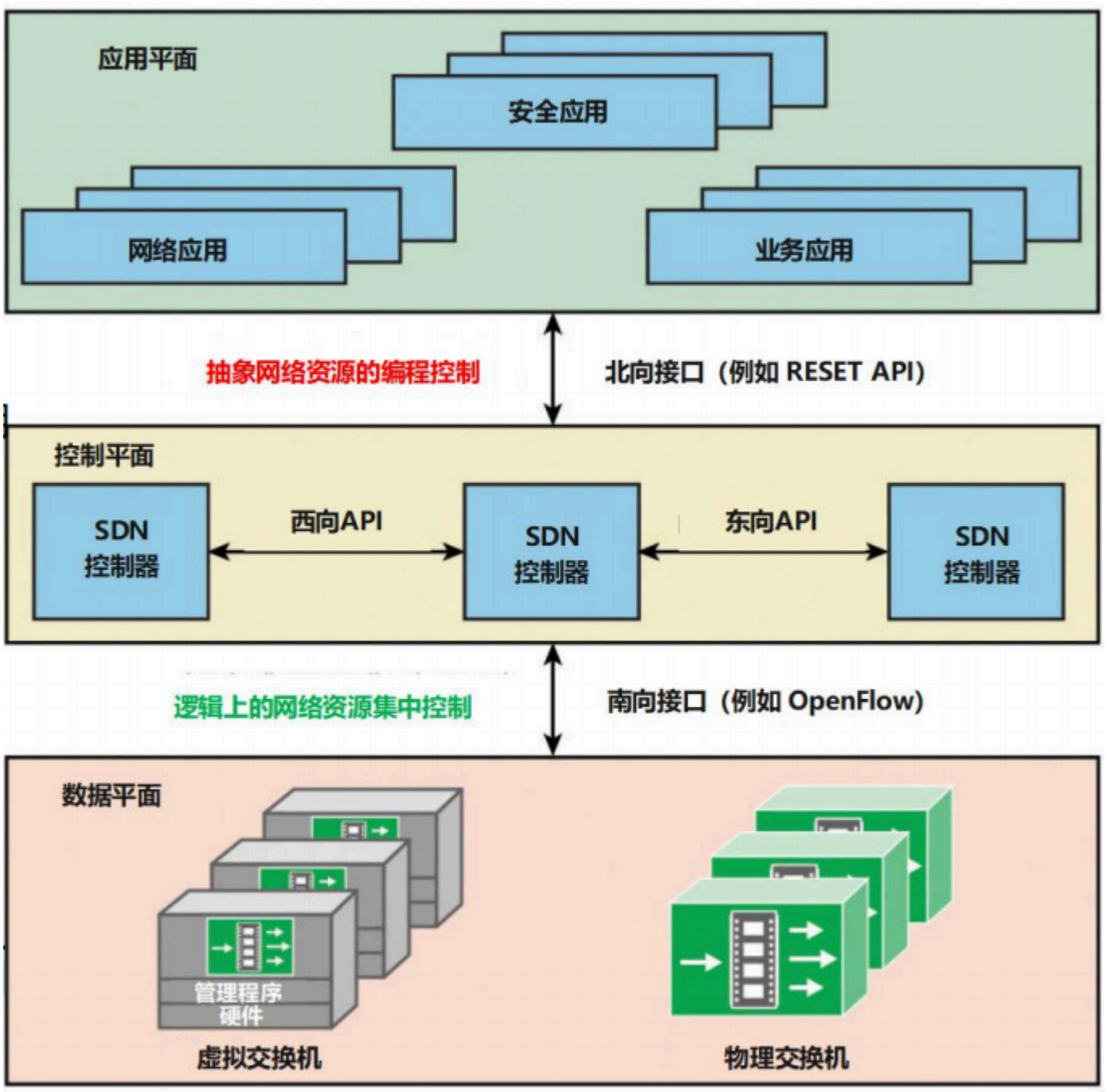


图3-3 软件定义的体系结构

SDN控制平面

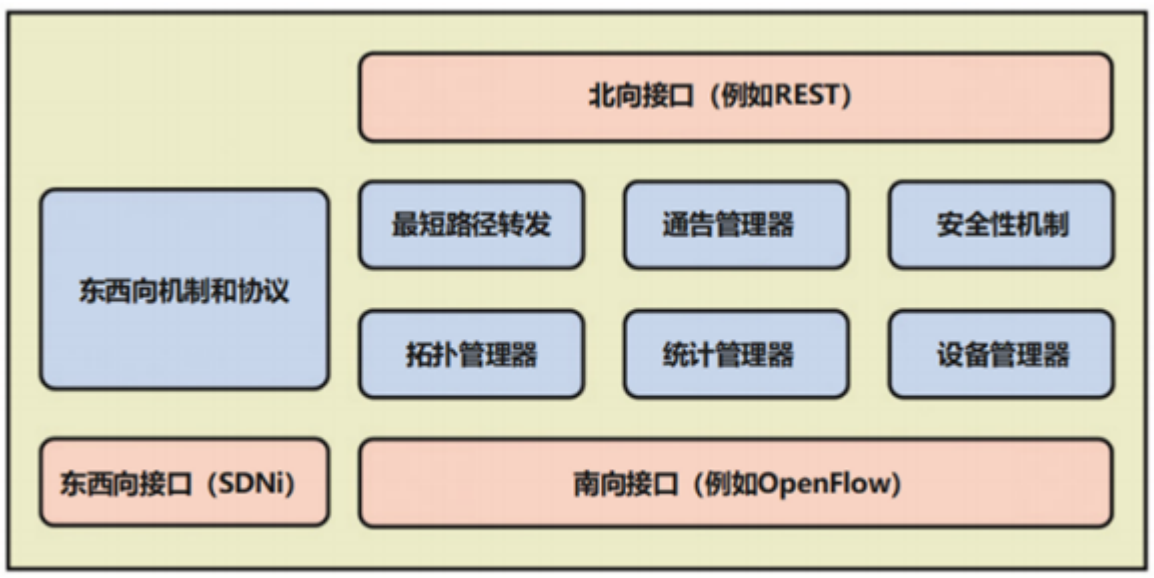


图5-2 SDN控制平面功能和接口

SDN数据平面

- 主要功能
  - **控制支撑**功能：与SDN控制层通过 **OpenFlow 协议**进行交互，控制层能够通过 OpenFlow 接口控制转发
  - **数据转发**功能：依据**SDN控制器发布的规则**转发从其他网络设备和端系统**接受和转发**的数据流
- **OpenFlow 交换机**的结构

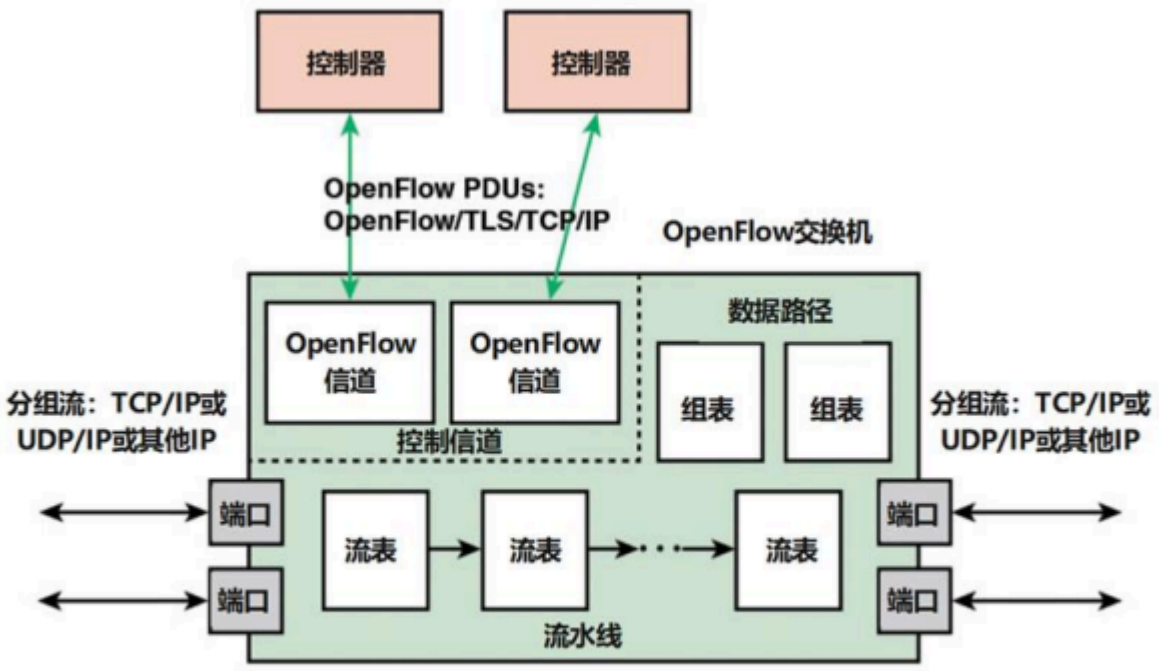


图4-4 OpenFLow交换机

  - OpenFlow 信道：控制器与交换机的管理接口，控制器通过OpenFlow信道发布流表项，管理交换机的转发
  - OpenFlow 端口：分组进入和离开的位置（对应P4中的 Ingress 和 Egress）
  - 表结构：流表（flow）、组表（group）、计量表（meter）

虚拟机

- VM（Virtual Machine）：运行在一台计算机上的**操作系统实例**，这些操作系统实例能运行各自的应用程序，且相互隔离
