Kubernetes-Flannel

1. Flannel网络

Flannel通过在每一个节点上启动一个叫做flanneld的进程，负责每一个节点上的子网划分，并将相关的配置信息（如各个节点的子网网段、外部IP等）保存到etcd中，而具体的网络包转发交给具体的backend实现。

flanneld可以在启动时通过配置文件指定不同的backend进行网络通信，Flannel支持三种backend：UDP、VXLAN、host-gw、ip-ip。目前VXLAN是官方最推崇的一种backend实现方式；host-gw一般用于对网络性能要求比较高的场景，但需要基础网络架构的支持。

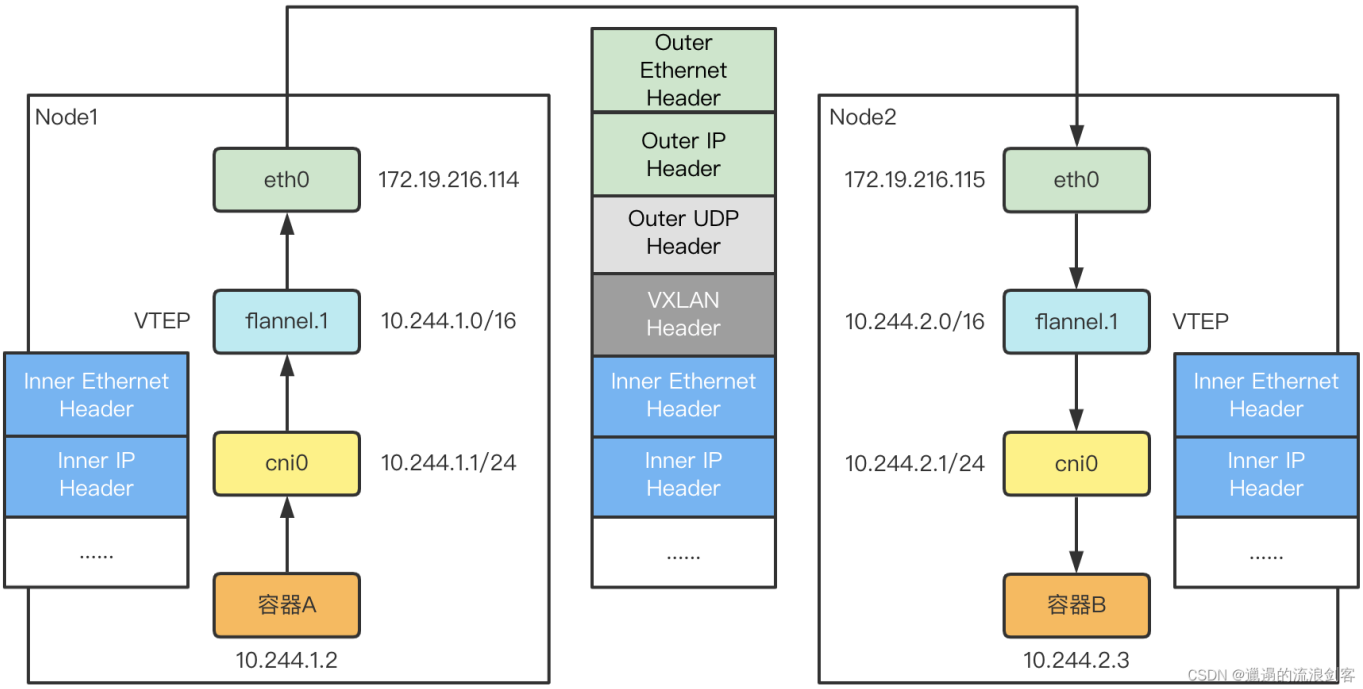
1.1 Backend-VXLAN模式

1.1.1 简介

VXLAN是Linux内核本身就支持的一种网络虚拟化技术，所以说，VXLAN可以完全在内核态实现上述封装和解封装的工作，从而通过与前面相似的隧道机制，构建出覆盖网络。

VXLAN的覆盖网络的设计思想是：在现有的三层网络之上，覆盖一层虚拟的、由内核VXLAN模块负责维护的二层网络，使得连接在这个VXLAN二层网络上的主机之间可以像在同一个局域网（LAN）里那样自由通信。为了能够在二层网络上打通隧道，VXLAN会在宿主机上设置一个特殊的网络设备作为隧道的两端，这个设备叫作VTEP。VTEP的作用其实跟前面的flanneld进程非常相似。只不过，它进行封装和解封装的对象是二层数据帧；而且这个工作的执行流程，全部是在内核里完成的（因为VXLAN本身就是Linux内核里的一个模块）。

1.1.2详解



前置条件：

宿主机Node1（IP地址是172.19.216.114）上的，容器A的IP地址是10.244.1.2，要访问宿主机Node2（IP地址是172.19.216.115）上的，容器B的IP地址是10.244.2.3

容器A访问容器B的整个通信过程分析

当容器A发出请求之后，这个目的地址是10.244.2.3的IP包，通过容器的网关进入cni0网桥，然后被路由到本机flannel.1设备进行处理。也就是说，来到了隧道的入口。为了方便叙述，接下来会把这个IP包称为**原始IP包**

当Node2启动并加入Flannel网络之后，在Node1以及所有其他节点上，flanneld就会添加一条如下所示的路由规则：

```shell
[root@k8s-node1 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
...
10.244.2.0      10.244.2.0      255.255.255.0   UG    0      0        0 flannel.1

