Flannel 网络初步探索

Name : 曲中岭

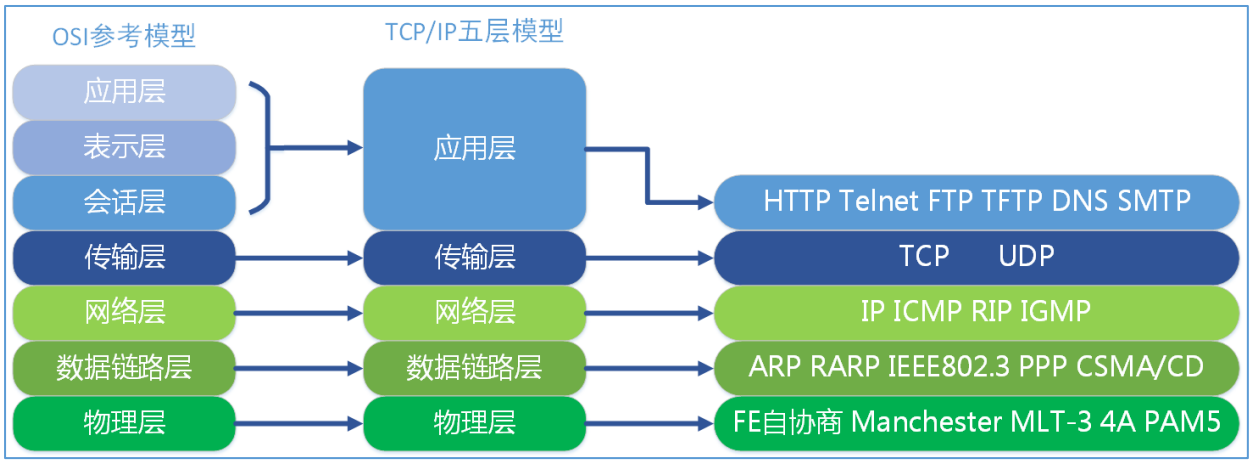
Email:zlingqu@126.com

Q Q :441869115

第一章 OSI 参考模型

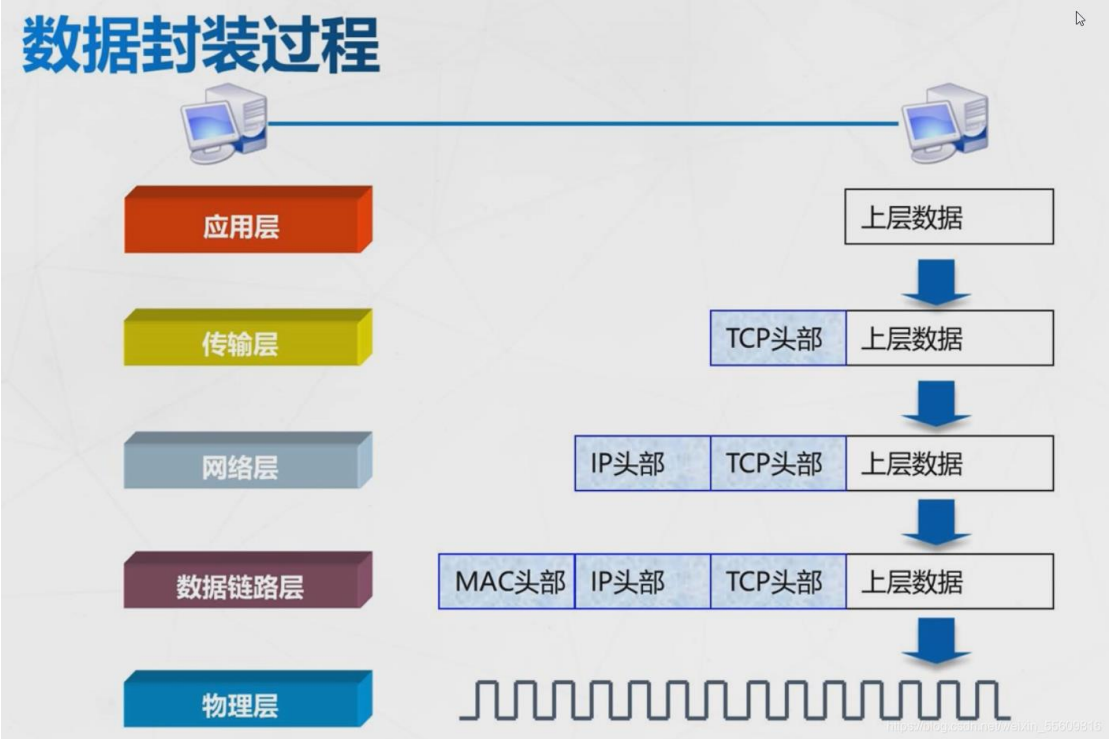
1.1 OSI 和 TCP/IP 模型

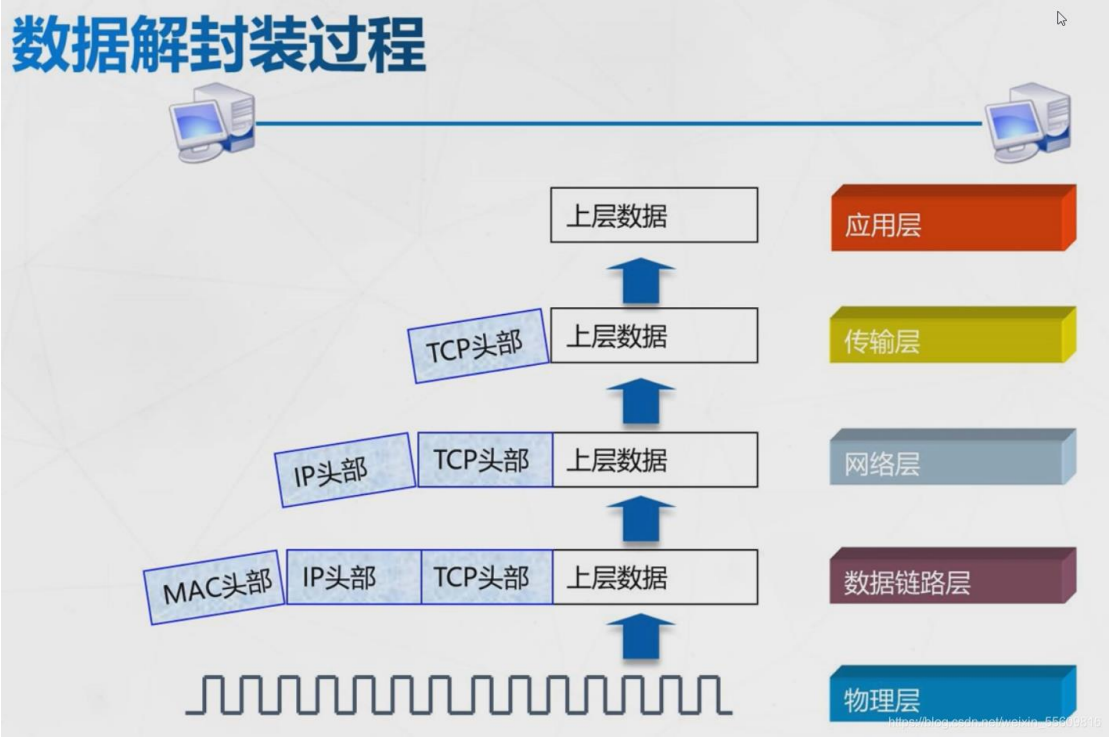
- OSI 模型，即开放式通信系统互联参考模型(Open System Interconnection)
- TCP/IP（Transmission Control Protocol/Internet Protocol，传输控制协议/网际协议）是指能够在多个不同网络间实现信息传输的协议簇。TCP/IP 协议不仅仅指的是 TCP 和 IP 两个协议，而是指一个由 FTP、SMTP、TCP、UDP、IP 等协议构成的协议簇， 只是因为在 TCP/IP 协议中 TCP 协议和 IP 协议最具代表性，所以被称为 TCP/IP 协议。



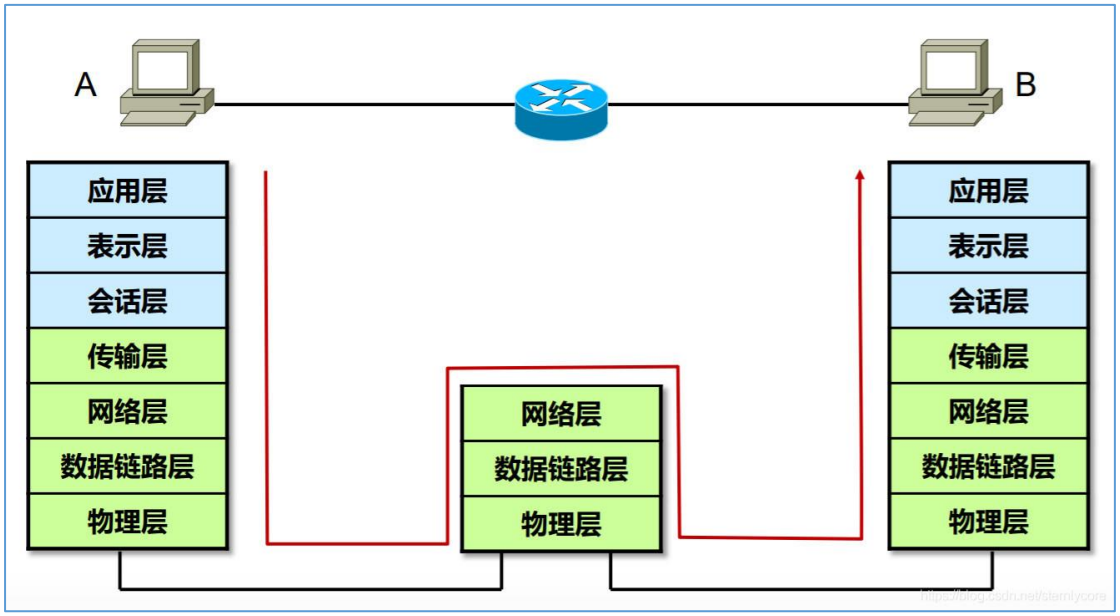
1） 物理层 Physical Layer
2） 数据链路层 Data Link Layer
3） 网络层 Network Layer
4） 传输层 Transport Layer
5） 会话层 Session Layer
6） 表示层 Presentation Layer
7） 应用层 Application Layer

1.2 数据封装和解封


数据封装过程
应用层
传输层
网络层
数据链路层
物理层
上层数据
TCP头部
上层数据
IP头部
TCP头部
上层数据
MAC头部
IP头部
TCP头部
上层数据

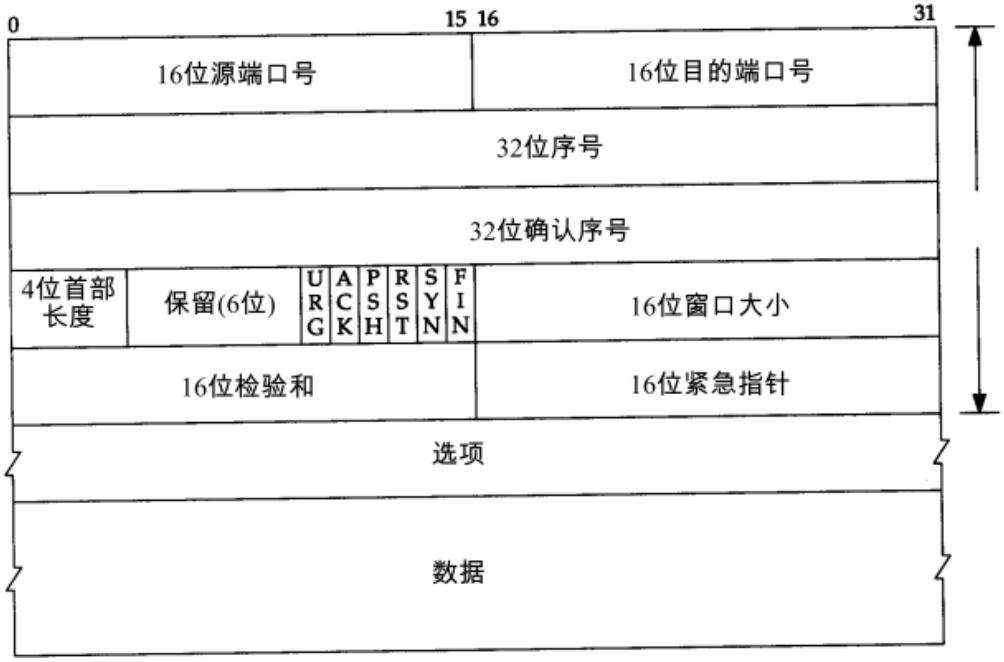

数据解封装过程
上层数据
应用层
TCP头部
上层数据
传输层
IP头部
TCP头部
上层数据
网络层
MAC头部
IP头部
TCP头部
上层数据
数据链路层
物理层