- VMM（Virtual Machine Monitor/Manager）：一种用于提供虚拟机环境的系统程序，又称 **hypervisor**
- 两种虚拟机
  - type1：部署在物理主机上，能够**直接控制主机的物理资源**
  - type2：部署在主机OS之上，需要**依赖OS和硬件交互**
- 虚拟机的优势：**（虚拟机是大数据和云计算基础设施的核心要素）**
  - 固节比（consolidation ration）：在同一台主机上运行的虚拟机的数量；相比传统的一台服务器上的只能运行一种应用而言，其优势为
    - 节省硬件资源
    - 节省能源（含冷却）
    - 节省光缆、场地、交换机等
  - 服务器资源虚拟化
    - 硬件资源灵活定制
    - 快速迁移（负载均衡，冗余备份）
- 虚拟机的特征

    - 虚拟机是一种**模拟物理服务器特征**的**软件**，它会被分配一些处理器、内存、存储资源和网络连接
    - 虚拟机之间相互**隔离**
    - 虚拟机由文件构成：**配置文件**用于描述虚拟机的属性；**附加文件**为虚拟磁盘，开机或实例化后创建
    - 虚拟机主机集中在一起形成了**计算机资源池**
- 容器（虚拟化技术并非虚拟机）
    - 将**应用程序底层操作环境**进行虚拟化的技术
    - 容器运行在**同一个内核**上，共享绝**大部分操作系统**
    - 容器比虚拟机更加**轻量级**