```

这条规则的意思是：凡是发往 `10.244.2.0/24` 网段的 `IP` 包，都需要经过 `flannel.1` 设备发出，并且，它最后被发往的网关地址是： `10.244.2.0` ，也就是Node2上的VTEP设备（也就是 `flannel.1` 设备）的 `IP` 地址。为了方便叙述，接下来会把 `Node1` 和 `Node2` 上的 `flannel.1` 设备分别称为源 `VTEP` 设备和目的 `VTEP` 设备。

这些VTEP设备之间，就需要想办法组成一个虚拟的二层网络，即：通过二层数据帧进行通信。所以在我们的例子中，源VTEP设备收到原始IP包后，就要想办法把原始IP包加上一个目的MAC地址，封装成一个二层数据帧，然后发送给目的VTEP设备。

这里需要解决的问题就是：目的VTEP设备的MAC地址是什么？此时，根据前面的路由记录，我们已经知道了目的VTEP设备的IP地址。而要根据**三层IP地址查询对应的二层MAC地址，即ARP表的功能。**

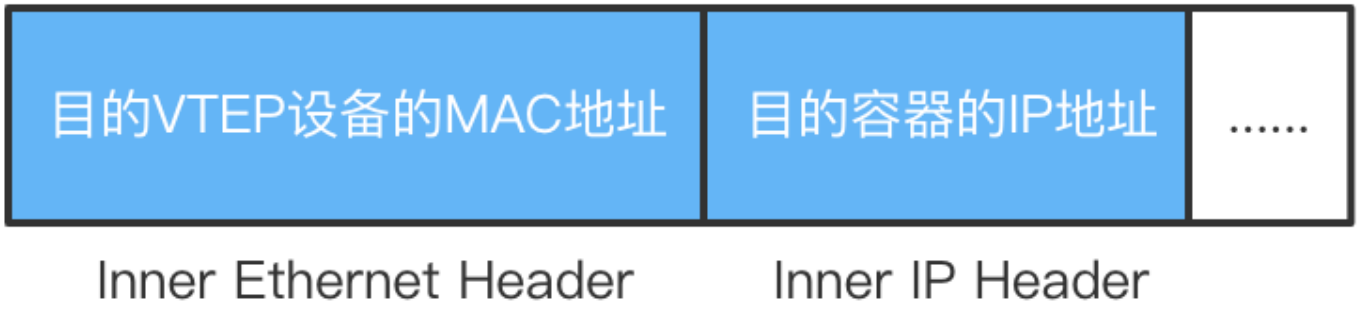
而这里要用到的ARP记录，也是flanneld进程在Node2节点启动时，自动添加在Node1上的

```shell
[root@k8s-node1 ~]# ip neigh show dev flannel.1
10.244.2.0 lladdr 82:9e:ca:29:46:96 PERMANENT
10.244.0.0 lladdr 16:76:50:b4:c3:a5 PERMANENT
```

IP地址10.244.2.0对应的MAC地址是82:9e:ca:29:46:96

> 最新版本的Flannel并不依赖L3 MISS事件和ARP学习，而会在每台节点启动时把它的VTEP设备对应的ARP记录，直接下放到其他每台宿主机上

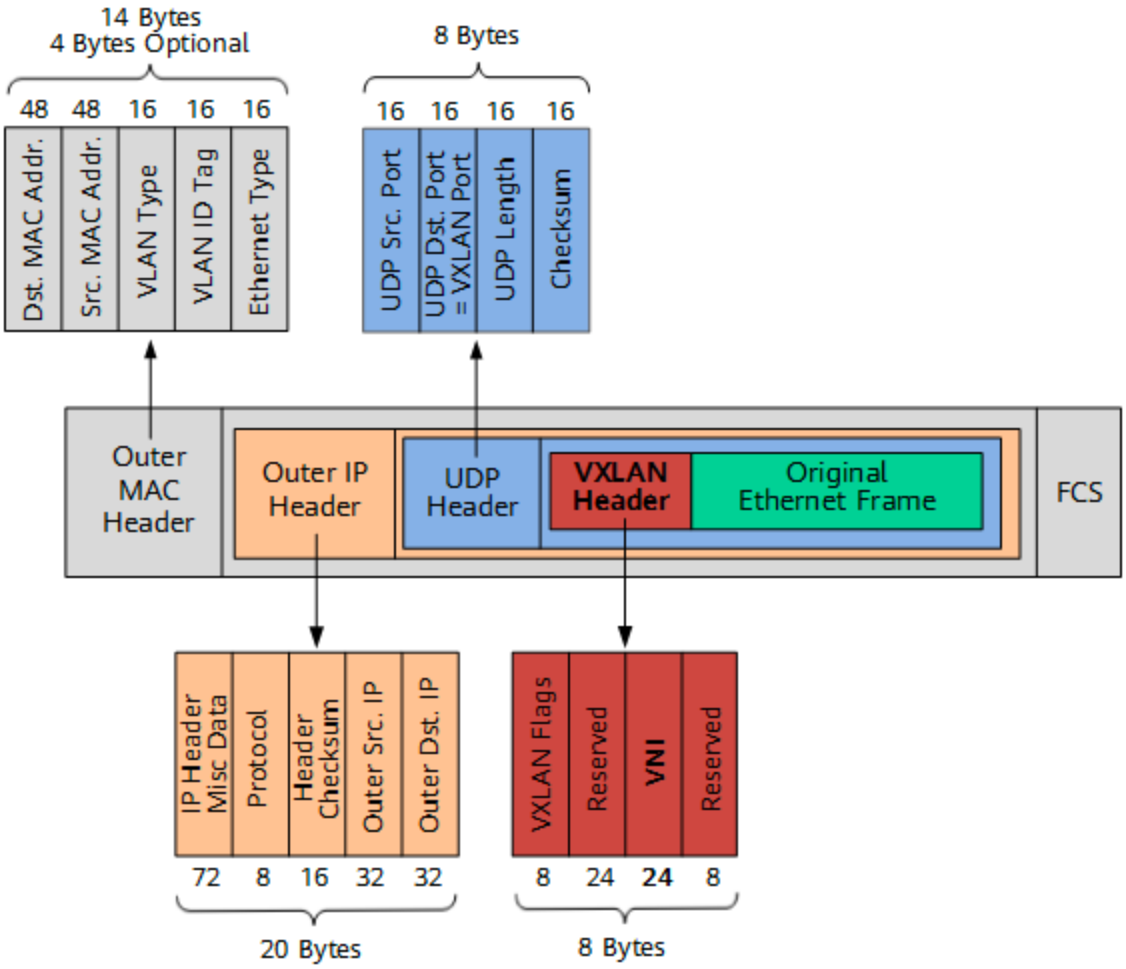
有了目的VTEP设备的MAC地址，Linux内核就可以开始二层封包工作了。这个二层帧的格式，如下所示：