1.3 有路由器存在的情况



可以看到路由器设备只工作在 OSI 的最下面三层。

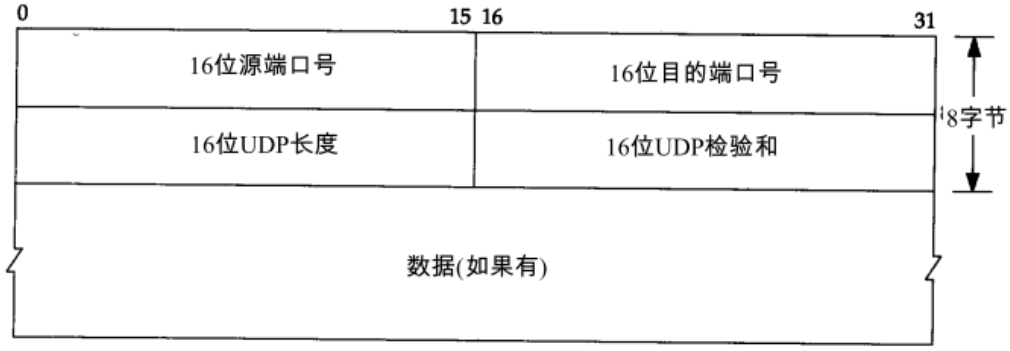
1.4 TCP 头部



关键点：

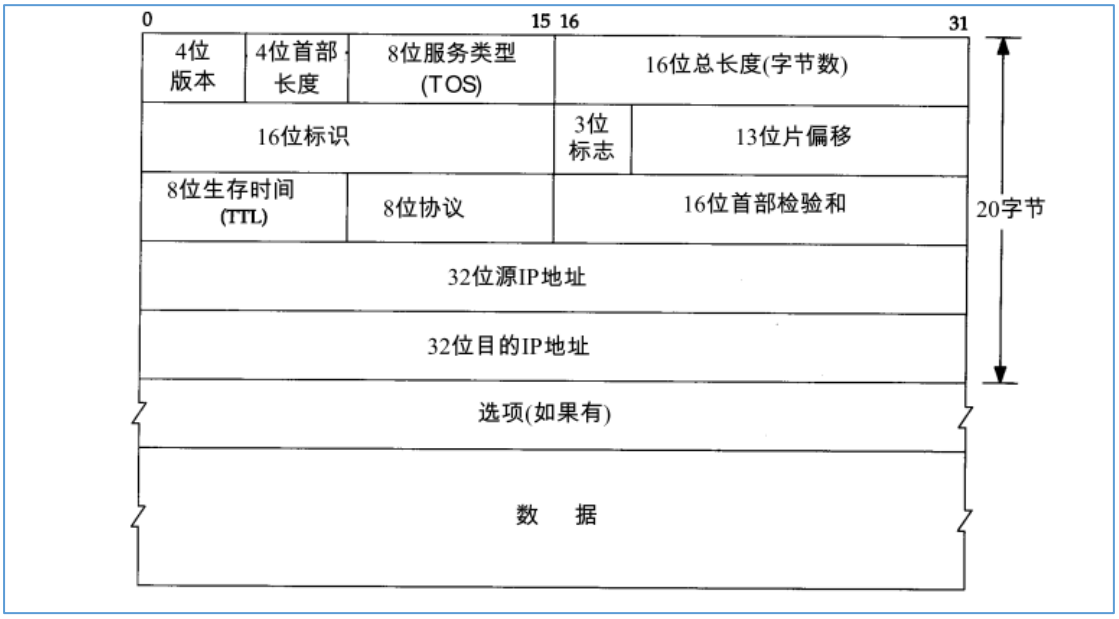
1） 头部固定 20 字节，选项部分最长 40 字节，所以头部最长 60 字节
2） 头部包含了端口号，但不包含 IP 地址
3） SYN、ACK、FIN 等和 TCP 的三次握手、四次挥手关系紧密

1.5 UDP 头部



首部固定 8 字节。

1.6 IP 头部



关键点:

4） 选项部分最多 40 字节，加上固定的 20 字节，最长 60 字节

5） 头部包含了 IP 地址，但没有端口号

6） 16 位总长度，2 字节，所以 IP 数据报最长只能有 65535 字节

7） 13 位片偏移，由于数据链路层有 MTU 的限制，如果 IP 数据包大于 MTU，就需要分片，分片后的需要使用这个偏移进行组合

1.7 以太网帧格式

1.7.1 MTU

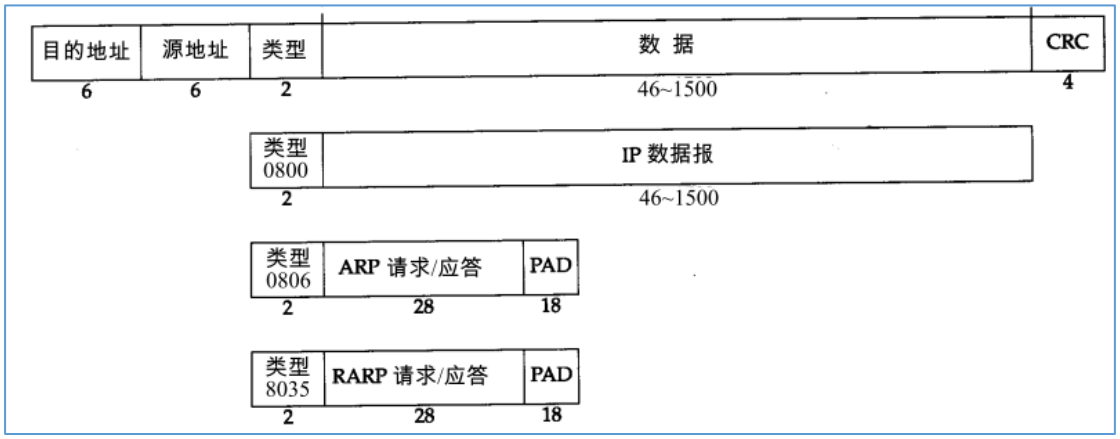
MTU，（Maximum Transmission Unit，最大传输单元），用来通知对方所能接受数据服务单元的最大尺寸。

常用的 MTU 值如下：

1） FDDI 协议：4352 字节
2） 以太网：1500 字节+18 字节（这 18 字节是以太网帧的首部和尾部大小）
3） PPPOE 协议：1492 字节
4） X.25 协议：576 字节
5） P2P 协议：4470 字节
6） PPP 协议：296 字节
7） IEEE 802.3：1492 字节
8） Token Ring（16Mbps）：17914 字节
9） Token Ring（4Mbps）：4464 字节（RFC 1042）
10） IEEE 802.4：8166 字节（RFC 1042）

```
[root@devops-10-12-19-31 ~]# cat /sys/class/net/eth0/mtu
1500
[root@devops-10-12-19-31 ~]#
[root@devops-10-12-19-31 ~]# cat /sys/class/net/docker0/mtu
1500
[root@devops-10-12-19-31 ~]#
[root@devops-10-12-19-31 ~]# cat /sys/class/net/lo/mtu
65536
[root@devops-10-12-19-31 ~]#
```

1.7.2 帧格式



关键点：
数据长：46-1500，为何最小是 46、最大是 1500，可点我查看
目的地址和源地址，都是 MAC 地址。
类型有很多种，图中列出了常见的三种。

1.8 ARP 表

记录着 IP 和 MAC 的对应关系；可以手动维护也可以自动学习。

1.9 FDB 表

FDB（Forwarding Data Base），转发数据库。
记录着：MAC+VLAN 和 PORT 的对应关系；

网络设备都以 MAC 地址来唯一地标识自己，而交换机要实现设备之间的通讯就必须知道自己的哪个端口连接着哪台设备，因此就需要一张 MAC 地址与端口号一一对应的表，以便用于在交换机内部实现二层数据转发，这张二层转发表就是 FDB 表，也叫 MAC 地址表，主要由 MAC 地址、VLAN 号、端口号和一些标志域等信息组成。

动态学习：

交换机在接收到数据帧以后，首先、会记录数据帧中的源 MAC 地址和对应的接口到 MAC 表中，接着、会检查自己的 MAC 表中是否有数据帧中目标 MAC 地址的信息，如果有则会根据 MAC 表中记录的对应接口将数据帧发送出去(也就是单播)，如果没有，则会将该数据帧从非接受接口发送出去(也就是广播)，当有主机回应后，将回应中的相关内容添加到表中。

手动配置：

```
bridge fdb append 00:00:00:00:00:00 dev vxlan100 dst 192.168.33.16
bridge fdb append e2:ca:8e:f0:7d:79 dev vxlan100 dst 192.168.33.16
```

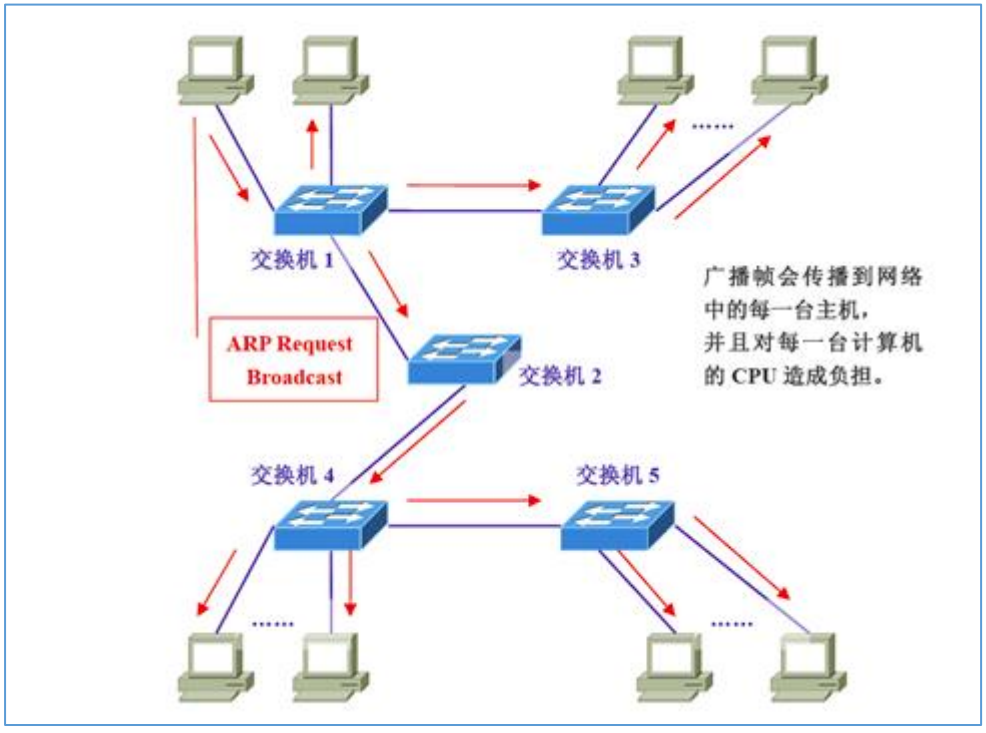
第二章 VLAN

2.1 广播

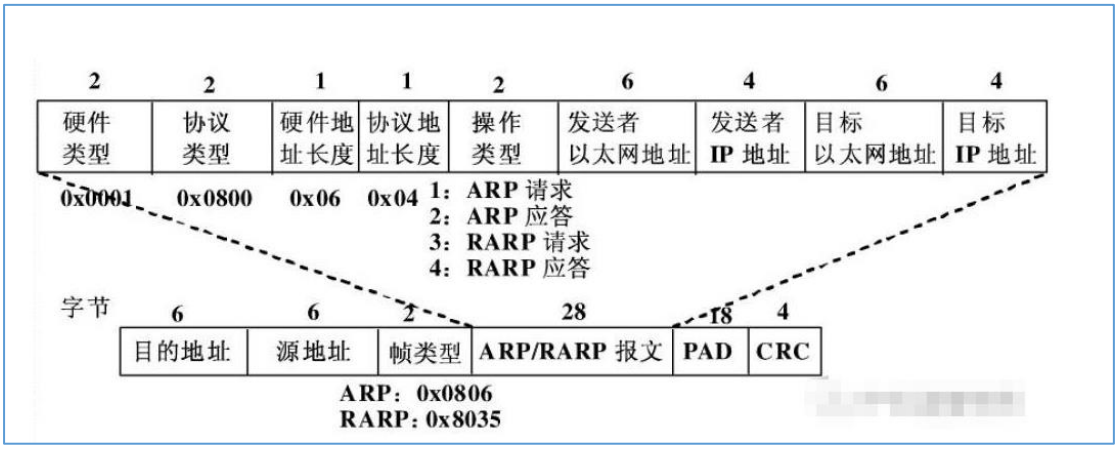
广播域，指的是广播帧（目标 MAC 地址全部为 1）所能传递到的范围，亦即能够直接通信的范围。严格地说，并不仅仅是广 播帧，多播帧（Multicast Frame）和目标不明的单播帧（Unknown Unicast Frame）也能在同一个广播域中畅行无阻。

广播就在我们身边。下面是一些常见的广播通信：

1） ARP 请求：建立 IP 地址和 MAC 地址的映射关系。
2） RIP：选路信息协议(Routing Infromation Protocol)。
3） DHCP：用于自动设定 IP 地址的协议。
4） NetBEUI：Windows 下使用的网络协议。
5） IPX：Novell Netware 使用的网络协议。
6） Apple Talk：苹果公司的 Macintosh 计算机使用的网络协议。



举例：arp、rarp 请求和应答的格式



图中可以看出，arp、rarp 属于 osi 的第二层的报文，也叫以太网报文。
在 TCP/IP 体系中，arp、rarp 等被认为工作在 IP 层，也是可以的。