NFV架构

- VNFs（虚拟网络功能）：网络功能，如，CSCF、MPLS
- NFVI（NFV基础设施）：虚拟硬件以及物理硬件
- NFV管理和编排

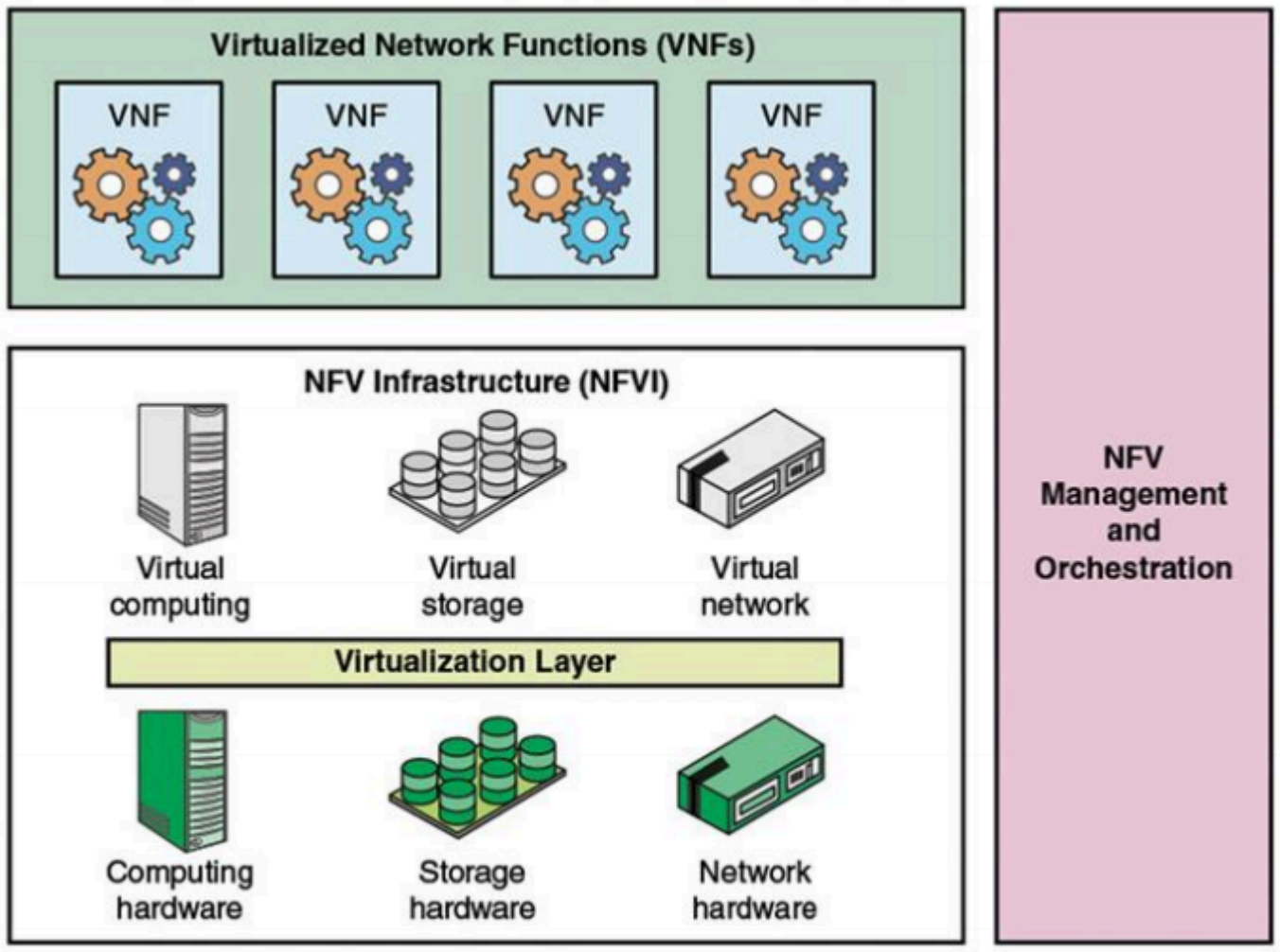
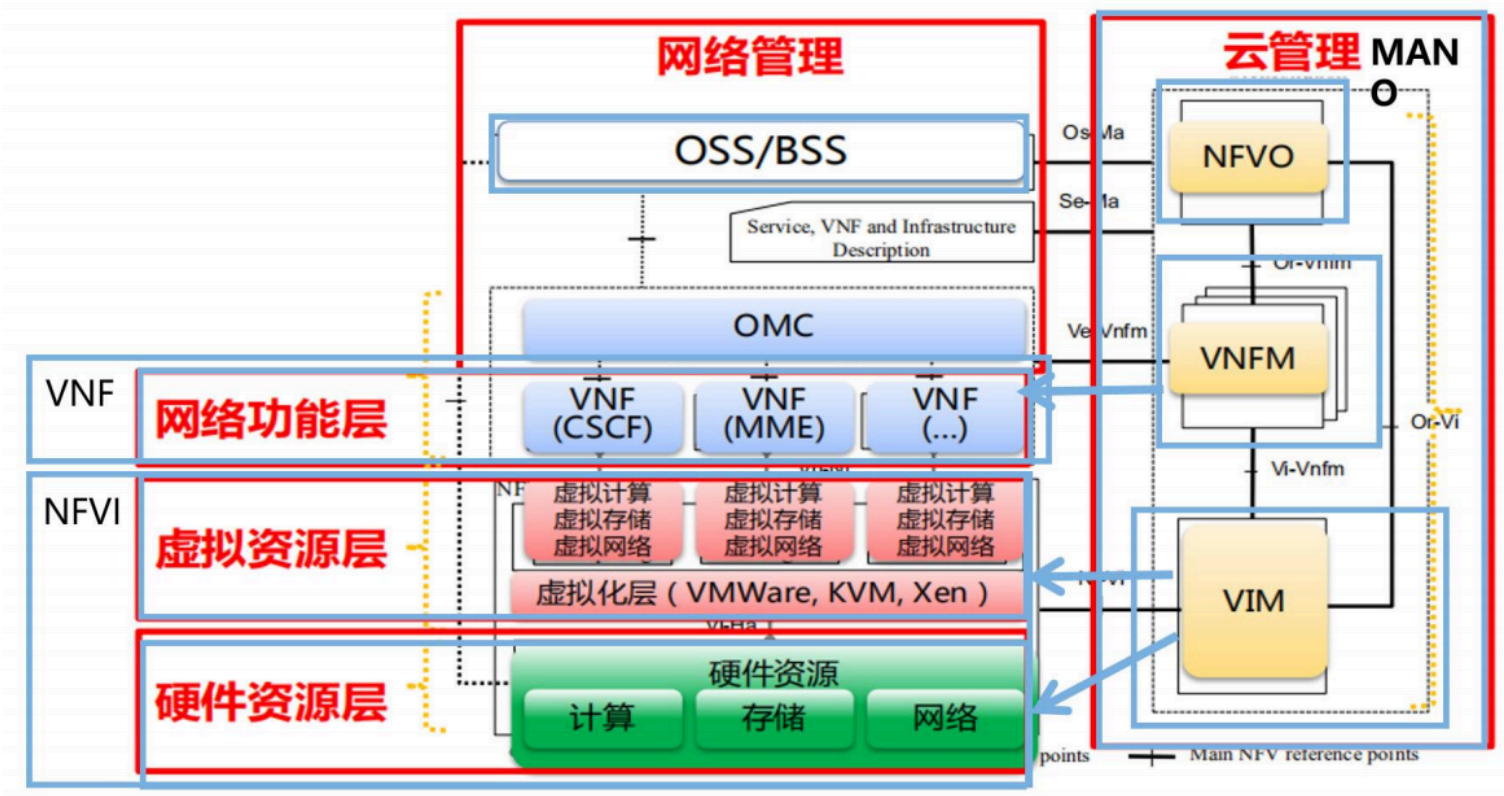