可以看到，Linux内核会把目的VTEP设备的MAC地址，填写在图中的Inner Ethernet Header字段，得到一个二层数据帧。上述封包过程只是加一个二层头，不会改变原始IP包的内容。所以图中的Inner IP Header字段，依然是容器B的IP地址，即：10.244.2.3。

但是，上面提到的这些VTEP设备的MAC地址，对于宿主机网络来说并没有什么实际意义。所以上面封装出来的这个数据帧，并不能在宿主机二层网络里传输。为了方便叙述，接下来把它称为**内部数据帧**。

所以接下来，Linux内核还需要再把内部数据帧进一步封装成为宿主机网络里的一个普通的数据帧，好让它载着内部数据帧通过宿主机的eth0网卡进行传输。把这次要封装出来的、宿主机对应的数据帧称为**外部数据帧**。



为了实现这个搭便车的机制，Linux内核会在内部数据帧前面，加上一个特殊的VXLAN头，用来表示这个乘客实际上是一个VXLAN要使用的数据帧。而在这个**VXLAN头里有一个重要的标志叫作VNI，VNI是**VTEP设备判断某个数据帧是不是应该归自己处理的重要标识。在Flannel中，**VNI的默认值是1**，这也是为何，宿主机上的VTEP设备都叫作flannel.1的原因，这里的1其实就是VNI的值。

然后，Linux内核会把这个数据帧封装进一个UDP包里发出去。跟UDP模式类似，在宿主机看来，它会以为自己的flannel.1设备只是在向另一台宿主机的flannel.1设备，发起一次普通的UDP链接。

一个flannel.1设备只知道另一端的flannel.1设备的MAC地址，却不知道对应的宿主机地址是什么。**那这个UDP包应该发给哪台宿主机呢?** flannel.1设备实际上要扮演一个网桥的角色，在二层网络进行UDP包的转发。而在Linux内核里面，网桥设备进行转发的依据来自于一个叫作 `FDB` 的转发数据库。 `flannel.1` 网桥对应的 `FDB` 信息，也是 `flanneld` 进程负责维护的，可以通过bridge fdb命令查看到，如下所示：

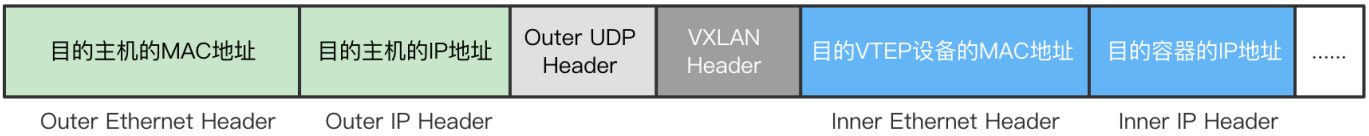
```shell
[root@k8s-node1 ~]# bridge fdb show flannel.1 | grep 82:9e:ca:29:46:96
82:9e:ca:29:46:96 dev flannel.1 dst 172.19.216.115 self permanent
```

在上面这条 `FDB` 记录里，指定了这样一条规则：发往目的 `VTEP` 设备（ `MAC` 地址是 `82:9e:ca:29:46:96` ）的二层数据帧，应该通过 `flannel.1` 设备，发往 `IP` 地址为 `172.19.216.115` 的主机。这台主机正是 `Node2` ，UDP包要发往的目的地就找到了。

所以接下来的流程就是一个正常的、宿主机网络上的封包工作

UDP包是一个四层数据包，所以Linux内核会在它前面加一个IP头（Outer IP Header），组成一个IP包。并且，在这个IP头里，会填上通过FDB查询出来的目的主机的IP地址，即Node2的IP地址172.19.216.115