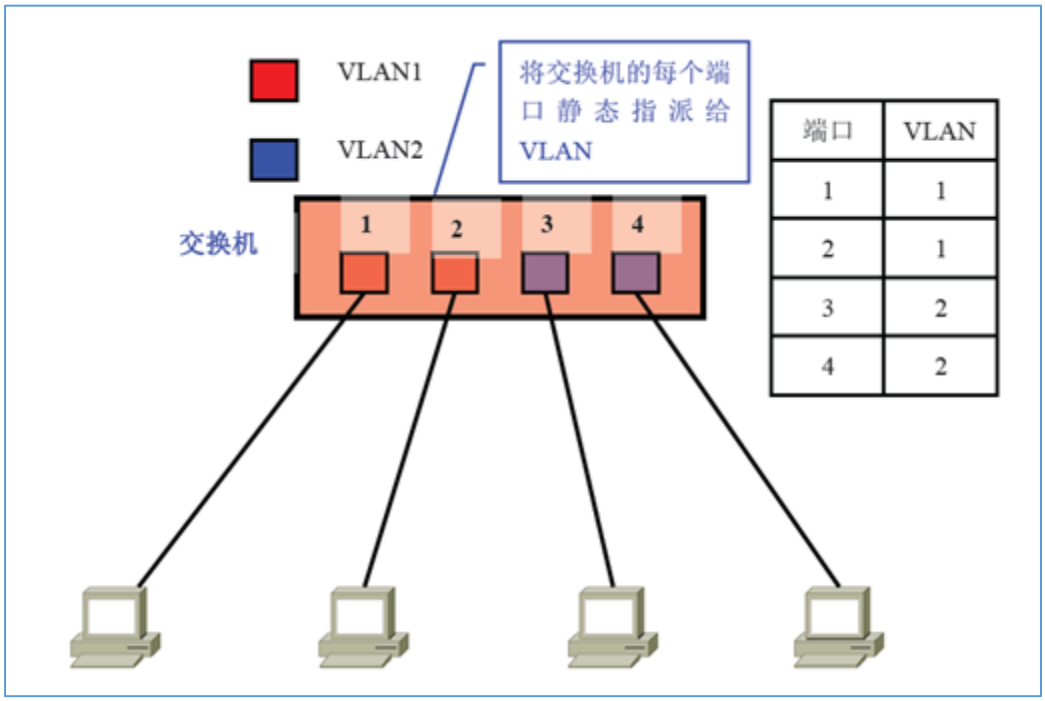
2.2 VLAN

VLAN（Virtual Local Area Network）又称虚拟局域网，是指在交换局域网的基础上，采用网络管理软件构建的可跨越不同网段、不同网络的端到端的逻辑网络。一个 VLAN 组成一个 逻辑子网，即一个逻辑广播域，它可以覆盖多个网络设备，允许处于不同地理位置的网络用户加入到一个逻辑子网中。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，VLAN 之间的通信是通过第 3 层的路由器来完成的。

VLAN 最主要的作用：隔离广播域，减少内部网络流量、减轻服务器处理压力。

2.3 VLAN 划分方法

2.3.1 静态 VLAN



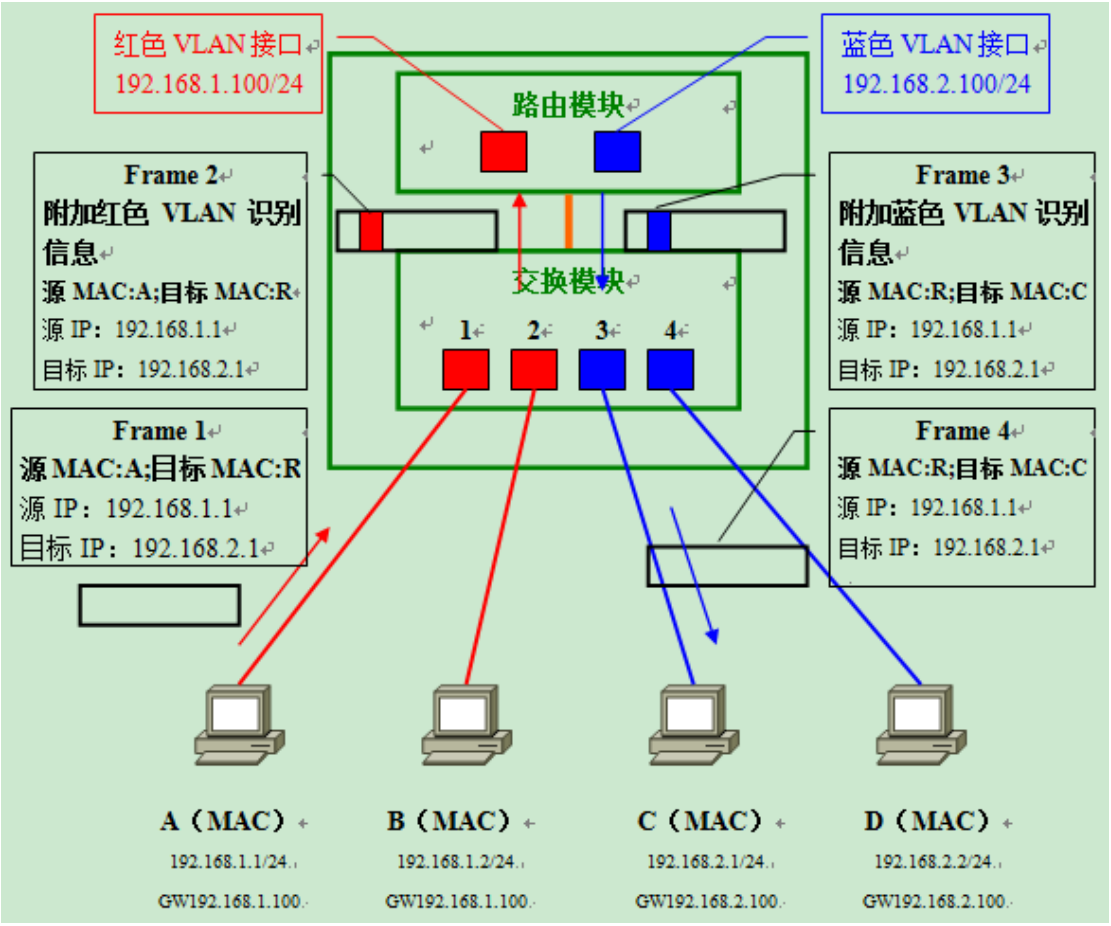
静态 VLAN 又被称为基于端口的 VLAN（Port Based VLAN）。顾名思义，就是明确指定各端口属于哪个 VLAN 的设定方法。比较稳定，且不像动态 VLAN 一样需要交换机芯片额外的工作，用的比较广泛。

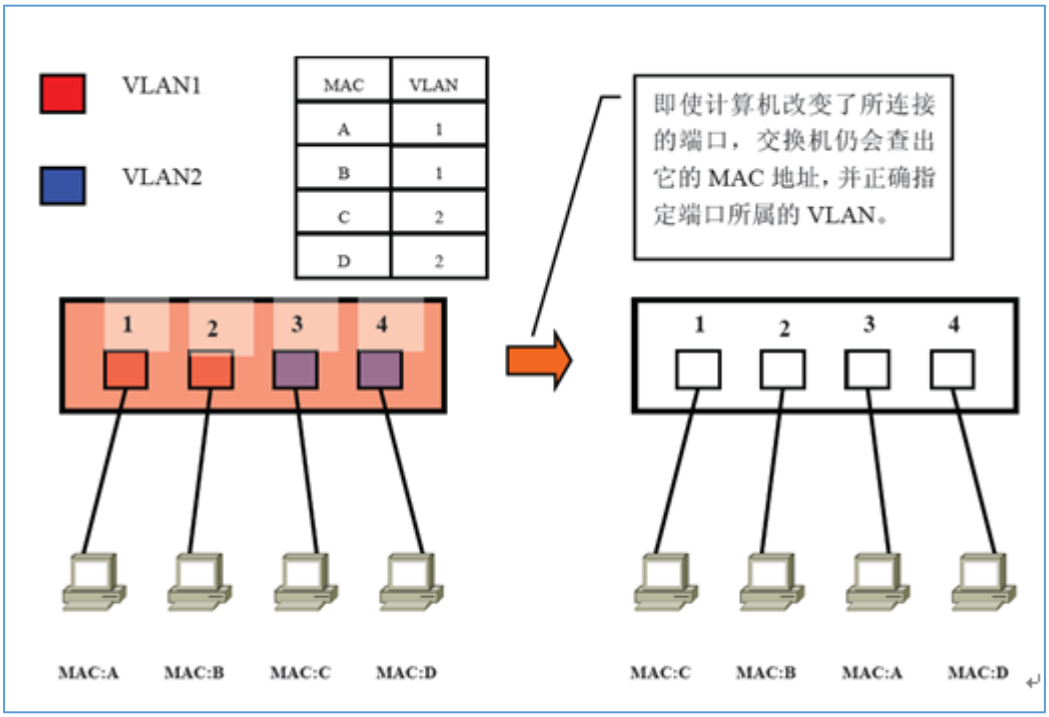
2.3.2 动态 VLAN

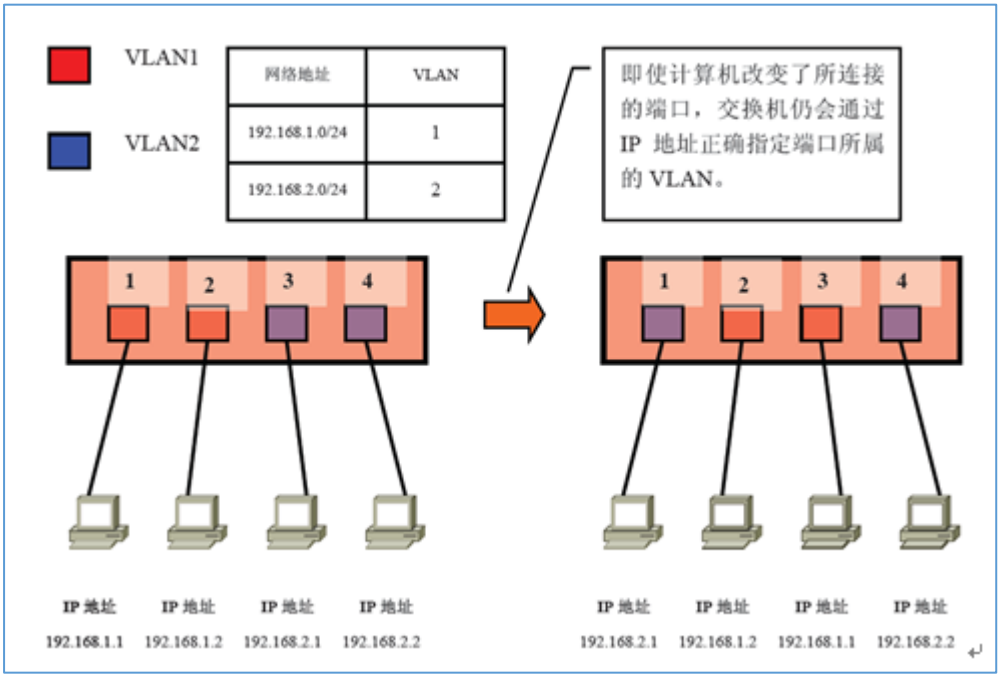
动态 VLAN 可以大致分为 3 类:

- 基于 MAC 地址的 VLAN（MAC Based VLAN）
  - 根据各端口所连计算机的 MAC 地址设定。
  - 工作在 OSI 参考模型的第 2 层
- 基于子网的 VLAN（Subnet Based VLAN）
  - 根据各端口所连计算机的 IP 地址设定
  - 工作在 OSI 参考模型的第 3 层
- 基于用户的 VLAN（User Based VLAN）
  - 根据端口所连计算机上登录用户设定。用到较少。

2.4 VLAN 间的路由


红色 VLAN 接口
192.168.1.100/24
蓝色 VLAN 接口
192.168.2.100/24
路由模块
交换模块
1
2
3
4
Frame 2
附加红色 VLAN 识别信息
源 MAC:A;目标 MAC:R
源 IP：192.168.1.1
目标 IP：192.168.2.1
Frame 3
附加蓝色 VLAN 识别信息
源 MAC:R;目标 MAC:C
源 IP：192.168.1.1
目标 IP：192.168.2.1
Frame 1
源 MAC:A;目标 MAC:R
源 IP：192.168.1.1
目标 IP：192.168.2.1
Frame 4
源 MAC:R;目标 MAC:C
源 IP：192.168.1.1
目标 IP：192.168.2.1
A（MAC）
192.168.1.1/24
GW192.168.1.100
B（MAC）
192.168.1.2/24
GW192.168.1.100
C（MAC）
192.168.2.1/24
GW192.168.2.100
D（MAC）
192.168.2.2/24
GW192.168.2.100


VLAN1
VLAN2
MAC
VLAN
A
1
B
1
C
2
D
2
即使计算机改变了所连接的端口，交换机仍会查出它的 MAC 地址，并正确指定端口所属的 VLAN。
1
2
3
4
MAC:A
MAC:B
MAC:C
MAC:D
MAC:C
MAC:B
MAC:A
MAC:D