FIGURE 7.7 High-Level NFV Framework

IMS的NFV实例



P4基础

P4代码的基本结构

- 解析器（Parser）
- 验证校验和（VerifyCheckSum）
- 进入端口处理（Ingress）
- 出端口处理（Egress）
- 计算校验和（ComputeCheckSum）
- 逆解析（将包内容整合）(Deparser)

基本数据类型

- bit<n>：无符号类型
- int<n>：有符号类型
- header：有序的成员集合，能够通过 `packet.extract(/*header*/)` 来自动匹配头部字段，若匹配成功，则会将header中的隐藏成员 Valid 至为1，并且将包中的对应代码填充到header中
- struct：无序的成员集合
- header stack：一组header
- header union：一些header中的某一个单元

P4的预定义数据类型

- 元数据 ： 用于携带数据和配置性的数据
  - 用户自定义元数据：用来携带P4程序运行过程中用户定义的数据
  - 固有元数据 `standard_metadata_t` ：用于携带交换机自身的配置信息；固有元数据中有三项基本信息需要记忆，`bit<9> ingress_port`、`bit<9> egress_spec`、`bit<9> egress_port`
- `packet_in` 类型：其是一个预定义的结构，专门用于在**解析器（parser）**中处理输入数据包的原始字节流。其核心作用是提供对数据包底层字节的访问和操作接口，帮助开发者提取协议头部字段或进行数据包验证
  - `extract()` 方法：能够从数据包中按位/字节提取字段，填充到自定义头部结构中，括号中传入自定义的首部中