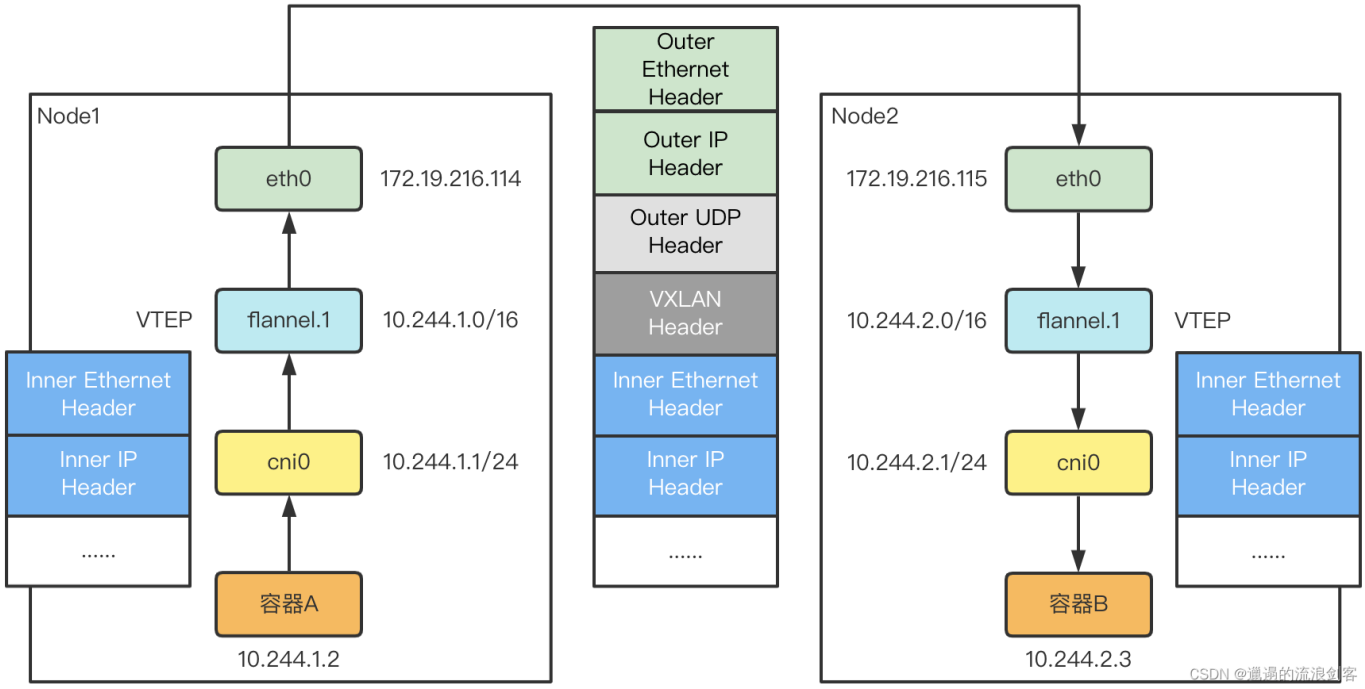
然后，Linux内核再在这个IP包前面加上二层数据帧头（Outer Ethernet Header），并把Node2的MAC地址填进去。这个MAC地址本身，是Node1的ARP表要学习的内容，无需Flannel维护。这时候，封装出来的外部数据帧的格式，如下图所示：



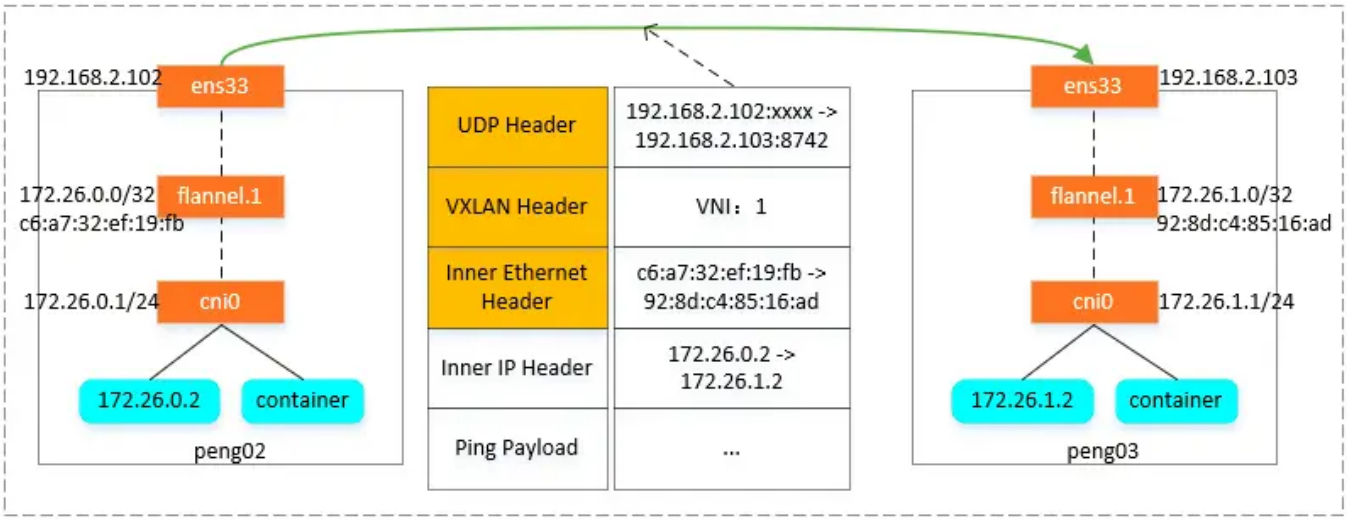
接下来，Node1上的flannel.1设备就可以把这个数据帧从Node1的eth0网卡发出来。这个帧经过宿主机网络来到Node2的eth0网卡。

这时候，Node2的内核网络栈会发现是这个数据帧里有VXLAN Header，并且VNI=1。所以Linux内核会对它进行拆包，拿到里面的内部数据帧，然后根据VNI的值，把它交给Node2上的flannel.1设备。