2.5 VLAN 帧结构

在交换机的汇聚链接上，可以通过对数据帧附加 VLAN 信息，构建跨越多台交换机的 VLAN。

附加 VLAN 信息的方法，最具有代表性的有下面两种：

- IEEE802.1Q
- ISL

相同点：

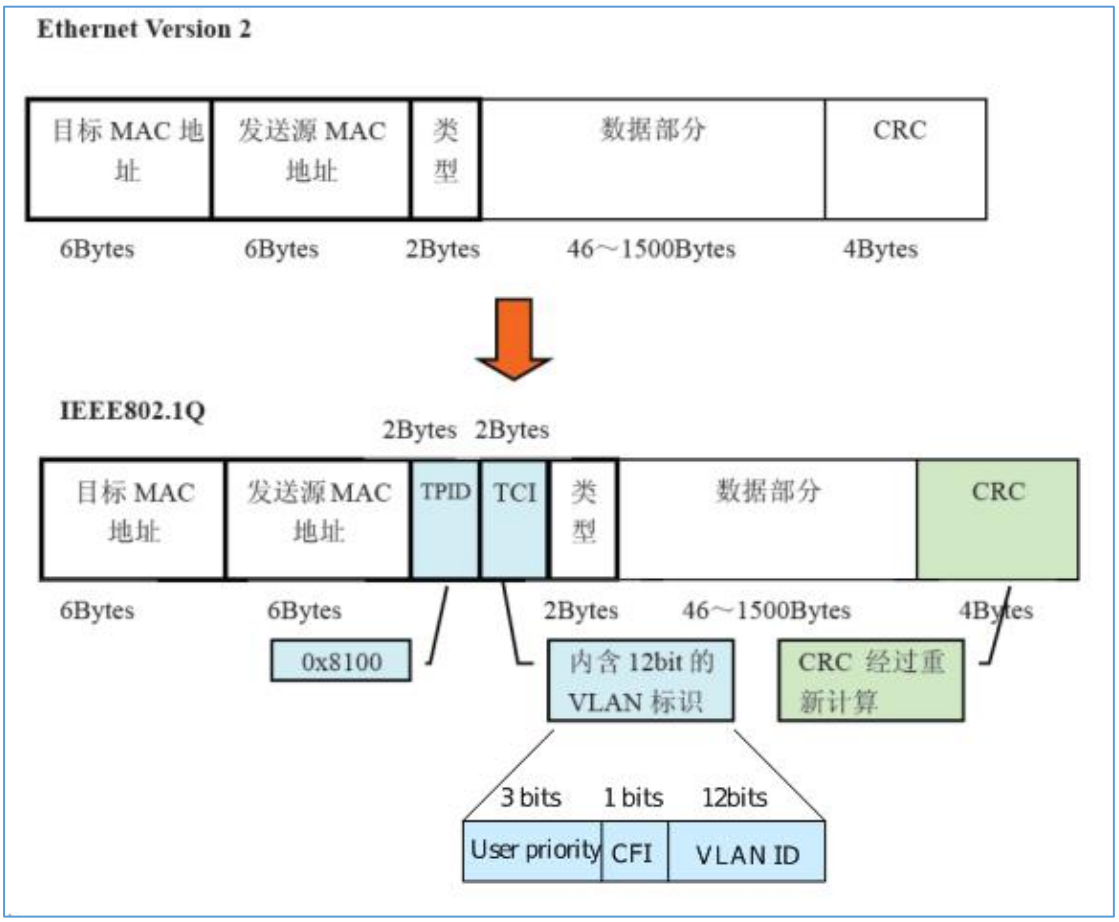
都是显式标记，即帧被显式标记了 VLAN 的信息。

不同点：IEEE 802.1Q 是公有的标记方式，ISL 是 Cisco 私有的，ISL 采用外部标记的方法，802.1Q 采用内部标记的方法，ISL 标记的长度为 30 字节，802.1Q 标记的长度为 4 字节。

有关交换机的汇聚链接，有需要了解的可以搜索下。简单理解就是跨 VLAN 通信时用的。

2.5.1 IEEE802.1Q

IEEE802.1Q，俗称“Dot One Q”，是经过 IEEE 认证的对数据帧附加 VLAN 识别信息的协议。



这里关注下 LAN ID，长度为 12 比特，表示该帧所属的 VLAN。在 VRP 中，可配置的 VLAN ID 取值范围为 1～4094。

三个特殊的 VID：

- 0x000：设置优先级但无 VID
- 0x001：缺省 VID
- 0xFFF：预留 VID

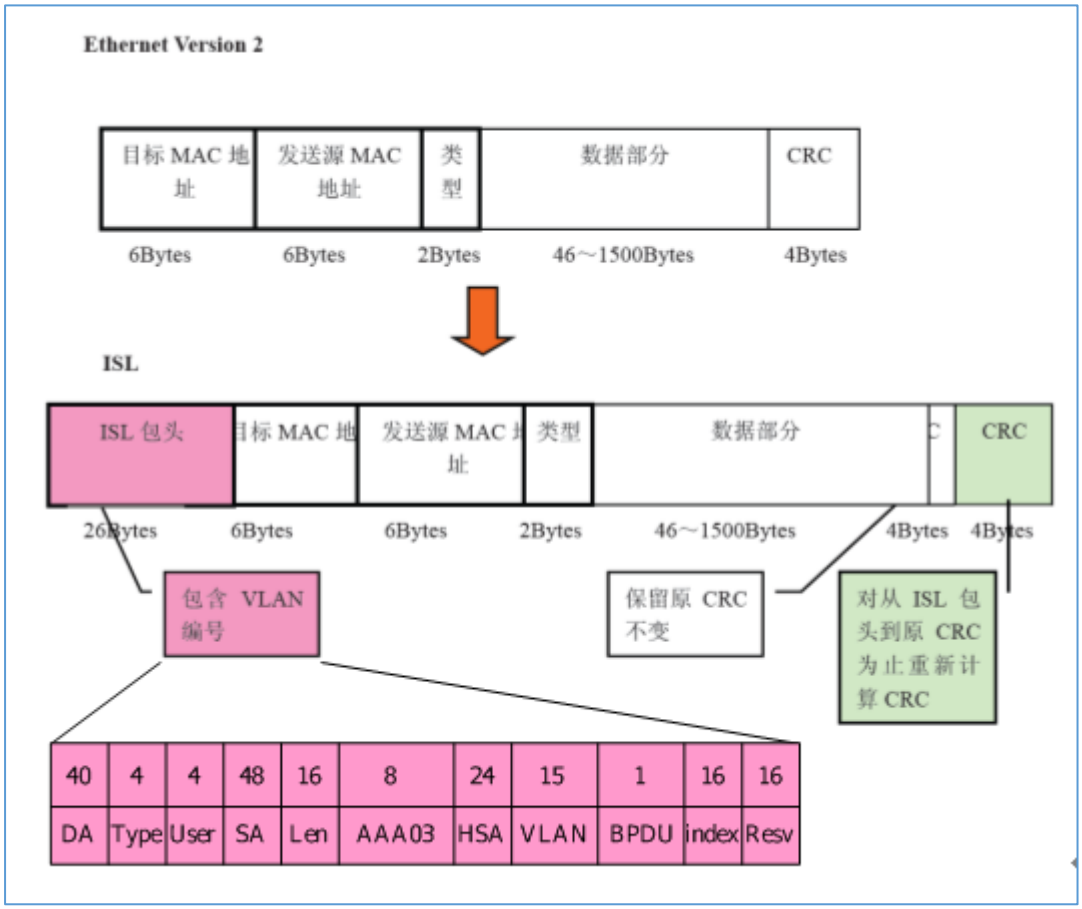
所以最多只能有 4094 个 VLAN

其他字段：https://blog.51cto.com/wushank/1132145

注意这里的 CRC，添加了 VLAN 标识后，会重新计算 CRC，去掉 VLAN 表示后，再次重新计算。

2.5.2 ISL

ISL，是 Cisco 产品支持的一种与 IEEE802.1Q 类似的、用于在汇聚链路上附加 VLAN 信息的协议。



使用 ISL 后，每个数据帧头部都会被附加 26 字节的“ISL 包头（ISL Header）”，并且在帧尾带上通过对包括 ISL 包头在内的整个数据帧进行计算后得到的 4 字节 CRC 值。换而言之，就是总共增加了 30 字节的信息。 在使用 ISL 的环境下，当数据帧离开汇聚链路时，只要简单地去除 ISL 包头和新 CRC 就可以了。由于原先的数据帧及其 CRC 都被完整保留，因此无需重新计算

VLAN — 15 位 VLAN ID。低 10 位用于 1024 VLAN。

2.6 VLAN 工作过程

发送方虚拟机怎么知道对方的 MAC 地址?
vtep 怎么知道目的虚拟机在哪一台宿主机上?

- 内层报文：通信的虚拟机双方要么直接使用 IP 地址，要么通过 DNS 等方式已经获取了对方的 IP 地址，因此网络层地址已经知道。同一个网络的虚拟机需要通信，还需要知道对方虚拟机的 MAC 地址，vxlan 需要一个机制来实现传统网络 ARP 的功能
- vxlan 头部：只需要知道 VNI，这一般是直接配置在 vtep 上的，要么是提前规划写死的，要么是根据内部报文自动生成的，也不需要担心

- UDP 头部：最重要的是源地址和目的地址的端口，源地址端口是系统生成并管理的，目的端口也是写死的，比如 IANA 规定的 4789 端口，这部分也不需要担心
- IP 头部：IP 头部关心的是 vtep 双方的 IP 地址，源地址可以很简单确定，目的地址是虚拟机所在地址宿主机 vtep 的 IP 地址，这个也需要由某种方式来确定
- MAC 头部：如果 vtep 的 IP 地址确定了，MAC 地址可以通过经典的 ARP 方式来获取，毕竟 vtep 网络在同一个三层，经典网络架构那一套就能直接用了

我们带着这些问题，进入下一章学习。

第三章 VXLAN

3.1 产生背景

1) VLAN 资源不足
   a) 传统的二层网络隔离技术 VLAN，因其标识相互隔离的虚拟二层网络的 Tag 域只有 12 比特，仅能划分出 4096 个相互隔离的虚拟二层网络，远远无法满足大二层网络中隔离大量租户的需求。
2) 虚拟机迁移
   a) 为了实现网络业务和资源的灵活调配，虚拟机跨设备甚至跨数据中心的迁移越来越频繁。为了保证虚拟机迁移过程中业务不中断，虚拟机迁移前后的 IP 地址和 MAC 地址需要保持不变， 而传统网络技术无法实现虚拟机迁移前后的 IP、MAC 不变。

3.2 介绍

VXLAN（Virtual eXtensible LAN，可扩展虚拟局域网络）是基于 IP 网络、采用“MAC in UDP”封 装形式的二层 VPN 技术。

一般情况下，多数据中心之间是通过路由连通的，天然是一 个三层网络。而要实现通过三层网络连接的两个二层网络互通，就必须实现“L2 over L3”。

技术特点:

- 支持大量的租户:

使用 24 位的标识符，最多可支持 2 的 24 次方（16777216）个 VXLAN， 支持的租户数目大规模增加，解决了传统二层网络 VLAN 资源不足的问题。

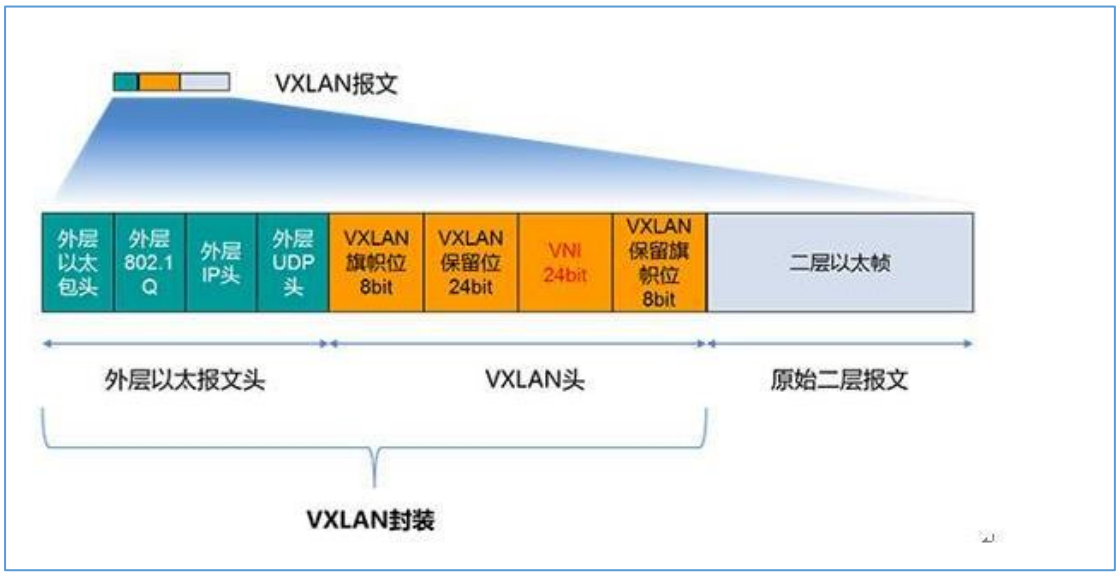
- 虚拟机迁移 IP、MAC 不变:

采用了 MAC in UDP 的封装方式，实现原始二层报文在 IP 网络 中的透明传输，保证虚拟机迁移前后的 IP 和 MAC 不变。

- 易于维护:

基于 IP 网络组建大二层网络，使得网络部署和维护更加容易，并且可以充分地利 用现有的 IP 网络技术，例如利用等价路由进行负载分担等；只有 IP 核心网络的边缘设备需要 进行 VXLAN 处理，网络中间设备只需根据 IP 头转发报文，降低了网络部署的难度和费用。

3.3 VXLAN 报文



VXLAN 头长度为 8 字节，主要包括以下部分：
标记位：“I”位为 1 时，表示 VXLAN 头中的 VXLAN ID 有效；为 0，表示 VXLAN ID 无 效。其他位保留未用，设置为 0。
VXLAN ID：用来标识一个 VXLAN 网络，长度为 24 比特。
Reserved：当前协议保留位
外层 UDP 头：目的端口号固定是 8472

- VXLAN header：vxlan 协议相关的部分，一共 8 个字节

VXLAN flags：标志位，“I”位为 1 时， VXLAN ID 有效；为 0，无 效
Reserved：保留位
VNID：24 位的 VNI 字段，这也是 vxlan 能支持千万租户的地方
Reserved：保留字段

- UDP 头部，8 个字节

UDP 应用通信双方是 vtep 应用，其中目的端口就是接收方 vtep 使用的端口，IANA 分配的端口是 4789

- IP 头部：20 字节

主机之间通信的地址，可能是主机的网卡 IP 地址，也可能是多播 IP 地址

- MAC 头部：14 字节

主机之间通信的 MAC 地址，源 MAC 地址为主机 MAC 地址，目的 MAC 地址为下一跳设备的 MAC 地址

从报文的封装可以看出，VXLAN 头和原始二层数据帧是作为 UDP 报文的载荷存在的。VTEP 之间 的网络设备只需要根据外层以太头和外层 IP 头进行转发、利用源 UDP 端口号进行负载分担。在这 一过程中，VXLAN 报文的处理与普通的 IP 报文完全相同。因此，除了 VTEP 设备，现网的大量设 备无需更换或升级即可支持 VXLAN 网络。