Parser解析器（要求根据状态机写代码/根据代码画状态机）

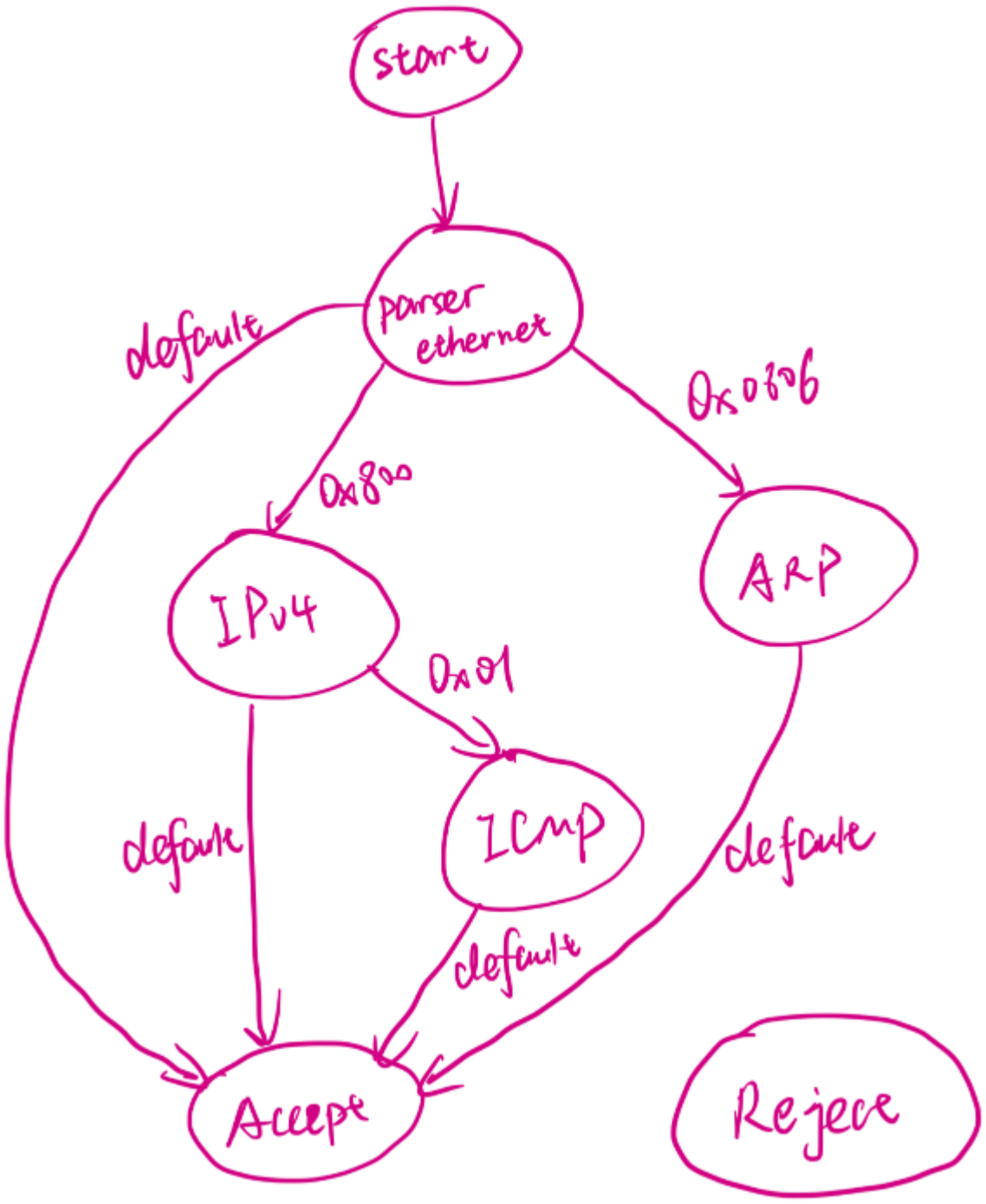
- Parser描述了一个状态机，起始有 start 一个状态，终止有 accept 和 reject 两个状态
- 每个状态解析一种协议的头部，分层解析，例如：一个三层数据报，先解析以太网头部，发现是IP数据包，再解析IP头部，头部的解析使用 `packet.extract(/*header*/)` 方式，将头部信息放入对应的首部结构中。
- Parser的语法：

    - state 语法：用于定义状态
    - transition 语法：用于状态转换
        - `transition accept/reject;`：用于跳转最终状态
        - `transition select(\* word *\){}`：用于根据首部字段跳转对应状态
    - `packet_in.extract();` 语法：用于解析首部，将首部放入预先定义的首部数据结构中
- Parser 一个 state 对应一种首部解析和条件选择判断跳转状态的方向

Parser示例：

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata)
{
    @name("_start") state start
    {
            transition parse_ethernet;
    }

    @name("_parse_ethernet") state parse_ethernet
    {
        //解析以太网头部
        packet.extract(hdr.ethernet);
        //transition select()用于选择 相当于 c语言中的switch case语句
        transition select(hdr.ethernet.etherType)
        {
            16w0x800:parse_ipv4;
            16w0x0806:parse_arp;
            default:accept;
        }
    }

    @name("_parse_arp") state parse_arp
    {
        //解析ARP头部
        packet.extract(hdr.arp);
        transition accept;
    }

    @name("_parse_ipv4") state parse_ipv4
    {
        //解析ipv4头部
        packet.extract(hdr.ipv4);
        //选择下一个状态
        transition select(hdr.ipv4.protocol)
        {   8w0x01:parse_icmp;
            default:accept;
        }
    }
    @name("_parse_icmp")state parse_icmp
    {
        //解析icmp头部
        packet.extract(hdr.icmp);
        transition accept;
    }
}
```

以上parser的状态机可以表示为



控制块--Ingress、Egress

Action 动作

相当于 C 语言中的函数，能够将一些运算封装成一个动作；能够再apply模块中调用，也可以再table中与匹配项对应

table 表

- 成员：
  - key：描述匹配规则，哪一些字段的匹配，使用哪些规则匹配
  - actions：动作列表，描述匹配后的动作集合
  - default_action：默认动作
  - size：表的大小期望

```
//前往下一跳
action set_nhop(bit<48> dstAddr,bit<9> port) {
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr= dstAddr;
    standard_metadata.egress_spec = port;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    p4_logger(hdr.ipv4.ttl);
}
//丢弃包
action drop() {
        mark_to_drop(standard_metadata);
}
//表成员定义
table ipv4_lpm {
    //匹配规则
    key = {
        hdr.ipv4.dstAddr:lpm;
    }
    //动作集合
    actions = {
        set_nhop;
        drop;
    }
```

```
    size = 1024;
    default_action = drop();
}
```

- 添加表项命令 `table_add` 命令，根据上述表定义的示例，我们可以使用以下命令添加表项

```
/添加流表/  /表名称/ /动作名称/ / 匹配值 /     /动作参数1/ /动作参数2/
table_add ipv4_lpm set_nhop 22.1.11.29 => 22.1.12.29 2
```

由于有动作有两个参数，所以表项的传入参数也有两个；从上述实例可以看出，p4中的表只是**定义成员列表**，和**匹配规则**，**具体　　表项内容未在程序中体现**

- 添加方式
  1. 在终端中输入