而flannel.1设备则会进一步拆包，取出原始IP包。接下来和主机内容器之间通信流程相同，最终，IP包进入到了容器B的Network Namespace里。



1.2.3 Vxlan补充说明–1



cni0是一个Linux虚拟网桥，flannel.1是一个Linux虚拟网络设备，类型为VXLAN，容器与cni0之间通过veth–pair连接。

假设节点Peng02上的容器172.26.0.2向节点Peng03上的容器172.26.1.2发送ping数据包（对应图上的Inner IP Header）。

首先数据包到达cni0网桥，cni0网桥接收到数据包后，最终会把包投递到主机的内核协议栈中（即会进入到ip_rcv函数）。根据下面的路由，数据包最终会从flannel.1网卡出去。

```shell
[root@peng02 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
...
172.26.0.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
172.26.1.0      172.26.1.0      255.255.255.0   UG    0      0        0 flannel.1
```

当数据包到达flannel.1时，需要对其进行二层封装。下一跳地址（上面那一条路由的网关）为172.26.1.0/32，于是就会查找flannel.1的ARP表（这个ARP表是由flanneld进行维护的），如下：

```shell
[root@peng02 ~]# ip neigh show dev flannel.1
172.26.1.0 lladdr 92:8d:c4:85:16:ad PERMANENT
```

查到找对应的ARP条目后，于是对数据包进行二层封装，即对应前面图中的Inner Ethernet Header。

此时，skb_buff要从flannel.1发送出去。设备flannel.1的类型为VXLAN，于是会调用VXLAN模块的发送函数vxlan_xmit（不同类型的设备，发送函数不一样）。

VXLAN模块首先检查flannel.1的VNI（所有节点的flannel1的VNI都为1，这也是为什么flannel叫flannel.1的原因），然后将对数据帧封装一层VXLAN Header，这个头部中会包含VNI。

接着，VXLAN模块继续进行UDP封装，（要进行UDP封装，就要知道四元组信息：源IP、源端口、目的IP、目的端口）。首先是目的端口，Linux内核中默认为VXLAN分配的UDP监听端口为8472；然后是源端口，源端口是根据封装的内部数据帧做一个哈希值得到。

接下来是目的IP。Linux下的VXLAN设备都有一个转发表。通过下面的命令查看flannel.1的转发表，如下：（由flanneld进程维护）：

```shell
[root@peng02 ~]# bridge fdb show dev flannel.1 | grep 92:8d:c4:85:16:ad
92:8d:c4:85:16:ad dev flannel.1 dst 192.168.2.103 self permanent
```

它记录着，目的MAC地址为92:8d:c4:85:16:ad的数据帧封装后，应该发往哪个目的IP。根据上面的记录可以看出，UDP的目的IP应该为192.168.2.103。

最后是源IP，任何一个VXLAN设备创建时都会指定一个三层物理网络设备作为VTEP，这个物理网络设备的IP就是UDP的源IP。可以使用以下的命令查看flannel.1的VTEP，发现是ens33，所以源IP是

192.168.2.102

```shell
$ ip -d link show flannel.1
...
```

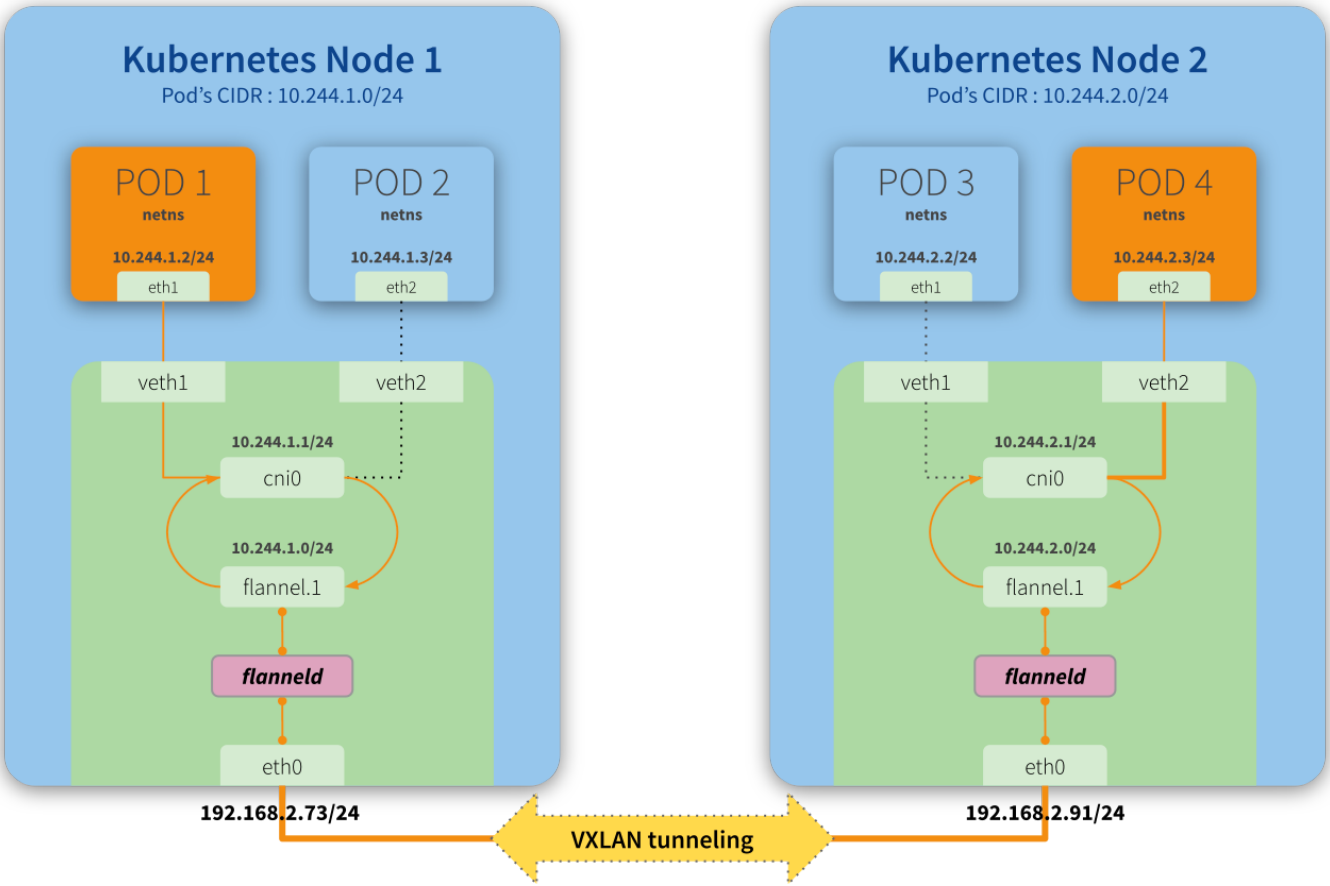
UDP的四元组确定后，vxlan调用UDP协议的发包函数进行发包，后面的过程和本机的UDP程序发包没什么区别。