3.4 三个关键角色

- **VTEP**（VXLAN Tunnel EndPoint）：

VTEP（VXLAN Tunnel Endpoints，VXLAN 隧道端点）是 VXLAN 网络的边缘设备，是 VXLAN 隧道的起点和终点，VXLAN 对用户原始数据帧的封装和解封装均在 VTEP 上进行。

VTEP 是 VXLAN 网络中绝对的主角，VTEP 既可以是一台独立的网络设备（比如华为的 CloudEngine 系列交换机），也可以是在服务器中的虚拟交换机。源服务器发出的原始数据帧，在 VTEP 上被封装成 VXLAN 格式的报文，并在 IP 网络中传递到另外一个 VTEP 上，并经过解封转还原出原始的数据帧，最后转发给目的服务器。

- **VXLAN** 网关：

网关除了具备 VTEP 的功能外，还负责 VLAN 报文与 VXLAN 报文之间的映射和转发。VXLAN 的虚拟机与传统 VLAN 的虚拟机之间互访，通过 VXLAN 网关来完成。

flannel 的 vxlan 模式中，可以理解 pod 网络访问宿主机网络，这个角色没有具体实体，是通过一些 iptables 规则实现的。

- **VXLANIP** 网关：

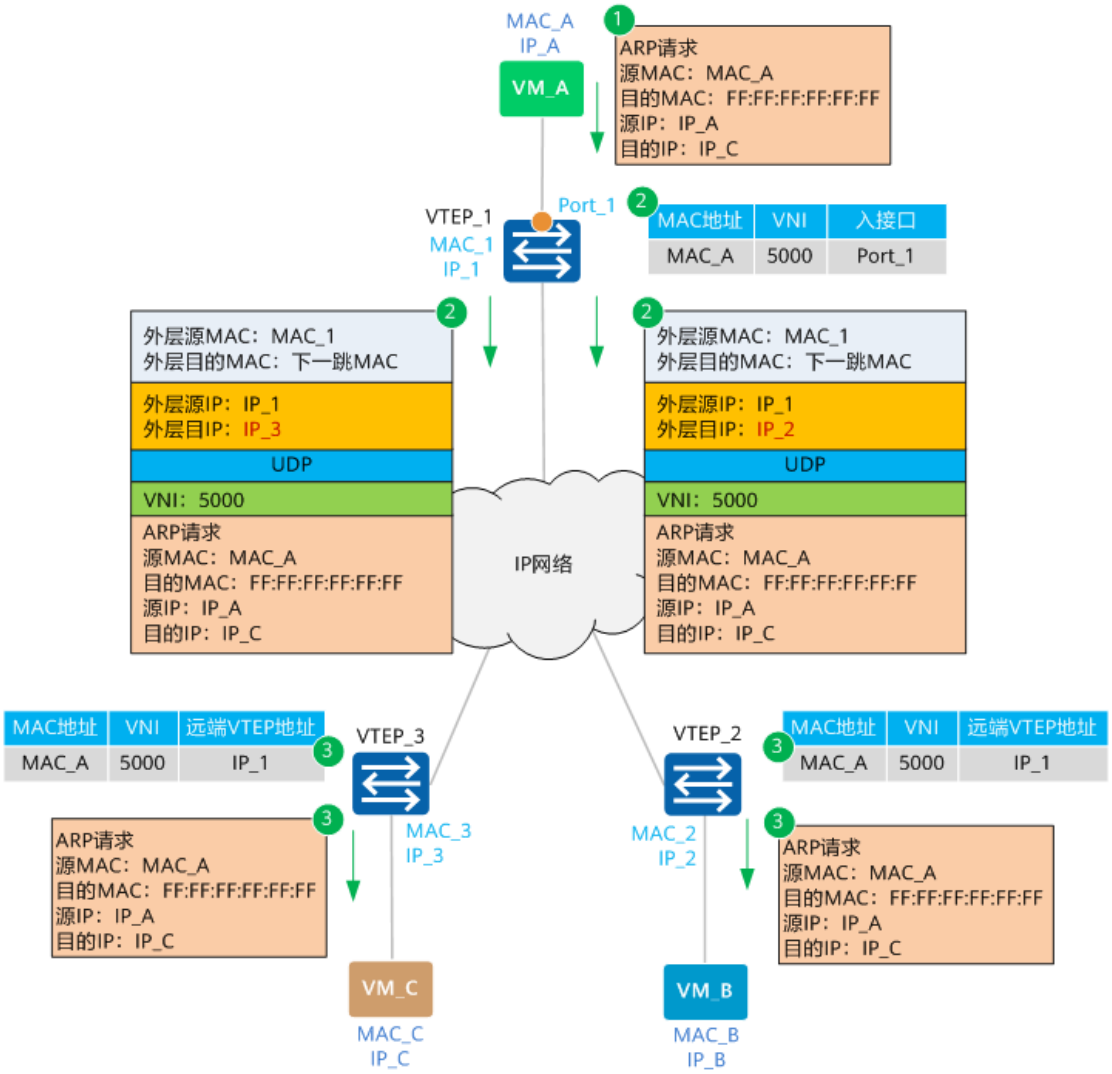
VXLANIP 网关具有 VXLAN 网关的所有功能，此外，还负责处理不同 VXLAN 之间的报文通信。不同 VXLAN 的虚拟机之间需要互访，必须经过 VXLANIP 网关完成。

flannel 的 vxlan 模式中，所有 pod 网络都属于同一个 vxlan，所以没有这个角色。

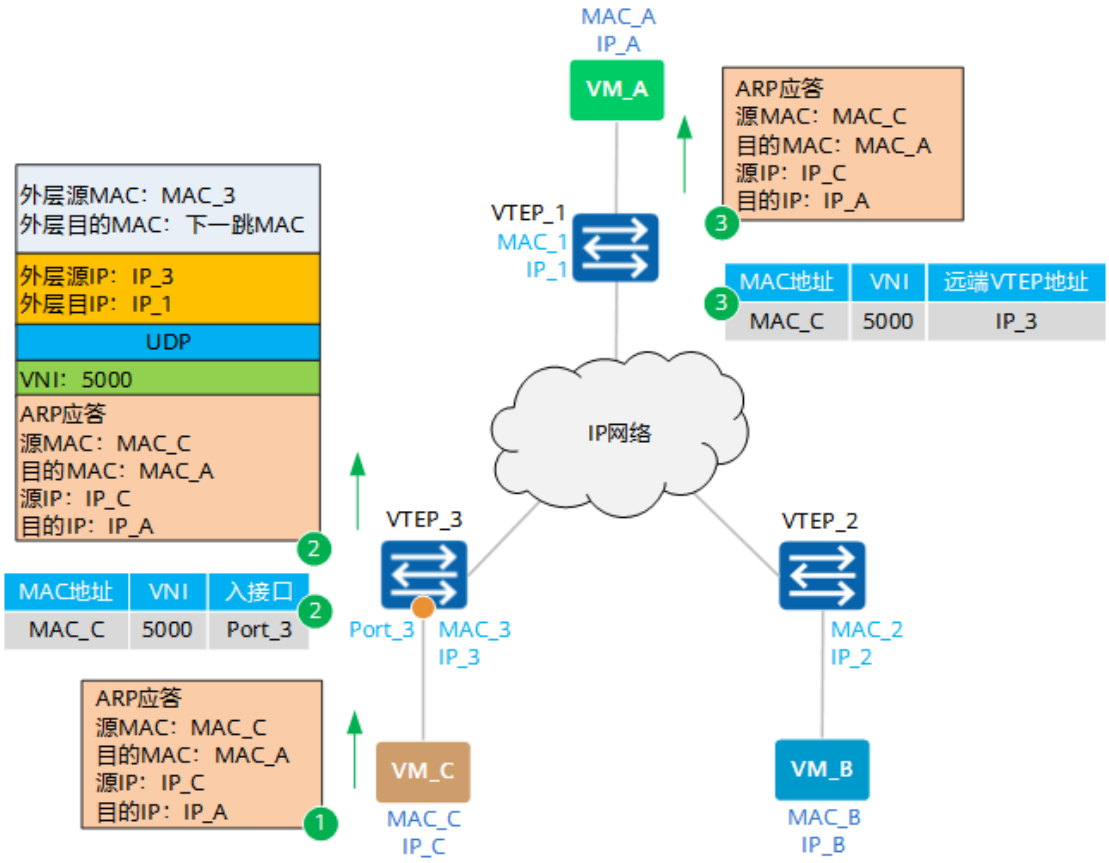
VTEP、VXLAN 网关、VXLAN IP 网关，这三种角色形态可以是虚拟交换机，即通过软件模型实现，也可以是物理交换机。当然也可以部分是虚拟的、部分是物理的。

3.5 VLAN 之 ARP 报文转发流程

ARP 请求报文：


MAC_A
IP_A
VM_A
1
ARP请求
源MAC：MAC_A
目的MAC：FF:FF:FF:FF:FF:FF
源IP：IP_A
目的IP：IP_C
VTEP_1
MAC_1
IP_1
Port_1
2
MAC地址 VNI 入接口
MAC_A 5000 Port_1
2
外层源MAC：MAC_1
外层目的MAC：下一跳MAC
外层源IP：IP_1
外层目IP：IP_3
UDP
VNI：5000
ARP请求
源MAC：MAC_A
目的MAC：FF:FF:FF:FF:FF:FF
源IP：IP_A
目的IP：IP_C
IP网络
2
外层源MAC：MAC_1
外层目的MAC：下一跳MAC
外层源IP：IP_1
外层目IP：IP_2
UDP
VNI：5000
ARP请求
源MAC：MAC_A
目的MAC：FF:FF:FF:FF:FF:FF
源IP：IP_A
目的IP：IP_C
MAC地址 VNI 远端VTEP地址
MAC_A 5000 IP_1
3
VTEP_3
MAC_3
IP_3
3
ARP请求
源MAC：MAC_A
目的MAC：FF:FF:FF:FF:FF:FF
源IP：IP_A
目的IP：IP_C
VM_C
MAC_C
IP_C
VTEP_2
MAC_2
IP_2
MAC地址 VNI 远端VTEP地址
MAC_A 5000 IP_1
3
3
ARP请求
源MAC：MAC_A
目的MAC：FF:FF:FF:FF:FF:FF
源IP：IP_A
目的IP：IP_C
VM_B
MAC_B
IP_B

ARP 响应报文：



第四章 Flannel

SDN（软件定义网络）改变了传统的网络世界规则，它的灵活性和开放性带来了成本的优势。而在容器生态中，Flannel 为容器集群构建 Overlay 网络。Flannel 是在 Underlay 网络之上使用隧道技术依托 UDP 协议层构建的 Overlay 的逻辑网络（MAC in UDP），并能灵活穿透三层 Underlay 网络，使逻辑网络与物理网络解耦，实现灵活的组网需求，不仅仅能适配 VM 虚拟机环境，还能用于 Container 容器环境。

4.1 Flannel 简介

Flannel 是由 go 语言开发，是一种基于 Overlay 网络的跨主机容器网络解决方案，也就是将数据包封装在另一种网络包里面进行路由转发和通信，Flannel 是 CoreOS 开发，专门用于 docker 多主机互联的一个工具，简单来说，它的功能是让集群中的不同节点主机创建的容器都具有全局唯一的虚拟 IP 地址。

k8s 网络组件 flannel 和其他组件比如 calico 一样，主要解决的问题是 k8s 节点之间容器网络的通信，flannel 要保证每个 pod 的 IP 是唯一的，怎么保证是唯一的，大部分组件的做法是在每个 Node 上分配一个唯一的子网，node1 是一个单独的子网，node2 是一个单独的子网，可以理解是不同网段，不同 vlan，所以每个节点都是一个子网。flannel 也是这样做的。

flannel 可以使用 etcd 或者 k8s-apiserver 作为存储。

4.2 Flannel 支持的网络模式

UDP：最早支持的一种方式，由于性能最差，目前已经弃用。
VXLAN：Overlay Network 方案，源数据包封装在另一种网络包里面进行路由转发和通信这也是网络的虚拟化技术，也就是原来是有一个包数据包，有源 IP 和目的 IP，但由于某些情况这个数据包到达不了目的地址上，这可能就会借助物理上的以太网网络进行封装一个数据包带上，然后通过这种物理网络传输到目的地址上，这是一种叠加式的网络，里面是有两种数据包的，这种也叫做隧道方案。
Host-GW：Flannel 通过在各个节点上的 Agent 进程，将容器网络的路由信息刷到主机的路由表上，这样一来所有的主机都有整个容器网络的路由数据了，这样它就知道这个数据包到达这个节点转发到这个机器上，也就是路由表之间转发的，这种也叫路由方案

4.3 集群支持

1）集群启用 cni
kubeadmin 部署:

```
kubeadm 部署指定 Pod 网段，默认支持 cni
kubeadm init --pod-network-cidr=10.244.0.0/16 #要和 flannel 网络对应得上
```