```
sudo simple_switch_CLI --thrift-port 9090 //9090 为对应的交换机的端口号
```

     输入 `table_add` 命令，能够手动添加流表项

  2. 在 `cmd.txt` 文件中事先将流表添加命令写好，输入 `p4run` 的时候系统将会执行对应的命令添加流表项

apply 模块

apply模块相当于主函数，当匹配动作完成后能够再apply模块通过编码实现特定的功能

示例：若当前交换机能够解包 arp，icmp，ipv4 等协议时，我们Ingress中的apply模块编写如下：

```
apply{
    if(hdr.arp.isValid())//判断arp首部是否匹配
    {
        if(hdr.arp.op_code == 1)//ARP协议中 opcode(操作码)为1表示arp请求，opcode=2表示arp应答
        {
            if(t_handle_ARP.apply().hit)//检查arp表项是否匹配
            {
                return;//应答完程序结束
            }
        }
    }
    if(hdr.icmp.isValid())//icmp是否匹配
    {
        if(hdr.icmp.type == 8)//icmp type为8时表示icmp请求
        {
            if(t_handle_icmp.apply().hit)//icmp 应答，检查icmp表项是否匹配
            {
                return;//结束
            }
            else
            {
                ipv4_lpm.apply();//直接转发
            }
        }
        else
        {
            ipv4_lpm.apply();//直接转发
        }
    }
    else
    {
        ipv4_lpm.apply();//直接转发
    }
}
```

注意上述代码中 `\*table*\.apply().hit` 和 `\*table*\.apply()` 的区别在于，**前者能够判断是否能够匹配表项**；

完整的Ingress实例如下：

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }
    //mac转发动作
    action forward(bit<9> port) {
        standard_metadata.egress_spec = port;
    }
    //ipv4转发动作
```

```
    action set_nhop(bit<48> dstAddr,bit<9> port) {
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr= dstAddr;
        standard_metadata.egress_spec = port;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
        p4_logger(hdr.ipv4.ttl);
    }
    //arp应答动作
    action send_arp_reply(bit<48> srcMACAddr) {
        hdr.ethernet.dstAddr = hdr.arp.src_mac;
        hdr.ethernet.srcAddr = srcMACAddr;
        hdr.arp.setValid();
        hdr.arp.op_code = 2;
        bit<32> ip_temp = hdr.arp.dst_ip;
        hdr.arp.dst_ip = hdr.arp.src_ip;
        hdr.arp.dst_mac = hdr.arp.src_mac;
        hdr.arp.src_mac = srcMACAddr;
        hdr.arp.src_ip = ip_temp;
        standard_metadata.egress_spec = standard_metadata.ingress_port;
    }
    //icmp应答动作
    action send_icmp_reply() {
        bit<48> mac_temp = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
        hdr.ethernet.srcAddr = mac_temp;

        bit<32> ip_temp = hdr.ipv4.dstAddr;
        hdr.ipv4.dstAddr = hdr.ipv4.srcAddr;
        hdr.ipv4.srcAddr = ip_temp;

        hdr.icmp.type = 0;
        standard_metadata.egress_spec = standard_metadata.ingress_port;
    }
    //ipv4动作匹配表
    table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr:lpm;
    }

    actions = {
        set_nhop;
            drop;
        }
        size = 1024;
    default_action = drop();
    }
    //arp动作匹配表
    table t_handle_ARP {
    key = {
        standard_metadata.ingress_port:exact;
        hdr.arp.dst_ip:exact;
    }

    actions = {
            send_arp_reply;
            drop;
        }
        size = 100;
    default_action = drop();
    }
    //以太网帧动作匹配表
    table mac_forward {
        key = {
            hdr.ethernet.dstAddr: exact;
        }

        actions = {
            forward;
            drop;
        }
        size = 1024;
        default_action = drop();
    }
    //icmp动作匹配表
    table t_handle_icmp{
    key = {
            hdr.ipv4.dstAddr: exact;
        }
```

```
    actions = {
            send_icmp_reply;
            drop;
        }
    }
    //执行匹配
    apply{
        if(hdr.arp.isValid())
        {
            if(hdr.arp.op_code == 1)
            {
                if(t_handle_ARP.apply().hit)
                {
                    return;
                }
            }
        }
        /**************************************
            若为ICMP报文，且为请求类型报文，且满足表中匹配则进行 ICMP回复
            否则当作ipv4进行转发
        **************************************/
        //判断是否符合icmp报文
        if(hdr.icmp.isValid())
        {
            //若为ICMP请求报文
                if(hdr.icmp.type == 8){
                    //判断是否匹配，若匹配进行转发
                    if(t_handle_icmp.apply().hit){
                        return;
                    }
                    //icmp未匹配，将该数据报当作ipv4进行转发
                    else{
                        ipv4_lpm.apply();
                    }
                }
                else{
                    ipv4_lpm.apply();
                }
            }
            else{
                ipv4_lpm.apply();
            }
        }

}
```

设计案例

- 二层交换
  - 二层交换只是进行以太网帧的转发，匹配以太网目的地址后，进行目的端口的转发

```
action ethernet_forward(bit<9> port){
    standard_metadata.egress_spec = port; //将标准数据中的输出端口标记为对应的端口
}
//定义表项
table mac_forward{
    //匹配规则
    key = {hdr.ethernet_t.dstAddr:exact;}
    actions = {
        ethernet_forward;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

    上述程序中 standard_metadata 是标准元数据，其在 `<v1modle.p4>` 等适配不同交换机的文件中，其标识了进入**交换机的数据报的控制与状态信息**，如输入输出端口

- 三层交换
  - 三层交换机进行IPv4数据包的转发，IPv4转发先改变数据报的mac源目的地址，之后需要将IP数据报的 ttl 减少 1，然后选择目的端口；
  - 匹配规则应当是最长掩码匹配，即 lpm