当数据包到达Peng03的8472端口后（实际上就是VXLAN模块），VXLAN模块就会比较这个VXLAN Header中的VNI和本机的VTEP（VXLAN Tunnel End Point，就是flannel.1）的VNI是否一致，然后比较Inner Ethernet Header中的目的MAC地址与本机的flannel.1是否一致，都一致后，则去掉数据包的VXLAN Header和Inner Ethernet Header，然后把数据包从flannel.1网卡进行发送。

然后，在Peng03节中上会有如下的路由（由flanneld维护），根据路由判断要把数据包发送到cni0网卡上。

```shell
[root@peng03 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
...
172.26.1.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
```

1.2.4 Vxlan补充说明-2



首先数据包从pod1内的eth0出来到达cni0网桥，cni0网桥接收到数据包后发现目的IP跟自己不在一个网段,那么自然需要转发出去，而Linux Bridge有个特殊规则： **网桥不会将这个数据包转发给任何设备，而是直接转交给主机的三层协议栈进行处理**, 因此通过本机的route得知，目的地址为10.224.1.0段的数据包都将转到flannel.1

Shell | 复制代码

```
[node1]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.224.0.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
10.224.2.0      10.224.2.0      255.255.255.0   UG    0      0        0 flannel.1
```

我们抛开所有因素不谈，先说一个前提，**两台节点通信，不管虚拟网络上怎么实现，最终还是需要通过物理网卡进行**

flannel为overlay的容器网络，是个**大二层**互通的解决方案， 但最终网络包还是要借助物理网卡出去

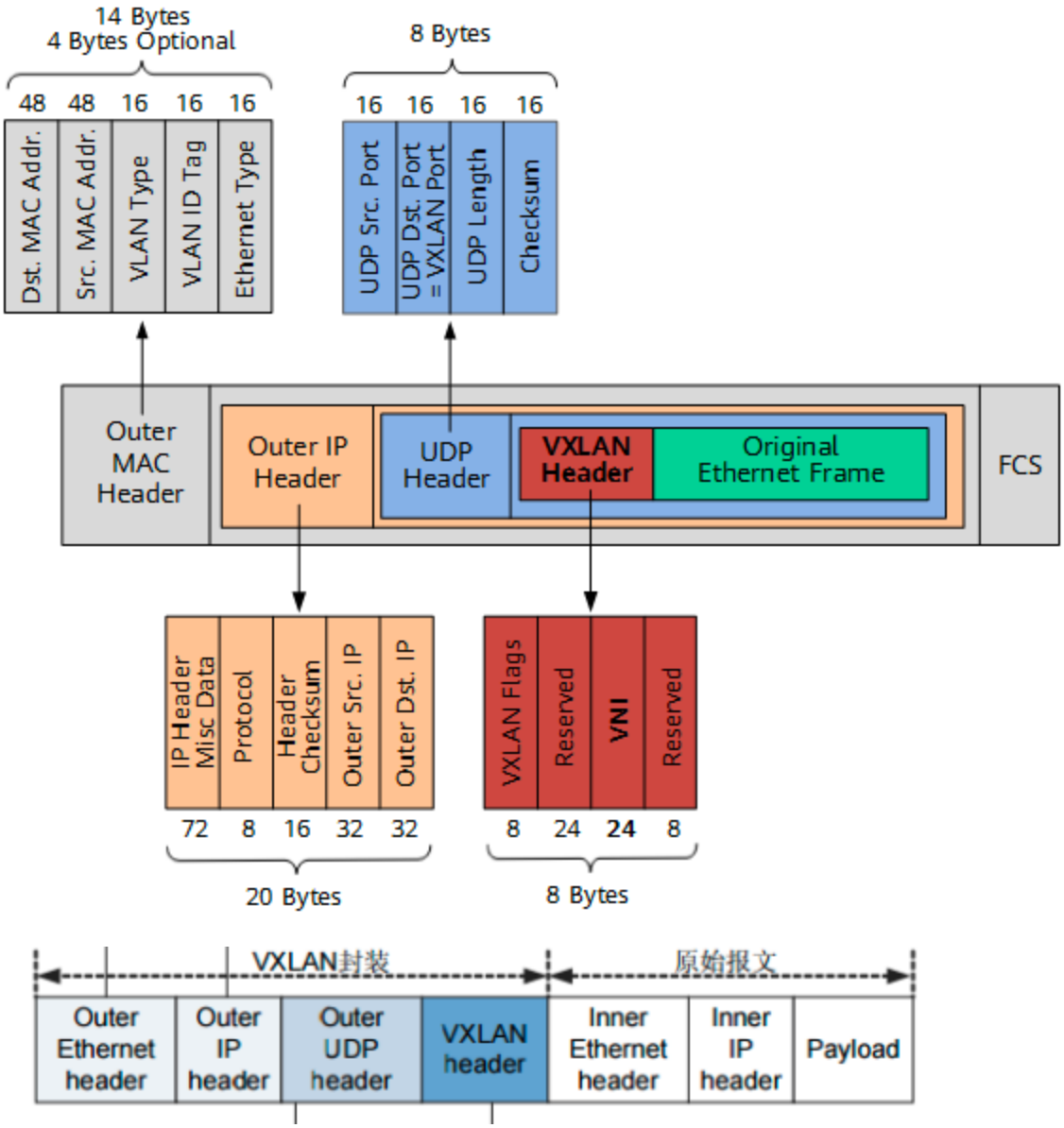
任何一个VXLAN设备创建时都会指定一个三层物理网络设备作为VTEP,flannel.1也不例外，可通过以下命令进行查看

```shell
ip -d link show flannel.1
# 在输出中可看到flannel.1是基于eth0上
```

当数据包到达flannel.1后，它如果要将这个包转到目的节点上去，它需要对vxlan报文进行填充

在源节点上flannel填充这些信息的过程就叫封包，在目标节点上解开这些信息就叫解包

先来看一下vxlan的报文形式



右边的为原始报文，左边的即为vxlan封装报文,我们一一来介绍一下.

Original Ethernet Frame

首先， Original Ethernet Frame是原始的报文，也就是pod1访问pod2的报文，因为是个正常网络报文，包含IP header、Ethernet header、及playload。

playload不多说，就是数据

IP header 很自然也就是pod1及pod2的ip地址信息

而Ethernet header需要注意的是，不是pod1及pod2的MAC地址，而应该是**两端flannel.1的MAC地址**,因为报文到flannel.1后通过`route -n`得到下一跳的地址为`10.224.2.0/32`，需要得到10.224.2.0的mac地址，这部分信息在**flanneld中进行维护，叫ARP表**，即通过IP可得到对应的MAC地址，如下：

```
1
2
```

```
[node1]# ip neigh show dev flannel.1
10.224.2.0 lladdr 92:8d:c4:85:16:ad PERMANENT
```

Vxlan Header

Vxlan header这里只需要关注一个字段，那就是VNI,前文简单提到过, 在目标node上的flannel.1上会对这个VNI字段进行check，看是否与自己的VNI一致，一致的话才会进行处理.

UDP Header

从上图中的颜色可以看出, UDP是把整个Original Ethernet Frame及Vxlan Header都囊括在一起了，我们知道，udp header中包含有源端口，目的端口，如图所示

所以很自然Src.port为node1上的flannel.1的端口，该端口是根据封装的内部数据帧计算出的一个哈希值

Dst.port(上面也显示为VxlanPort)为node2上flannel.1的端口，Linux内核中默认为VXLAN分配的UDP监听端口为8472

Outer IP header

上面的信息其实还都是在flannel中, 上面说过任何虚拟出来的网络最终都是要经过实体网卡设备出去,目前封装出来的udp header是不能在宿主机网络中传递的，只能进一步把udp再封装成正常的

网络包通过节点物理网络发送出去，根据tcp/ip协议，有了UDP header自然是需要封装在ip报文中，所以有ip header

ip header中包含有源ip及目的ip，源ip即为flannel.1所绑定的物理ip,即node1节点的eth0 ip

而目标ip，那肯定是node2的eth0 ip了，这个ip是需要根据目标flannel.1的mac地址获得，这部分信息同样维护在flanneld中的，可通过以下命令查询

```
1
2
```

```
[node1 ~]# bridge fdb show dev flannel.1 | grep 92:8d:c4:85:16:ad
92:8d:c4:85:16:ad dev flannel.1 dst 192.168.2.91 self permanent
```

192.168.2.19即为目标ip

可以总结一下，flanneld中维护了这两部分信息:

- flannel.1的ip与mac地址对应关系，通过flannel.1的ip可以查询到flannel.1 的mac地址
- flannel.1的mac地址及其所在node ip对应关系，通过flannel.1的mac地址可以查询到node ip

Outer MAC Header

而ip header则又需要由封装在mac header中，通过上面的查询，mac header 自然就是两台node的mac信息了

这样的话，整个封装包组成了标准的tcp/ip协议包，可以在物理网络上传递了

解包

当数据包到达node2的8472端口后（实际上就是VXLAN模块），VXLAN模块就会比较这个VXLAN Header中的VNI和本机的VTEP（VXLAN Tunnel End Point，就是flannel.1）的VNI是否一致，然后比较Inner Ethernet Header中的目的MAC地址与本机的flannel.1是否一致，都一致后，则去掉数据包的VXLAN Header和Inner Ethernet Header，然后把数据包从flannel.1网卡进行发送。

然后，在node2上会有如下的路由（由flanneld维护），根据路由判断要把数据包发送到cni0网卡上

```
1
2
3
4
5
```

```
[node2 ~]# route -n
Kernel IP routing table
Destination     Gateway
Genmask         Flags Metric Ref    Use Iface
...
10.224.2.0      0.0.0.0
255.255.255.0   U     0      0
      0 cni0
```

最后通过 inner ip header中的ip知道需要由10.224.2.2的pod进行响应.

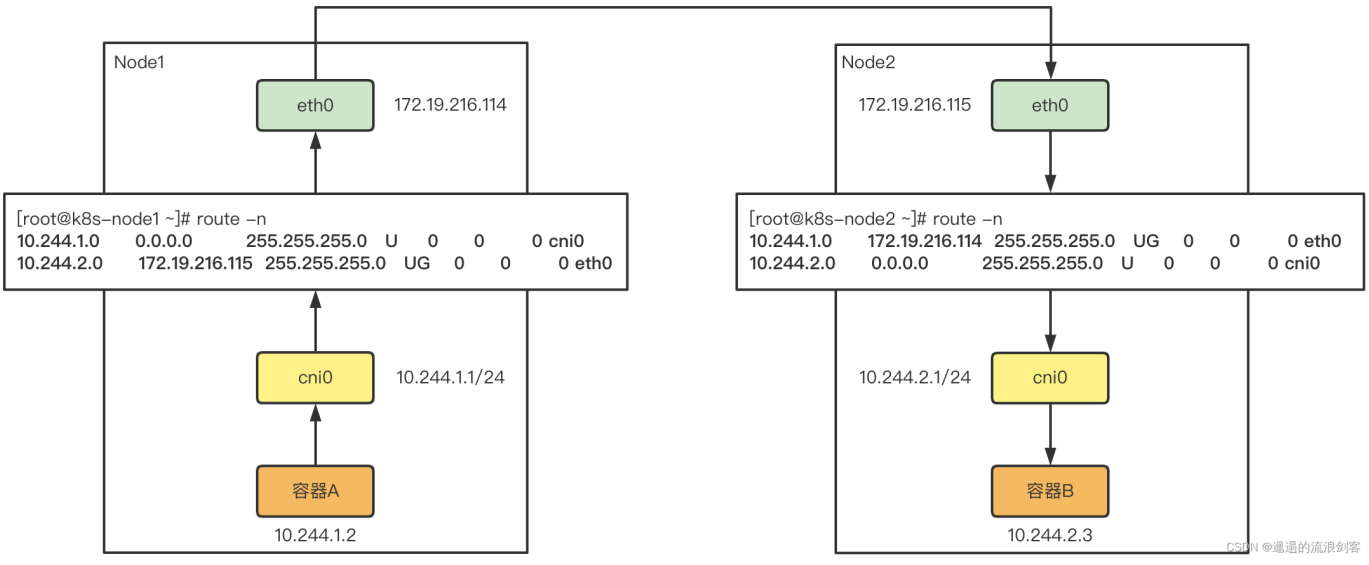
vxlan在内核中进行封装、解封装的过程，致使flannel vxlan的性能有所下降.

1.2.5 总结

经过封装，VXLAN隧道的两个端口是二层可通的；也就是说，VXLAN虚拟出来的是一个大二层网络。

1.2 Backend-Host-GW模式

Flannel的host-gw模式是一种纯三层网络方案，它的工作原理如下图所示：



还是上面这个例子，宿主机Node1（IP地址是172.19.216.114）上的容器A的IP地址是10.244.1.2，要访问宿主机Node2（IP地址是172.19.216.115）上的容器B的IP地址是10.244.2.3

当设置Flannel使用host-gw模式之后，flanneld会在宿主机上创建这样一条规则，以Node1为例：

```shell
[root@k8s-node1 ~]# route -n
...
10.244.2.0      172.19.216.115  255.255.255.0   UG    0      0        0 eth0

```

这条路由规则的含义是：目的IP地址属于10.244.2.0/24网段的IP包。应该经过本机的eth0设备发出去；并且，它的下一跳（next-hop）是172.19.216.115。

所谓下一跳地址就是：如果IP包从主机A发到主机B，需要经过路由设备X的中转。那么X的IP地址就应该配置为主机A的下一跳地址。

一旦配置了下一跳地址，那么接下来，当IP包从网络层进入链路层封装成帧的时候，eth0设备就会使用下一跳地址对应的MAC地址，作为该数据帧的目的MAC地址。显然，这个MAC地址正是Node2的MAC地址。

这样，这个数据帧就会从Node1通过宿主机的二层网络顺利到达Node2上。

而Node2的内核网络栈从二层数据帧里拿到IP包后，会看到这个IP包的目的IP地址是10.244.2.3，即容器B的IP地址。这时候，根据Node2上的路由表，目的地址会匹配到10.244.2.0对应的路由规则，从而进入cni0网桥，进而进入到容器B中。

host-gw模式的工作原理其实就是将每个Flannel子网的下一跳设置成了该子网对应的宿主机的IP地址。这个主机会充当这条容器通信路径里的网关，这也正是host-gw的含义

Flannel子网和主机的信息都是保存在Etcd当中的。flanneld只需要WATCH这些数据的变化，然后实时更新路由表即可。

而在这种模式下，容器通信的过程就免除了额外的封包和解包带来的性能损耗。

host-gw模式能够正常工作的核心，就在于IP包在封装成帧发出去的时候，会使用路由表的下一跳设置目的MAC地址。这样，它就会经过二层网络到达目的宿主机。**所以说，Flannel host-gw模式必须要求集群宿主机之间是二层连通的。**

宿主机之间二层不连通的情况也是广泛存在的。比如，宿主机分布在了不同的子网（VLAN）里。但是，在一个Kubernetes集群里，宿主机之间必须可以通过IP地址进行通信，也就是说至少是三层可达的。三层可达也可以通过为几个子网设置三层转发来实现。

1.3 Backend-IPIP模式