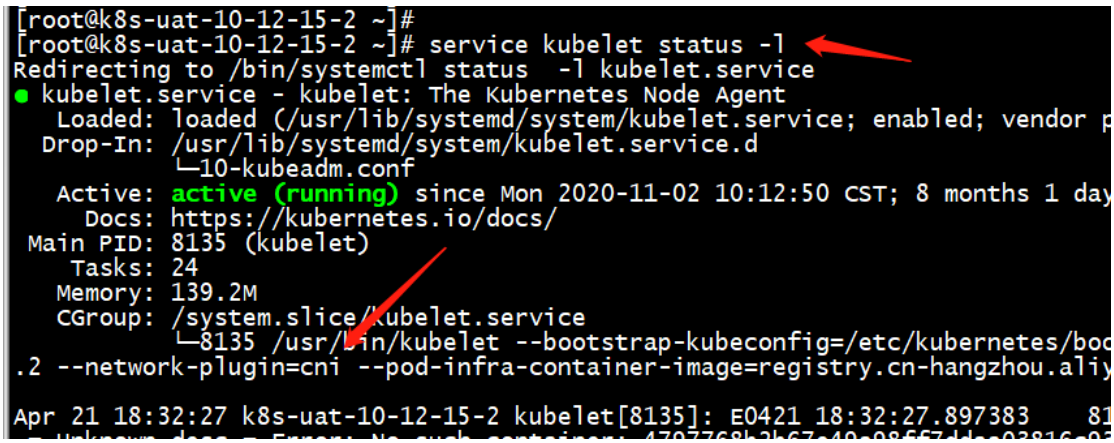
二进制部署:

```
cat /opt/kubernetes/cfg/kube-controller-manager.conf
--allocate-node-cidrs=true \      #允许 node 自动分配 cidr 这个网络
--cluster-cidr=10.244.0.0/16 \    #指定 pod 网络的网段，这个网段要和 flannel 的网段对应上
```

2）kubelet 支持:
cat /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
cat /var/lib/kubelet/kubeadm-flags.env

```
[root@k8s-uat-10-12-15-2 ~]#
[root@k8s-uat-10-12-15-2 ~]# cat /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.c
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=-/etc/sysconfig/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS
[root@k8s-uat-10-12-15-2 ~]#
```

```
[root@k8s-uat-10-12-15-2 ~]#
[root@k8s-uat-10-12-15-2 ~]# cat /var/lib/kubelet/kubeadm-flags.env
KUBELET_KUBEADM_ARGS="--cgroup-driver=cgroupfs --hostname-override=10.12.15.2 --network-plugin=cni --pod-infra-container-i
mage=registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.1"
[root@k8s-uat-10-12-15-2 ~]#
[root@k8s-uat-10-12-15-2 ~]#
```



4.4 Flannel 部署

flannel 以 daemonset 的形式在每个 node 上启动一个 pod（hostnetwork 形式，直接监听）,来启动一个 flannel 的守护进程，主要负责本机路由表的设定和 etcd 中的数据，本地的子网上报到 etcd 中，所以守护进程是非常重要的。

4.4.1 部署用的 yaml 文件

省略了 rbac 部分，内容如下：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: kube-flannel-cfg
  namespace: kube-system
  labels:
    tier: node
    app: flannel
data:
  cni-conf.json: |
    {
      "name": "cbr0",
      "cniVersion": "0.3.1",
      "plugins": [
        {
          "type": "flannel",
          "delegate": {
            "hairpinMode": true,
            "isDefaultGateway": true
          }
```

```
      },
      {
        "type": "portmap",
        "capabilities": {
          "portMappings": true
        }
      }
    ]
  }
  net-conf.json: |
    {
      "Network": "10.244.0.0/16",
      "Backend": {
        "Type": "vxlan"
      }
    }
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: kube-flannel-ds-amd64
  namespace: kube-system
  labels:
    tier: node
    app: flannel
spec:
  selector:
    matchLabels:
      tier: node
      app: flannel
  template:
    metadata:
      labels:
        tier: node
        app: flannel
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: kubernetes.io/os
                    operator: In
                    values:
```

```
                - linux
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
      hostNetwork: true
      tolerations:
      - operator: Exists
        effect: NoSchedule
      serviceAccountName: flannel
      initContainers:
      - name: install-cni
        image: quay.io/coreos/flannel:v0.12.0-amd64
        command:
        - cp
        args:
        - -f
        - /etc/kube-flannel/cni-conf.json
        - /etc/cni/net.d/10-flannel.conflist
        volumeMounts:
        - name: cni
          mountPath: /etc/cni/net.d
        - name: flannel-cfg
          mountPath: /etc/kube-flannel/
      containers:
      - name: kube-flannel
        image: quay.io/coreos/flannel:v0.12.0-amd64
        command:
        - /opt/bin/flanneld
        args:
        - --ip-masq      #代表出公网要走 snat
        - --kube-subnet-mgr  #代表使用 kube 的 subnet-manager，有别于 etcd 的
subnet-manager,该类型基于 k8s 的节点 CIDR
        resources:
          requests:
            cpu: "300m"
            memory: "150Mi"
          limits:
            cpu: "300m"
            memory: "150Mi"
        securityContext:
          privileged: false
          capabilities:
            add: ["NET_ADMIN"]
```

```
      env:
      - name: POD_NAME
        valueFrom:
          fieldRef:
            fieldPath: metadata.name
      - name: POD_NAMESPACE
        valueFrom:
          fieldRef:
            fieldPath: metadata.namespace
      volumeMounts:
      - name: run
        mountPath: /run/flannel
      - name: flannel-cfg
        mountPath: /etc/kube-flannel/
    volumes:
      - name: run
        hostPath:
          path: /run/flannel
      - name: cni
        hostPath:
          path: /etc/cni/net.d
      - name: flannel-cfg
        configMap:
          name: kube-flannel-cfg
```

```
    serviceAccountName: flannel
```

使用了 sa: flannel，使 pod 具有某些特殊权限。

init 容器：

```
      - cp
      args:
      - -f
      - /etc/kube-flannel/cni-conf.json
      - /etc/cni/net.d/10-flannel.conflist
```

挂载了 configmap kue-flannel-cfg，并将 cni-confi.json 拷贝到 cni 目录，cni 目录是 hostPath 挂载的，所以会存在于宿主机上，用于将配置同步到宿主机，该目录路径是固定的，可被 kubelet 识别。当 kubelet 开启了 cni 后会从这个目录查找相关配置。

查下 kubelet 源码如下：

不止是 flannel，任何其它网络插件部署上来也是如此，把配置文件扔到这个目录下被 kubelete 所加载。当 kubelete 需要创建一个 pod，这个 pod 应该有网络和网卡，那么它的网络和网卡怎么生成的呢？kubelet 就调用这个目录下的由配置文件指定的网络插件，由网络插件代为实现地址分配，接口创建，网络创建等各种功能。这就是 CNI，CNI 是一个 JSON 格式的配置文件，另外他有必要有可能还会调 IP 地址管理一些二层的模块来实现一些更为强大的管理功能。

这个过程中，kubelet 是客户端，flannel 是服务端。其实 CSI、CRI 也是类似的逻辑。

业务容器：

挂载了 configmap kue-flannel-cfg 到自己 pod 本地

```
[root@k8s-uat-10-12-15-2 ~]# kubectl exec -it kube-flannel-ds-amd64-rmg6c -n kube-system sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
/ #
/ # ls /etc/kube-flannel/
cni-conf.json  net-conf.json
/ # cat /etc/kube-flannel/*
{
  "name": "cbr0",
  "cniVersion": "0.3.1",
  "plugins": [
    {
      "type": "flannel",
      "delegate": {
        "hairpinMode": true,
        "isDefaultGateway": true
      }
    },
    {
      "type": "portmap",
      "capabilities": {
        "portMappings": true
      }
    }
  ]
}
{
  "Network": "10.244.0.0/16",
  "Backend": {
    "Type": "vxlan"
  }
}
/ #
```

使用 hostPath 形式挂载了/run/flannel 目录，用于生产运行时的相关环境信息

```
[root@k8s-uat-10-12-15-2 ~]# ls /run/flannel/subnet.env
/run/flannel/subnet.env
[root@k8s-uat-10-12-15-2 ~]# cat !$
cat /run/flannel/subnet.env
FLANNEL_NETWORK=10.244.0.0/16
FLANNEL_SUBNET=10.244.0.1/24
FLANNEL_MTU=1450
FLANNEL_IPMASQ=true
```

```
      securityContext:
        privileged: false
        capabilities:
          add: ["NET_ADMIN"]
```

授予业务容器管理宿主机的网络，比如维护路由表、添加删除网络设备等。

启动参数中：

```
--kube-subnet-mgr
```

这个参数会初始化一个 kubernetes client，获取本地 node 的 pod-cidr，这个 pod-cidr 将会作为 flannel 为 node 本地容器规划的 ip 网段。记录到/run/flannel/subnet.env

4.4.2 vxlan 模式

```
kube-flannel.yml
net-conf.json: |
    {
      "SubnetLen": 24, #设置子网长度，默认无
      "SubnetMin": "10.244.10.0", #子网最小值，默认无
      "SubnetMax": "10.244.50.0", #子网最大值，默认无
      "Network": "10.244.0.0/16",
      "Backend": {
        "Type": "vxlan"
      }
    }
```

这种模式由于是走三层的，所以 node 不位于同一个 vlan 也是可以的。由于有封装、解封装的过程，效率可能低一些。

路由表如下类似：

```
[root@k8s-uat-10-12-15-2 ~]# ip route
default via 10.12.0.1 dev eth0 proto static metric 100
```

```
10.12.0.0/16 dev eth0 proto kernel scope link src 10.12.15.2
metric 100
10.244.0.0/24 dev cni0 proto kernel scope link src 10.244.0.1
10.244.1.0/24 via 10.244.1.0 dev flannel.1 onlink
10.244.2.0/24 via 10.244.2.0 dev flannel.1 onlink
10.244.3.0/24 via 10.244.3.0 dev flannel.1 onlink
10.244.4.0/24 via 10.244.4.0 dev flannel.1 onlink
10.244.5.0/24 via 10.244.5.0 dev flannel.1 onlink
10.244.6.0/24 via 10.244.6.0 dev flannel.1 onlink
10.244.7.0/24 via 10.244.7.0 dev flannel.1 onlink
10.244.8.0/24 via 10.244.8.0 dev flannel.1 onlink
10.244.9.0/24 via 10.244.9.0 dev flannel.1 onlink
10.244.10.0/24 via 10.244.10.0 dev flannel.1 onlink
10.244.11.0/24 via 10.244.11.0 dev flannel.1 onlink
10.244.12.0/24 via 10.244.12.0 dev flannel.1 onlink
10.244.13.0/24 via 10.244.13.0 dev flannel.1 onlink
10.244.14.0/24 via 10.244.14.0 dev flannel.1 onlink
10.244.15.0/24 via 10.244.15.0 dev flannel.1 onlink
172.17.0.0/16 dev docker0 proto kernel scope link src
172.17.0.1
```

跨节点 pod 通信过程:
节点 1 的 pod A(容器)->cni0->flannel.1(VTEP 设备)->eth0（节点 1 的 node ip 所在网卡）->internel -> eth1(节点 2 的 node ip 所在网卡)->flannel.1->cni0->pod B(容器)

4.4.3 host-gw 模式

```
kube-flannel.yml

net-conf.json: |
    {
      "Network": "10.244.0.0/16",
      "Backend": {
        "Type": "host-gw"
      }
    }
```

这种模式是把每一个 node 当做自己的网关，**需要所有 node 位于同一个 vlan**。不需要 vxlan 协议，不需要 veth 进行报文的封装、解封装，也就是不需要 flannel.1 这个设备，所以效率最高。

路由表类似于如下:

```
ip route

default via 192.168.31.1 dev ens33 proto static metric 100
```

```
10.244.0.0/24 dev cni0 proto kernel scope link src 10.244.0.1
10.244.1.0/24 via 192.168.31.63 dev ens33
10.244.2.0/24 via 192.168.31.61 dev ens33
192.168.31.0/24 dev ens33 proto kernel scope link src
192.168.31.62 metric 100
```

跨节点 pod 通信过程：

节点 1 的 pod A(容器)->cni0->eth0（节点 1 的 node ip 所在网卡）->internel-> eth1(节点 2 的 node ip 所在网卡)->cni0->pod B(容器)

4.5 部署后观察

4.5.1 查看容器路由表

```
root@base-engine-image-process-cpp-teacher-67d8c47946-lbjrc:/app# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.244.19.1     0.0.0.0         UG    0      0        0 eth0
10.244.0.0      10.244.19.1     255.255.0.0     UG    0      0        0 eth0
10.244.19.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
root@base-engine-image-process-cpp-teacher-67d8c47946-lbjrc:/app#
root@base-engine-image-process-cpp-teacher-67d8c47946-lbjrc:/app#
root@base-engine-image-process-cpp-teacher-67d8c47946-lbjrc:/app# hostname  -I
10.244.19.170
root@base-engine-image-process-cpp-teacher-67d8c47946-lbjrc:/app#
```

容器只有一个 eth0 网口，所有的流量都是从这个网卡出去。

4.5.2 查看宿主机路由表