```
action ipv4_forward(bit<48> dstAddr,bit<9> port){
    hdr.ethernet_t.srcAddr = hdr.ethernet_t.dstAddr;
```

```
    hdr.ethernet_t.dstAddr = dstAddr;
    hdr.ipv4_t.ttl = hdr.ipv4_t.ttl-1;
    standard_metadata.egress_spec = port;
}

table ipv4_forward_t{
    key = {hdr.ipv4_t.dstAdr:lpm}//匹配规则为最长掩码匹配法
    actions = {
        ipv4_forward;
        drop;
    }
    size = 1024;
    default_action = drop();
}

/*以下为添加表项，注意添加表项的时候，需要标识掩码多少位*/
table_add ipv4_lpm set_nhop 22.1.11.29/32 => 00:00:16:01:0b:1d 1
table_add ipv4_lpm set_nhop 22.1.12.29/32 => 00:00:16:01:6f:1d 2
```

- ICMP报文回复（用于验证网络连通性，在交换机中 h1 ping h2 就是使用ICMP协议来验证网络连通性）
  - ICMP回复报文需要**交换MAC和IPv4的源目的地址**，然后将ICMP的类型标识**type更改为0（8为请求，0为回复），转发端口为输入端口**，因为为应答信号;
  - ICMP报文的匹配规则应当是IPv4的目的地址，当完全匹配时进行应答

```
action icmp_reply(){
    bit<48> temp_mac = hdr.ethernet_t.dstAddr;
    hdr.ethernet_t.dstAddr = hdr.ethernet_t.srcAddr;
    hdr.ethernet_t.srcAddr = temp_mac;

    bit<32> temp_ip = hdr.ipv4_t.dstAddr;
    hdr.ipv4_t.dstAddr = hdr.ipv4_t.srcAddr;
    hdr.ipv4_t.srcAddr = temp_ip;

    hdr.icmp.type = 0;
    //输出端口就是输入端口，因为是一个应答
    standard_metadata.egress_spec = standard_metadata.ingress_port;
}

table icmp_handle_t{
    key = {hdr.ipv4_t.dstAddr:exact;}
    actions = {
        icmp_reply;
        NoAction;
    }
    default_action = NoAction();
    size = 1024;
}
```

- ARP报文回复
  - ARP协议用于绑定网络设备的MAC地址和IP地址，ARP报文中包含op_code用于标识ARP协议是发送还是接收，并且设置了发送方IP、MAC，接收方IP、MAC用于传输绑定的IP和MAC信息；另外，ARP协议属于二层协议，其回复报文需要交换数据报中的MAC目的和源地址；最后ARP协议是已知设备的IP地址，绑定设备的MAC地址，所以匹配项应当是ARP报文中的目的IP地址；

```
action arp_reply(bit<32> srcMacAddr){
    hdr.ethernet_t.dstAddr = hdr.arp.senderAddr;
    hdr.ethernet_t.srcAddr = srcMacAddr;

    bit<32> ip_temp = hdr.arp.senderIPAddr;
    hdr.arp.senderIPAddr = hdr.arp.targetIPAddr;//发送ip交换
    hdr.arp.targetIPAddr = ip_temp;//目的ip交换
    hdr.arp.targetMacAddr = hdr.arp.senderMacAddr;//target mac和sender mac交换
    hdr.arp.senderMacAddr = srcMacAddr;//发送方是我方

    hdr.arp.op_code = 2;//1标识请求，2标识响应

    standard_metadata.egress_spec = standard_metadata.ingress_port;
}

table arp_handle_t{
    key = {
        hdr.arp.targetIPAddr:exact;
        standard_metadata.ingress_port:exact;
    }
    actions = {
        arp_reply;
        NoAction;
```

```
    }
    default_action = NoAction();
    size = 1024;
}
```

不同协议的表项总结

协议	匹配项	动作参数	动作描述
二层转发	ethernet 的目的 MAC 地址：exact 匹配	转发端口 号	将 egress_spec 设置为表项中传入的端口号
三层转发	IP 首部的目的 IP 地址：lpm 匹配	下一跳MAC地址； 转发端口号	改变ethernet首部的目的和源地址；IP 首部 TTL 字段减 1
ARP 回复	ARP 首部的目的 IP 地址：exact 匹配；进入端口号：exact 匹配	目的 MAC 地址	交换 ethernet 首部的源/目的地址；更改 ARP 报文中的 op_code 为2；更改目的 MAC地址 为源 MAC 地址；更改 ARP 报文中的 源MAC地址 为此交换机 MAC；交换源/目的IP地址；设置 egress_spec 为传入的端口
ICMP 回复	IP 首部的目的 IP 地址：exact 匹配	无	交换 ethernet 首部的源/目的地址；交换 IP 报文的源/目的地址；设置 ICMP 报文首部 Type 字段为 0 ；设置 egress_spec 为传入的端口

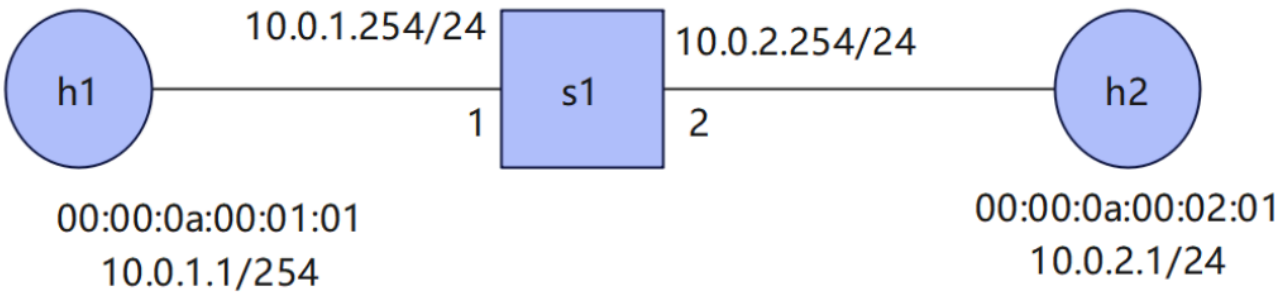
Deparser 逆解析器

- Deparser是一个控制块
- 其功能是将首部组成一个完整的数据报
- packet_out 外部函数定义在 core.p4中，其中有一个函数是 emit(hdr)：其功能为若首部有效，将其序列化加入该首部

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
      packet.emit(hdr.ethernet);
      packet.emit(hdr.ipv4);
      packet.emit(hdr.arp);
      packet.emit(hdr.icmp);
    }
}
```

相关习题

表项填充




```
header ethernet_t{
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16> etherType;
}

header ipv4_t{
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    bit<32> srcAddr;
    bit<32> dstAddr;
}

struct headers{
    ethernet_t ethernet;
    ipv4_t ipv4;

}
```

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata)
{
    action set_nhop(bit<48> dstAddr, bit<9> port){




    }
    action drop(){
        mark_to_drop(standard_metadata);
    }
    table ipv4_lpm{




    }
    apply{
        ipv4_lpm.apply();
    }
}
```

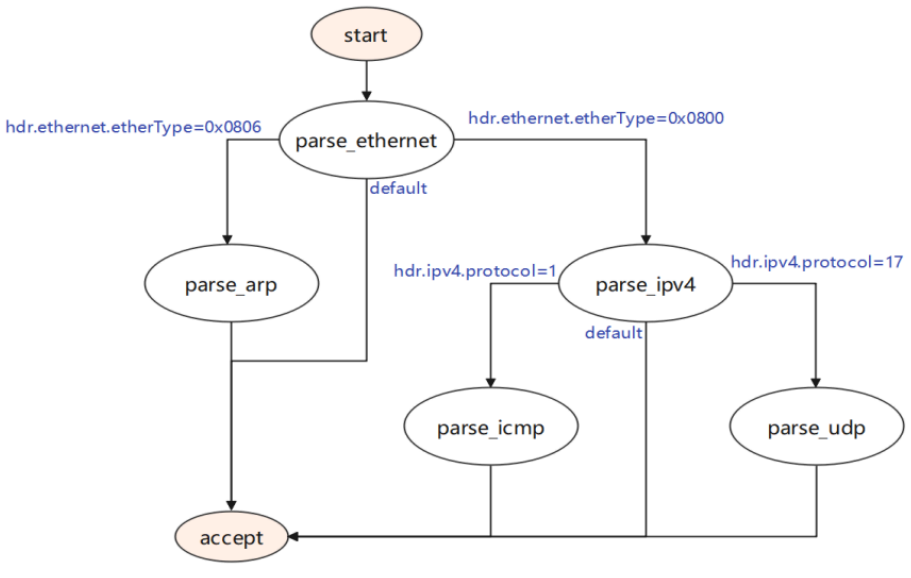
```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata)
{
    action set_nhop(bit<48> dstAddr,bit<9> port){
        hdr.ethernet.srcAddr = hdr.ethernet,dstAddr;
        hdr.ethernet,dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl-1;
        standard_metadata.egress_spec = port;
    }

    action drop(){
        mark_to_drop(standard_metadata);
    }

    table ipv4_lpm{
        key = {hdr.ipv4.dstAddr:lpm}
        actions = {
            set_nhop;
            drop;
        }
        size = 1024;
        default_action = drop();
    }
    apply{
        ipv4_lpm.apply();
    }
}

/*table add 命令*/
table_add lpv4_lpm set_nhop 10.0.1.254/24 => 00:00:0a:00:01:01 1
table_add lpv4_lpm set_nhop 10.0.2.254/24 => 00:00:0a:00:02:01 2
```

Parser考点

给状态机写代码 or 给代码画状态机

eg1：给出状态机写代码

- 状态 和状态机
  - Start
  - parse_etherne
  - parse_arp
  - parse_ipv4
  - parse_icmp
  - parse_udp
  - accept




```
parser MyParser(packet_in packet,
                out myheaders hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata){
    state start{
        transition parse_ethernet;
    }

    state parse_ethernet{
        packet_extract(hdr.ehernet);
        transition select(hdr.ehernet.etherType){
            16w0x0800:parse_ipv4;
            16w0x0806:parse_arp;
            default:accept;
        }
    }

    state parse_arp{
        packet_extract(hdr.arp);
        transition accept;
    }

    state parse_ipv4{
        packet_extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol){
            8w0x01:parse_icmp;
            8w0x11:parse_udp;
            default:accept;
        }
    }

    state parse_icmp{
        packet_extract(hdr.icmp);
        transition accept;
    }

    state parse_udp{
        packet_extract(hdr.udp);
        transition accept;
    }
}
```