```
[root@k8s-dev-10-12-12-6 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.12.0.1       0.0.0.0         UG    100    0        0 eth0
10.12.0.0       0.0.0.0         255.255.0.0     U     100    0        0 eth0
10.244.0.0      10.244.0.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.1.0      10.244.1.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.2.0      10.244.2.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.3.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
10.244.4.0      10.244.4.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.5.0      10.244.5.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.6.0      10.244.6.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.7.0      10.244.7.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.8.0      10.244.8.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.9.0      10.244.9.0      255.255.255.0   UG    0      0        0 flannel.1
10.244.10.0     10.244.10.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.11.0     10.244.11.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.12.0     10.244.12.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.13.0     10.244.13.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.14.0     10.244.14.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.15.0     10.244.15.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.16.0     10.244.16.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.19.0     10.244.19.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.20.0     10.244.20.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.21.0     10.244.21.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.22.0     10.244.22.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.23.0     10.244.23.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.24.0     10.244.24.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.25.0     10.244.25.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.26.0     10.244.26.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.27.0     10.244.27.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.28.0     10.244.28.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.29.0     10.244.29.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.30.0     10.244.30.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.31.0     10.244.31.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.32.0     10.244.32.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.33.0     10.244.33.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.34.0     10.244.34.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.35.0     10.244.35.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.36.0     10.244.36.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.37.0     10.244.37.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.38.0     10.244.38.0     255.255.255.0   UG    0      0        0 flannel.1
10.244.40.0     10.244.40.0     255.255.255.0   UG    0      0        0 flannel.1
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
[root@k8s-dev-10-12-12-6 ~]#
```

观察以上路由表:

1) 本机的 pod 子网是 10.244.3.0/24，直接走 cni0;
2) 集群中每个节点都是一个 vxlan 子网，和其中一条路由规则绑定;
3) 宿主机网络走 eth0

4.5.3 查看 cni 网桥

cni0:网桥设备，每创建一个 pod 都会创建一对 veth pair。其中一端是 pod 中的 eth0，另一端是 Cni0 网桥中的端口（网卡）。Pod 中从网卡 eth0 发出的流量都会发送到 Cni0 网桥设备的端口（网卡）上。cni0 设备获得的 ip 地址是该节点分配到的网段的第一个地址。

```
[root@k8s-dev-10-12-12-6 ~]# brctl show cni0
bridge name     bridge id               STP enabled     interfaces
cni0            8000.d65a9511df69       no              veth041bb722
                                                        veth0b7e1ee3
                                                        veth1b885ff4
                                                        veth26b4d926
                                                        veth28b68569
                                                        veth3940c310
                                                        veth3ff86dda
                                                        veth424008f9
                                                        veth4378474b
                                                        veth4964c70a
                                                        veth513cbadb
                                                        veth5ec86afd
                                                        veth6c3c55a0
                                                        veth87451897
                                                        veth8fa388bd
                                                        vethc1943774
                                                        vethd08a4e6a
                                                        vethdc9c4ba5
                                                        vethe36dac2d
                                                        vethebab7f30
                                                        vethf3d41afa
```

interfaces 每一行代表一个接口，可理解为交换机的接口编号，改接口和 pod 一一对应。漆对应关系，如何知道是怎么对应的呢，下面是一种方法

宿主机执行:
ip link 或者 ip a，查看接口的 id
进入容器执行:
ip link show eth0
cat /sys/class/net/eth0/iflink
如果二者是对应的则表示是绑定的。

4.5.4 Flanneld

Flannel 可以使用二进制形式直接运行在宿主机上，也可以以 daemonset 形式运行为 pod 形式。flannel 在每个主机中运行 flanneld 作为 agent，它会为所在主机从集群的网络地址空间中，获取一个小的网段 subnet，本主机内所有容器的 IP 地址都将从中分配。同时 Flanneld 监听 K8s 集群数据库，为 flannel.1 设备提供封装数据时必要的 mac，ip 等网络数据信息。

如果使用 k8s-apiserver 作为存储，flannel 会连接 k8s-api，根据 node.spec.podCIDR 配置本地的 flannel 网络子网。

```
[root@node1 ~]# ps ux|grep flanneld
root     10947  0.1  0.1 1062712 24040 ?       Ssl   2020 610:47 /opt/bin/flanneld --ip-masq --kube-subnet-mgr
root     31930  0.0  0.0 112716   960 pts/0    S+   14:59   0:00 grep --color=auto flanneld
[root@node1 ~]#
[root@node1 ~]#
```

flannel 运行后，会生成本 node 的子网配置:
[root@k8s-dev-10-12-12-6 ~]# cat /var/run/flannel/subnet.env
FLANNEL_NETWORK=10.244.0.0/16
FLANNEL_SUBNET=10.244.3.1/24

FLANNEL_MTU=1450
FLANNEL_IPMASQ=true

4.5.5 查看 flannel.1 的 arp 记录表

Flannel.1 是 overlay 网络的设备，用来进行 vxlan 报文的处理（封包和解包）。不同 node 之间的 pod 数据流量都从 overlay 设备以隧道的形式发送到对端。

每个节点都有一个 flannel.1 扮演 vtep(二层)，二层网络间，每个 vtep 都要之前其他 vtep 的 mac 地址，以便于封装。查看对应的 arp 记录，这些记录是固定的，每加入一个新 node，会更新集群内 所有 node 的 arp 记录表：

```
ip neigh show dev flannel.1
arp -n -i flannel.1
```

```
[root@k8s-dev-10-12-12-6 ~]# ip neigh show dev flannel.1
10.244.22.0 lladdr ae:a9:76:26:92:0c PERMANENT
10.244.35.0 lladdr b2:d4:44:0e:4c:77 PERMANENT
10.244.0.0 lladdr 1e:2f:7f:8f:23:1e PERMANENT
10.244.31.0 lladdr 86:8b:7d:88:d0:be PERMANENT
10.244.40.0 lladdr 96:67:f3:32:08:f8 PERMANENT
10.244.9.0 lladdr aa:2e:c0:4e:b8:64 PERMANENT
10.244.26.0 lladdr b2:31:d3:95:c2:27 PERMANENT
10.244.4.0 lladdr b2:d3:6b:91:65:0c PERMANENT
10.244.34.0 lladdr 8e:6d:4b:54:ce:f6 PERMANENT
10.244.13.0 lladdr f2:7c:e6:8a:3c:12 PERMANENT
10.244.30.0 lladdr 26:ea:ce:b3:77:fd PERMANENT
10.244.8.0 lladdr 5e:17:63:29:c8:96 PERMANENT
10.244.21.0 lladdr be:d2:83:88:43:b8 PERMANENT
10.244.38.0 lladdr c6:3b:c1:c6:88:c6 PERMANENT
10.244.7.0 lladdr 92:1a:1a:79:ae:f2 PERMANENT
10.244.16.0 lladdr 8e:f9:06:74:6b:7f PERMANENT
10.244.2.0 lladdr da:d8:db:8e:b8:83 PERMANENT
10.244.12.0 lladdr 66:99:f6:f9:0a:3b PERMANENT
10.244.25.0 lladdr 06:24:10:fd:9c:55 PERMANENT
10.244.11.0 lladdr 4a:9c:d5:9a:8b:f4 PERMANENT
10.244.20.0 lladdr 7a:13:3a:36:22:1c PERMANENT
10.244.33.0 lladdr 0e:5c:43:bf:f0:22 PERMANENT
10.244.6.0 lladdr 7a:3a:dc:40:0c:96 PERMANENT
```

也可使用如下命令查看：

arp -n -i flannel.1

```
[root@k8s-dev-10-12-12-6 ~]# arp -n -i flannel.1
Address                  HWtype  HWaddress           Flags Mask            Iface
10.244.22.0              ether   ae:a9:76:26:92:0c   CM                    flannel.1
10.244.35.0              ether   b2:d4:44:0e:4c:77   CM                    flannel.1
10.244.0.0               ether   1e:2f:7f:8f:23:1e   CM                    flannel.1
10.244.31.0              ether   86:8b:7d:88:d0:be   CM                    flannel.1
10.244.40.0              ether   96:67:f3:32:08:f8   CM                    flannel.1
10.244.9.0               ether   aa:2e:c0:4e:b8:64   CM                    flannel.1
10.244.26.0              ether   b2:31:d3:95:c2:27   CM                    flannel.1
10.244.4.0               ether   b2:d3:6b:91:65:0c   CM                    flannel.1
10.244.34.0              ether   8e:6d:4b:54:ce:f6   CM                    flannel.1
10.244.13.0              ether   f2:7c:e6:8a:3c:12   CM                    flannel.1
10.244.30.0              ether   26:ea:ce:b3:77:fd   CM                    flannel.1
10.244.8.0               ether   5e:17:63:29:c8:96   CM                    flannel.1
10.244.21.0              ether   be:d2:83:88:43:b8   CM                    flannel.1
10.244.38.0              ether   c6:3b:c1:c6:88:c6   CM                    flannel.1
10.244.7.0               ether   92:1a:1a:79:ae:f2   CM                    flannel.1
10.244.16.0              ether   8e:f9:06:74:6b:7f   CM                    flannel.1
10.244.2.0               ether   da:d8:db:8e:b8:83   CM                    flannel.1
10.244.12.0              ether   66:99:f6:f9:0a:3b   CM                    flannel.1
10.244.25.0              ether   06:24:10:fd:9c:55   CM                    flannel.1
10.244.11.0              ether   4a:9c:d5:9a:8b:f4   CM                    flannel.1
10.244.20.0              ether   7a:13:3a:36:22:1c   CM                    flannel.1
10.244.33.0              ether   0e:5c:43:bf:f0:22   CM                    flannel.1
10.244.6.0               ether   7a:3a:dc:40:0c:96   CM                    flannel.1
10.244.19.0              ether   82:1a:45:33:bc:e0   CM                    flannel.1
10.244.29.0              ether   96:35:4e:ae:d3:8a   CM                    flannel.1
10.244.15.0              ether   6a:34:58:c0:c7:09   CM                    flannel.1
10.244.24.0              ether   42:65:22:1c:df:d3   CM                    flannel.1
10.244.37.0              ether   8a:c5:05:ee:c2:8b   CM                    flannel.1
10.244.10.0              ether   02:c5:91:37:d1:69   CM                    flannel.1
10.244.23.0              ether   6a:89:82:06:13:bc   CM                    flannel.1
10.244.32.0              ether   ba:fa:3f:9e:46:19   CM                    flannel.1
10.244.1.0               ether   7e:29:11:9d:a9:c6   CM                    flannel.1
10.244.28.0              ether   0a:f7:50:ab:7d:4c   CM                    flannel.1
10.244.14.0              ether   aa:c0:7d:2a:ec:98   CM                    flannel.1
10.244.27.0              ether   72:fb:1c:b6:a5:a7   CM                    flannel.1
10.244.36.0              ether   ee:39:12:8c:21:fd   CM                    flannel.1
10.244.5.0               ether   a2:a6:7b:87:36:fd   CM                    flannel.1
```

命令说明具体如下。

Address：主机地址。

Hwtype：硬件类型。

Hwaddress：硬件地址。

Flags Mask：记录标志，“C”表示 arp 高速缓存中的条目，“M”表示静态的 arp 条目。

Iface：网络接口。

PERMANENT： 永久的，不会自动学习和更新

比如可以手动帮定：

```
arp -s 10.0.0.99 00:0c:29:c0:5a:ef  #绑定 IP 地是和 MAC 地址
```

4.5.6 查看 flannel.1 的 FDB 表

bridge fdb show dev flannel.1

```
[root@k8s-dev-10-12-12-6 ~]# bridge fdb show  dev flannel.1
46:b0:db:48:05:25 dst 10.12.15.36 self permanent
6a:34:58:c0:c7:09 dst 10.12.15.35 self permanent
d6:31:eb:c3:3a:19 dst 10.12.19.2 self permanent
16:ba:26:16:b1:63 dst 10.12.15.20 self permanent
92:1a:1a:79:ae:f2 dst 10.12.15.20 self permanent
5e:a2:5a:c9:29:36 dst 10.12.19.1 self permanent
3e:4b:2d:ce:91:4b dst 10.12.15.19 self permanent
82:1a:45:33:bc:e0 dst 192.168.68.4 self permanent
32:35:db:4f:a2:5b dst 192.168.68.66 self permanent
0e:5c:43:bf:f0:22 dst 10.12.19.24 self permanent
aa:2e:c0:4e:b8:64 dst 10.12.15.22 self permanent
96:67:f3:32:08:f8 dst 10.12.14.49 self permanent
ca:ed:5d:8c:5f:5e dst 192.168.3.4 self permanent
c6:a5:29:13:36:7a dst 192.168.68.68 self permanent
26:ea:ce:b3:77:fd dst 10.12.19.11 self permanent
d6:68:02:56:f8:87 dst 10.12.12.2 self permanent
7a:8f:43:95:9e:8a dst 192.168.3.4 self permanent
b6:d1:ee:ac:92:ea dst 10.12.19.5 self permanent
02:c4:4e:b1:ce:02 dst 192.168.3.4 self permanent
b2:d4:44:0e:4c:77 dst 10.12.19.27 self permanent
ae:d4:45:e8:4f:e3 dst 10.12.15.19 self permanent
7a:13:3a:36:22:1c dst 10.12.19.1 self permanent
```

PERMANENT： 永久的，不会自动学习和更新

4.5.7 查看 vtep 配置

ip -d link show flannel.1

```
[root@k8s-dev-10-12-12-6 ~]# ip -d link show flannel.1
6: flannel.1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN mode DEFAULT group defau
    link/ether c6:7a:d4:0e:e2:5e brd ff:ff:ff:ff:ff:ff promiscuity 0
    vxlan id 1 local 10.12.12.6 dev eth0 srcport 0 0 dstport 8472 nolearning ageing 300 noudpcsum noudp6zeroc
[root@k8s-dev-10-12-12-6 ~]#
[root@k8s-dev-10-12-12-6 ~]#
```

- VNI 标识： vid:1
- 关闭自学习，不自动学习其他 VTEP 的 mac： nolearning
- 通过 vxlan 隧道发送出去使用的本地 ip： local:10.12.12.6，vxlan 封装用的
- 隧道使用的端口： port:8472

```
[root@k8s-dev-10-12-12-6 ~]# netstat -ulnp | grep 8472
udp        0      0 0.0.0.0:8472            0.0.0.0:*                           -
[root@k8s-dev-10-12-12-6 ~]#
```

最后那个横线，表示是内核态监听，不归属于用户态程序。

4.6 流量分析

K8s 网络需要解决以下问题：

4.6.1 集群外

1) 外部应用与服务之间的通信

可通过 nodePort、ingress、loadbalance 等方式，不是本文档的重点，不再介绍。

4.6.2 集群内

1) Pod 和服务之间的通信

即 pod IP 与 cluster IP 之间直接通信，他们其实不在同一个网段，但是他们通过我们本地的 IPVS 或者 iptables 规则能实现通信，而且我们知道 1.11 上的 kube-proxy 也支持 IPVS 类型的 service，只不过我们以前没有激活过。即 pod IP 与 cluster IP 通信是通过系统上已有的 iptables 或 ipvs 规则来实现的，这里特别提醒一下 ipvs 取代不了 iptables，因为 ipvs 只能拿来做负载均衡，做 nat 转换这个功能就做不到。

参考:《Kubernetes-核心资源之 Service》，不是本文重点，不再介绍。

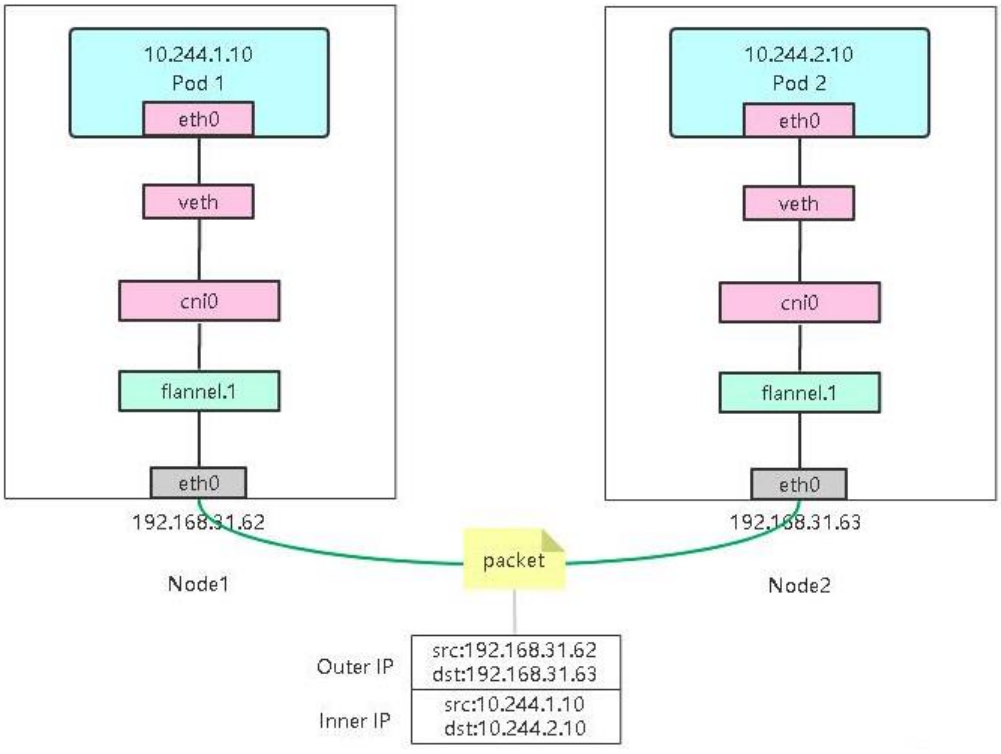
2) 容器与容器之间的通信

Kubernetes 创建 Pod 时，首先会创建一个 pause 容器，为 Pod 指派一个唯一的 IP 地址。然后，以 pause 的网络命名空间为基础创建容器，各容器共享这个网络名称空间，所以使用 localhost 即可通信。

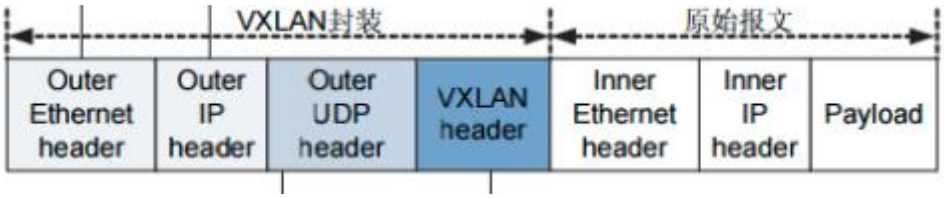
3) Pod 和 Pod 之间的通信

本地流量：比如宿主机到本机的 pod、本机的多个 pod 直接通信，直接通过 cni0 即可，不需要 vxlan 封装。这和非 k8s 环境的多个 docker 容器内部通信使用 docker0 作为网桥通信原理类似，不需要经过 vxlan 的封包和解包。

外部流量：比如节点 1 的 pod 要向节点 2 的 pod 通信，效果如下



flnnel.1 扮演了 vxlan 中 vtep 的作用。



宿主机抓包:

其中一个 pod 夸主机 ping 另一个 pod，在宿主机抓包查看

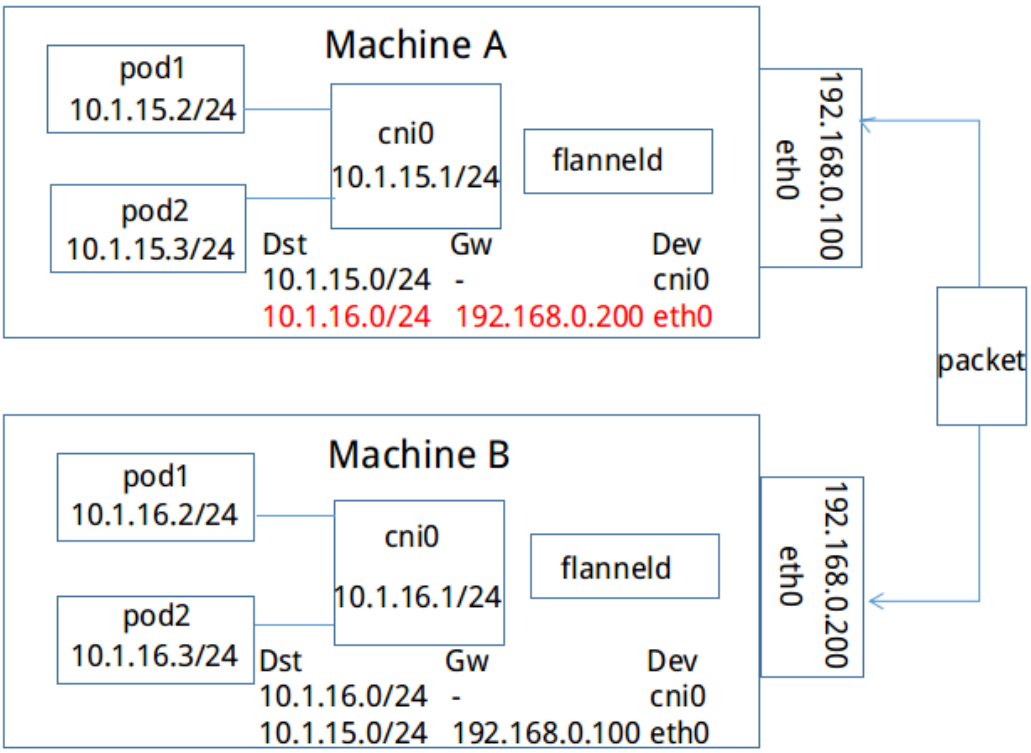
tcpdump -i eth0 dst 10.12.19.12 -vvvvvvv and udp -c 20

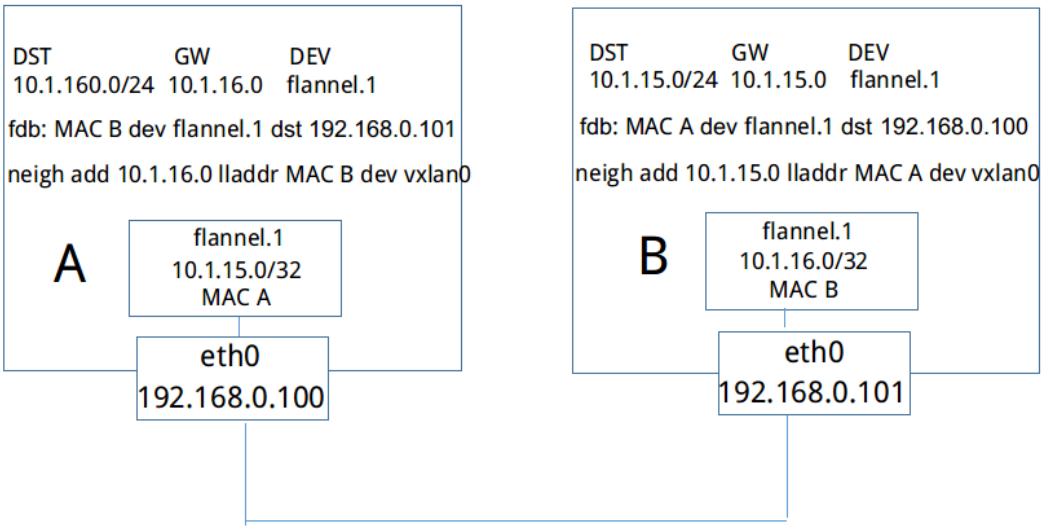
```
IP (tos 0x0, ttl 63, id 63116, offset 0, flags [DF], proto ICMP (1), length 84)
    10.244.28.205 > 10.244.31.190: ICMP echo request, id 29, seq 75, length 64
17:49:31.909268 IP (tos 0x0, ttl 64, id 55740, offset 0, flags [none], proto UDP (17), length 134)
    10.12.19.10.38338 > 10.12.19.12.otv: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 1
IP (tos 0x0, ttl 63, id 63672, offset 0, flags [DF], proto ICMP (1), length 84)
    10.244.28.205 > 10.244.31.190: ICMP echo request, id 29, seq 76, length 64
    10.12.19.10.49313 > 10.12.19.12.otv: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 1
IP (tos 0x0, ttl 63, id 23366, offset 0, flags [DF], proto TCP (6), length 52)
```

4.6.3 不同宿主机 pod 直接通信详解

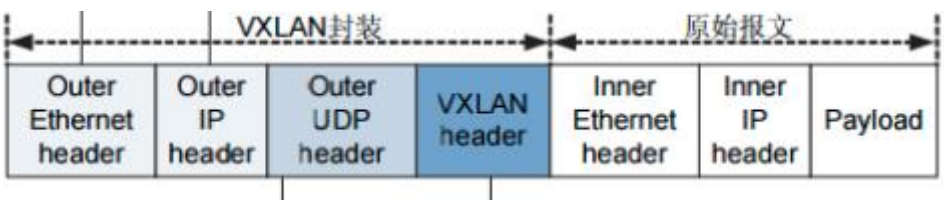
每当有新节点接入集群时（以 B 为例），其他任何 node 节点都会更新以下三部分信息:
1) **路由信息**：所有通往目的地址 10.1.16.0/24 的封包都通过 vtep 设备 flannel.1 设备发出，发往的网关地址为 10.1.16.0，即主机 B 中的 flannel.1 设备。(route -n 看下)
2) **fdb 信息**：MAC 地址为 MAC B 的封包，都将通过 vxlan 首先发往目的地址 192.168.0.101，即主机 B（bridge fdb show dev flannel.1 看下）
3）**arp 信息**：网关地址 10.1.16.0 的地址为 MAC B

如下示例：10.15.2 向 10.1.16.2 通信





到达 flannel.1-A 时，会进行报文封装：



加上主机 A 中有个一个 pod-A 发包到主机 B 的某个 pod-B:

原始报文：

- playload：有效载荷，比如 TCP 或者 UDP 包，其中会包含端口号，这样到到目的主机才能识别出是哪个程序
- InnerIP 头部：src IP 是 podIP，dst IP 是目的的 podIP
- Inner Ethernet 头部：以太网帧头部，源 mac 是 flannel.1-A 的 mac，目的 mac 是目的 flannel.1-B 的 MAC。可将 cni0 理解为网关，flannel.1 理解为路由器，且使用网线直连的，各个路由器也是使用网线直连的。数据包出网关 cni0 会修改 mac 头部，出路由器也会修改 mac 头部。所以这里理解为两个路由器通信，所以 mac 是 flannel.1 的。

→以上组成原始报文，会到达 cni0 网桥
→经过查询路由表，应该到 flnnel.1
→falnnel.1 会进行 vxlan 封装，内核完成

vxlan 封装：

- vxlan 头部：包含了 vxlan 的 id，固定是 1
- Outer UDP 头部：源端口随意生成，目的端口固定为 8472
- Outer IP 头部：源 ip 是 192.168.0.100（已经绑定，见 4.5.7），目的 ip 是 192.168.0.101，目的 ip 是查看了 fdb 表获取的。
- Outer Ethernet 头部：源 mac 是 flannel.1-A 的 mac，目的 mac 是 flannel.1-B 的 mac，需要注意的是，这是出 flannel.1 封装的，当初 A 的 eth0 时，会重置此 mac 头，当经过各种网络设备时也会再次充值此 mac 头，这和普通的三层网络流量是一致的规则。

此次封装完毕，源 ip 和目的 ip 都是宿主机的 ip，通过宿主机的 eth0 出去，经过三层网络传输，可以跨三层设备。当目标主机 B 收到报文后，拆解 mac 头部，发现目的 ip 是本机，不会丢球。拆解 udp 头部，发现是发往 UDP8472 端口的，将数据发往 falnneld 处理，flanneld 分析 xvlan 头部，发现是属于 xvlan id 是归属于自己的，拆解 mac 头部，拆解 ip 头部，查看路由后，发往 cni0 处理，cni0 也会有 mac 头重新封装的过程，终止 cni0 相当于本机 pod 网络的一个接入交换机，然后流量就到了 pod。

以下是另一个图帮助理解: