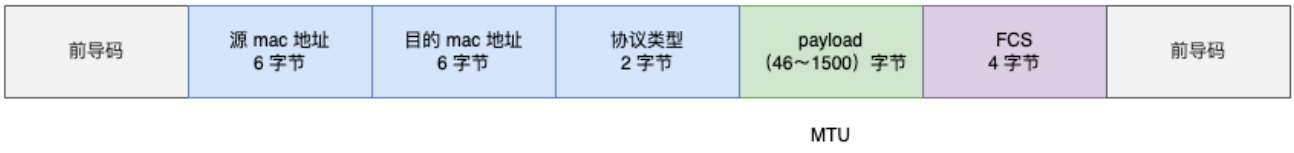
介绍

网络 OSI 七层模型

	层	数据单元	功能
7	应用层 Application Layer	数据 Data	提供为应用软件提供服务的接口，用于与其他应用软件之间的通信。典型协议：HTTP、HTTPS、FTP、Telnet、SSH、SMTP、POP3等。
6	表达层 Presentation Layer	数据 Data	把数据转换为能与接收者的系统格式兼容并适合传输的格式。
5	会话层 Session Layer	数据 Data	负责在数据传输中设置和维护计算机网络中两台计算机之间的通信连接。
4	传输层 Transport Layer	数据段 Segments	把传输表头加至数据以形成数据包。传输表头包含了所使用的协议等发送信息。典型协议：TCP、UDP、RDP、SCTP、FCP等。
3	网络层 Network Layer	数据包 Packets	决定数据的传输路径选择和转发，将网络表头附加至数据段后以形成报文（即数据包）。典型协议：IPv4/IPv6、IGMP、ICMP、EGP、RIP等。
2	数据链路层 Data Link Layer	数据帧 Frame	负责点对点的网络寻址、错误侦测和纠错。当表头和表尾被附加至数据包后，就形成数据帧（Frame）。典型协议：WiFi（802.11）、Ethernet（802.3）、PPP等。
1	物理层 Physical Layer	比特流 Bit	在局域网上传送数据帧，它负责管理电脑通信设备和网络媒体之间的互通。包括了针脚、电压、线缆规范、集线器、中继器、网卡、主机接口卡等。

协议

二层数据帧



其中：

- 前导码：用于区分数据帧；
- FCS：用于校验数据帧完整性；

`tcpdump` 的 `-e` 选项可以查看包的数据帧。示例如下：

```
# tcpdump -i bond0 'icmp and host 10.xx.xx.2' -vvv -n -e -X -s0
dropped privs to tcpdump
tcpdump: listening on bond0, link-type EN10MB (Ethernet), capture size
262144 bytes
17:46:15.138815 58:25:75:6f:19:45 > 58:25:75:6f:22:37, ethertype IPv4
(0x0800), length 98: (tos 0x0, ttl 64, id 12833, offset 0, flags [DF],
proto ICMP (1), length 84)
```

```
    10.xx.xx.1 > 10.xx.xx.2: ICMP echo request, id 42657, seq 1, length 64
        0x0000:  4500 0054 3221 4000 4001 c058 0a16 9a01  E..T2!@.@..X....
        0x0010:  0a16 9a02 0800 95c9 a6a1 0001 66c3 2168  ............f.!h
        0x0020:  0000 0000 6895 0c00 0000 0000 1011 1213  ....h...........
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
17:46:15.138845 58:25:75:6f:22:37 > 58:25:75:6f:19:45, ethertype IPv4
(0x0800), length 98: (tos 0x0, ttl 64, id 1574, offset 0, flags [none],
proto ICMP (1), length 84)
    10.xx.xx.2 > 10.xx.xx.1: ICMP echo reply, id 42657, seq 1, length 64
        0x0000:  4500 0054 0626 0000 4001 2c54 0a16 9a02  E..T.&..@.,T....
        0x0010:  0a16 9a01 0000 9dc9 a6a1 0001 66c3 2168  ............f.!h
        0x0020:  0000 0000 6895 0c00 0000 0000 1011 1213  ....h...........
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
```

示例1: tcpdump icmp

在客户端 10.xx.xx.1 ping 服务端 10.xx.xx.2，抓包如上。

其中，icmp 请求包数据帧的源 mac 地址为 `58:25:75:6f:19:45`，目的地址为 `58:25:75:6f:22:37`，数据帧类型为 `0x800`，数据帧长度为 98。

length = len(source mac) + len(dest mac) + type + payload = 6 + 6 + 2 + 84 = 98。

vlan

vlan 用于二层隔离。

服务器接口如果为 vlan 接口，出去的数据帧会带上 vlan id（不过抓包时没看到，可能是抓的姿势不对）。这里的隔离主要是靠交换机，交换机会转发同一 vlan 下的包，隔离不通 vlan 的包。

交换机是怎么做到的?

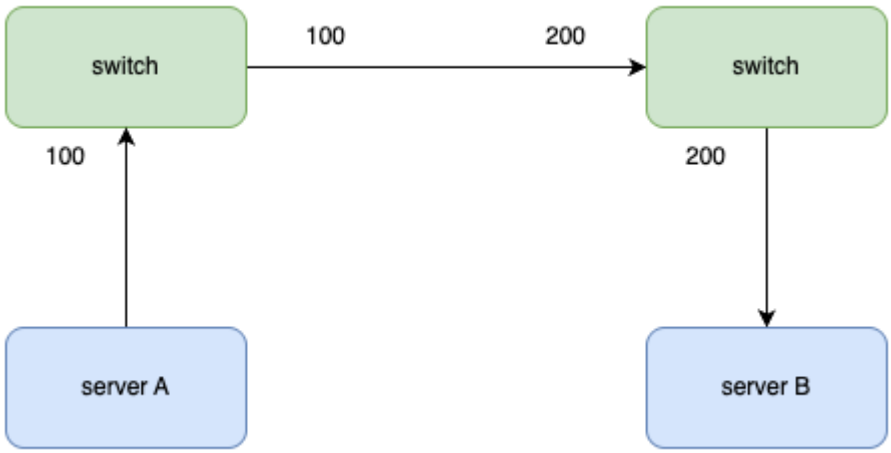
交换机的 access 口是一对一连接服务器的端口，access 配置规则为：

- 如果数据帧带 vlan，检查 vlan 是否允许放通，允许则转发该数据帧。否则将丢弃该数据帧；
- 如果数据帧不带 vlan，交换机会打上默认 vlan，转发该数据帧；

交换机的 trunk 是连接交换机的端口。trunk 口配置规则为：

- 如果数据帧带 vlan，检查该 vlan 是否允许放通，允许则转发。否则将丢弃；
- 如果数据帧不带 vlan，为该数据帧打上 native vlan，转发；
- 如果数据帧是 native vlan，会被交换机剥离 vlan 在发送；（这样的考量是基于 native vlan 是隐式发送，不应该显式标记的考量）

如果交换机两端 trunk 口配置不一致会造成 vlan 跳跃的问题：



从 server A 发出去的 vlan100 的数据帧经过交换机的 trunk 口会被剥离（vlan 和 native vlan 一致）发往交换机 native vlan 200 的 trunk 口，交换机为该数据帧打上 vlan200 的标签，在发往 access vlan 200 的口，access 剥离数据帧发送至 server B，导致 server B 和 server A vlan 不同却可以访问。

解决方案也很简单，配置 trunk 口的 native vlan 一致即可。

MTU

这里的 MTU 指的是最大传输的数据帧单元。对于以太网其 MTU 的最大值为 1500 字节。

思考：如果 MTU 设置过大/过小会怎么样?

数据帧是通过网线传输，网线的数据帧传递是顺序的。如果传送的数据帧过大，会导致其它数据帧延迟发送。假设 MTU=65536，传输数据帧=65536，网络带宽 10M/s，传送一个数据帧需要 655368/101024*1024=50ms。50ms 的延时太大了。

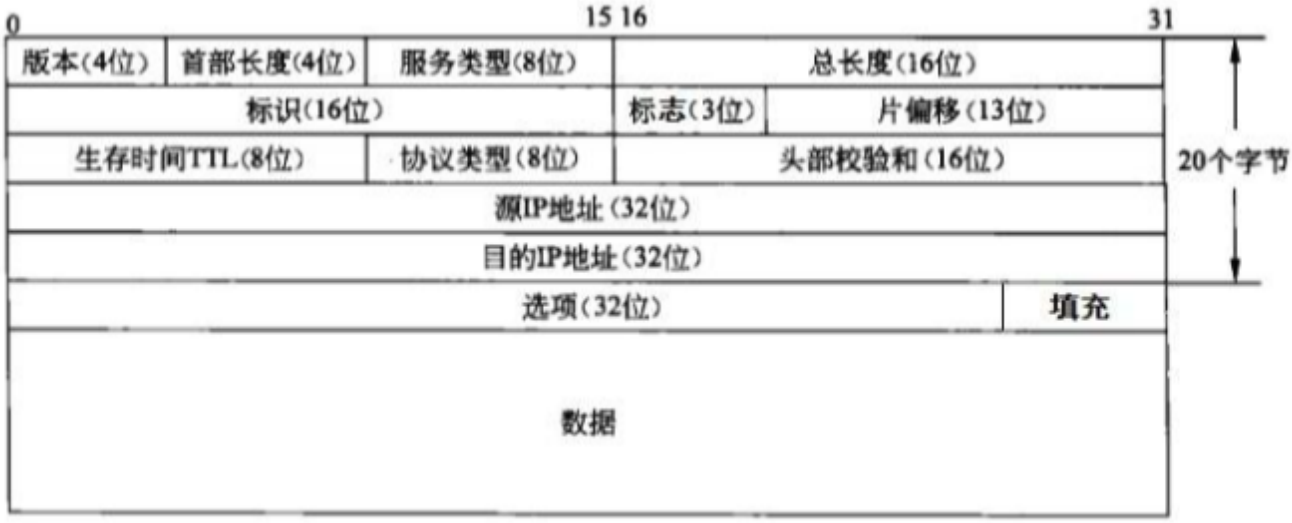
如果数据帧设置过小，会导致数据报被频繁的分片，重组，消耗系统资源。

通过 `ip a` 查看网卡的 MTU 如下：

```
# ifconfig bond0
bond0: flags=5187<UP,BROADCAST,RUNNING,MASTER,MULTICAST>  mtu 1500
        inet 10.xx.xx.1  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::5a25:75ff:fe6f:1945  prefixlen 64  scopeid 0x20<link>
        ether 58:25:75:6f:19:45  txqueuelen 1000  (Ethernet)
        RX packets 305929605  bytes 201059285272 (187.2 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 288572902  bytes 140082405006 (130.4 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ip 协议

ip 协议是三层网络层协议。其报文格式如下：



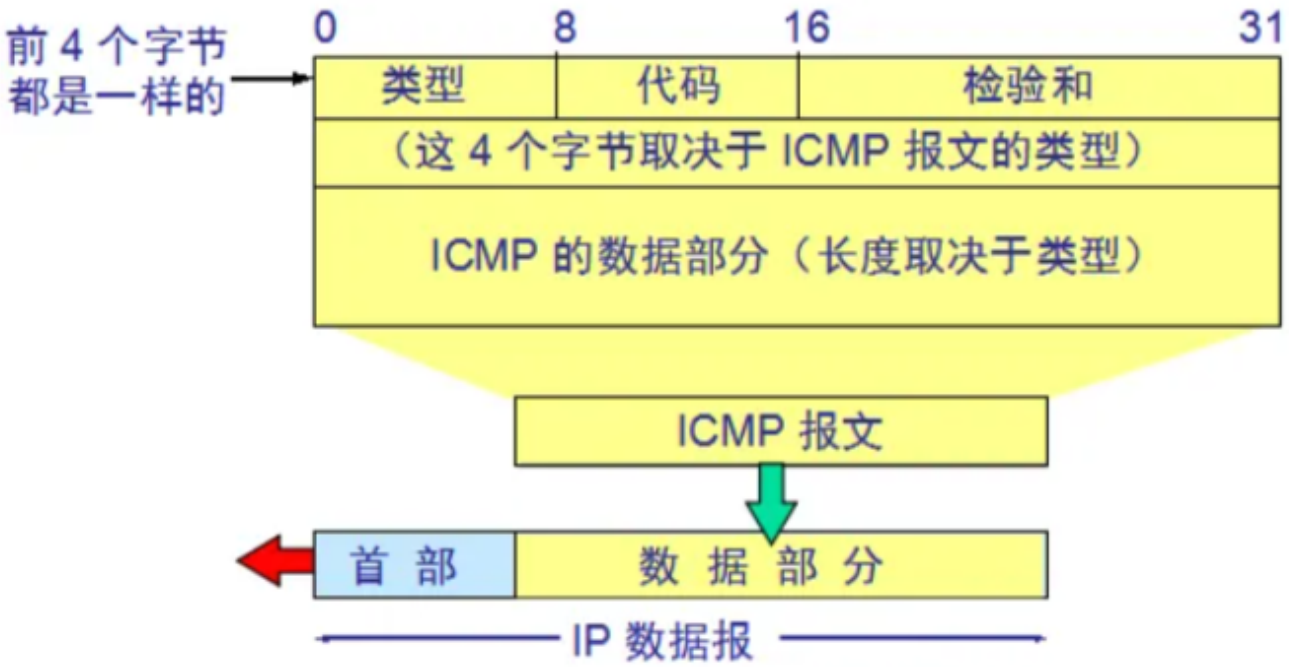
内核会根据 MTU 对 ip 数据报进行分片/重组等操作。

如示例 1 所示，ip 数据报的源地址为 10.xx.xx.1，目的地址为 10.xx.xx.2，版本 ipv4，首部长度 20，服务类型 tos 0x0，生存时间 64，片偏移 0，flags DF（不分片），协议类型 ICMP，总长度 84。

数据是 icmp 报文。

icmp 协议

icmp 协议是基于 ip 协议的网络层协议。其格式如下：



在示例 1 中，icmp 长度为 64 字节。icmp 头占 8 字节，数据部分占 56 字节。

ip 分片

如果 ip 数据报超过 MTU 会被分片。示例如下：

```
# ping -M do -s 1800 -c 1 10.xx.xx.2
PING 10.xx.xx.2 (10.xx.xx.2) 1800(1828) bytes of data.
ping: local error: message too long, mtu=1500

--- 10.xx.xx.2 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

`-M do` 设置不分片，传输包 1800 超过 MTU 1500 限制，ping 发送失败，报错 `message too long, mtu=1500`。

更新 ip 报的标志位允许分片。

客户端：

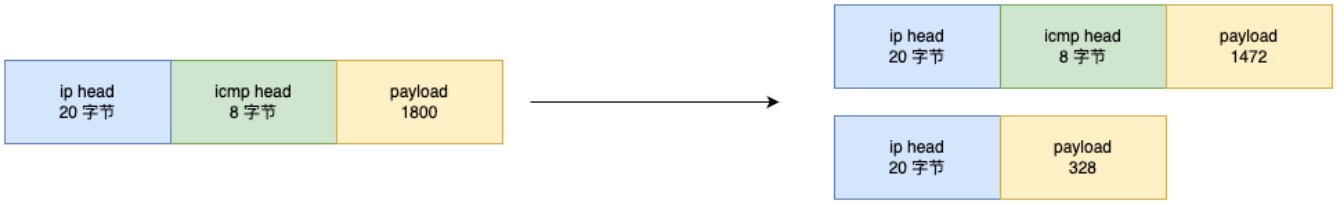
```
# ping -M dont -s 1800 -c 1 10.xx.xx.2
PING 10.xx.xx.2 (10.xx.xx.2) 1800(1828) bytes of data.
1808 bytes from 10.xx.xx.2: icmp_seq=1 ttl=64 time=0.165 ms

--- 10.xx.xx.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.165/0.165/0.165/0.000 ms
```

服务端：

```
# tcpdump -i bond0 'icmp and host 10.xx.xx.2' -vvv -n
20:45:10.884316 IP (tos 0x0, ttl 64, id 53715, offset 0, flags [+], proto
ICMP (1), length 1500)
    10.xx.xx.1 > 10.xx.xx.2: ICMP echo request, id 25966, seq 1, length
1480
20:45:10.884326 IP (tos 0x0, ttl 64, id 53715, offset 1480, flags [none],
proto ICMP (1), length 348)
    10.xx.xx.1 > 10.xx.xx.2: ip-proto-1
20:45:10.884361 IP (tos 0x0, ttl 64, id 48228, offset 0, flags [+], proto
ICMP (1), length 1500)
    10.xx.xx.2 > 10.xx.xx.1: ICMP echo reply, id 25966, seq 1, length 1480
20:45:10.884364 IP (tos 0x0, ttl 64, id 48228, offset 1480, flags [none],
proto ICMP (1), length 348)
    10.xx.xx.2 > 10.xx.xx.1: ip-proto-1
```

数据包被分为两段，如下：



注意，第二个分片数据报无 icmp 头。为什么没有，因为根据 ip 偏移就可以重组报文。

arp 协议

arp 协议是地址解析协议，其通过 ip 获取目标机器的 mac 地址，工作在二层。

arping 获取目标地址所在的网卡 MAC 地址。如下：

客户端：

```
# arping 10.xx.xx.2
ARPING 10.xx.xx.2 from 10.xx.xx.1 bond0
Unicast reply from 10.xx.xx.2 [58:25:75:6F:22:37]  0.599ms
Unicast reply from 10.xx.xx.2 [58:25:75:6F:22:37]  0.602ms
```

有必要解释下 arping 命令的输出：

- ARPING 10.xx.xx.2 from 10.xx.xx.1 bond0，从本地 bond0 发出 arp 广播包，广播包的源 ip 是 10.xx.xx.1，目的 ip 是 10.xx.xx.2；
- Unicast reply from 10.xx.xx.2 [58:25:75:6F:22:37]，从目的 ip 发过来的单播包，目的 MAC 地址为 58:25:75:6F:22:37；

服务端：

```
# tcpdump -i bond0 'host 10.xx.xx.2' -vvv -n
dropped privs to tcpdump
tcpdump: listening on bond0, link-type EN10MB (Ethernet), capture size
262144 bytes

21:18:08.963175 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has
10.xx.xx.2 (Broadcast) tell 10.xx.xx.1, length 46
21:18:08.963187 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.xx.xx.2 is-
at 58:25:75:6f:22:37, length 28
21:18:09.963206 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has
10.xx.xx.2 (58:25:75:6f:22:37) tell 10.xx.xx.1, length 46
21:18:09.963217 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.xx.xx.2 is-
at 58:25:75:6f:22:37, length 28
```

输出解释如下：

- ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.xx.xx.2 (Broadcast) tell 10.xx.xx.1：二层广播包，广播包内容为 `who-has 10.xx.xx.2 (Broadcast) tell 10.xx.xx.1`。
- Reply 10.xx.xx.2 is-at 58:25:75:6f:22:37：发给客户端的单播包。

操作系统维护 arp 表，如下：

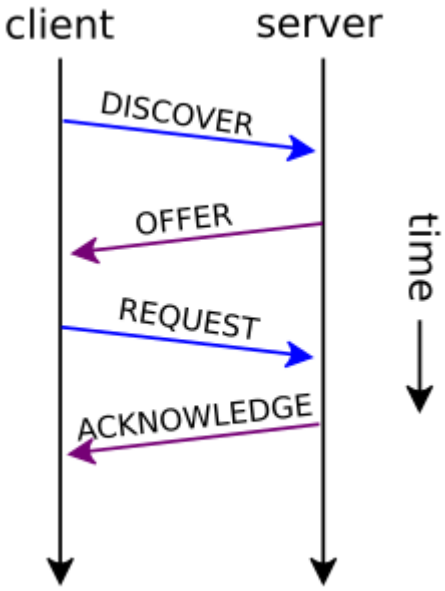
```
# arp -en
Address                  HWtype  HWaddress           Flags Mask
Iface
100.72.xx.xx                      (incomplete)
bond2.1920
100.71.xx.xx             ether   54:2b:de:e5:ee:01   C
bond0.1239
100.72.xx.xx                      (incomplete)
bond1.3625
100.73.xx.xx             ether   28:11:ec:b1:3e:b1   C
bond2.1920
```

arp 是工作在二层的，如果跨网段，arp 表中只会记录网关对应的 MAC 地址。

arp 应用非常广泛，自上而下封装的 ip 包只有目的 ip 地址，并没有目的 MAC 地址信息。这时候就需要系统发送 arp 广播包获取目标机器的 MAC 地址。

dhcp 协议

dhcp 动态主机配置协议是用于分配机器 ip 地址的协议，其工作在 udp 层。协议的交互流程图如下：



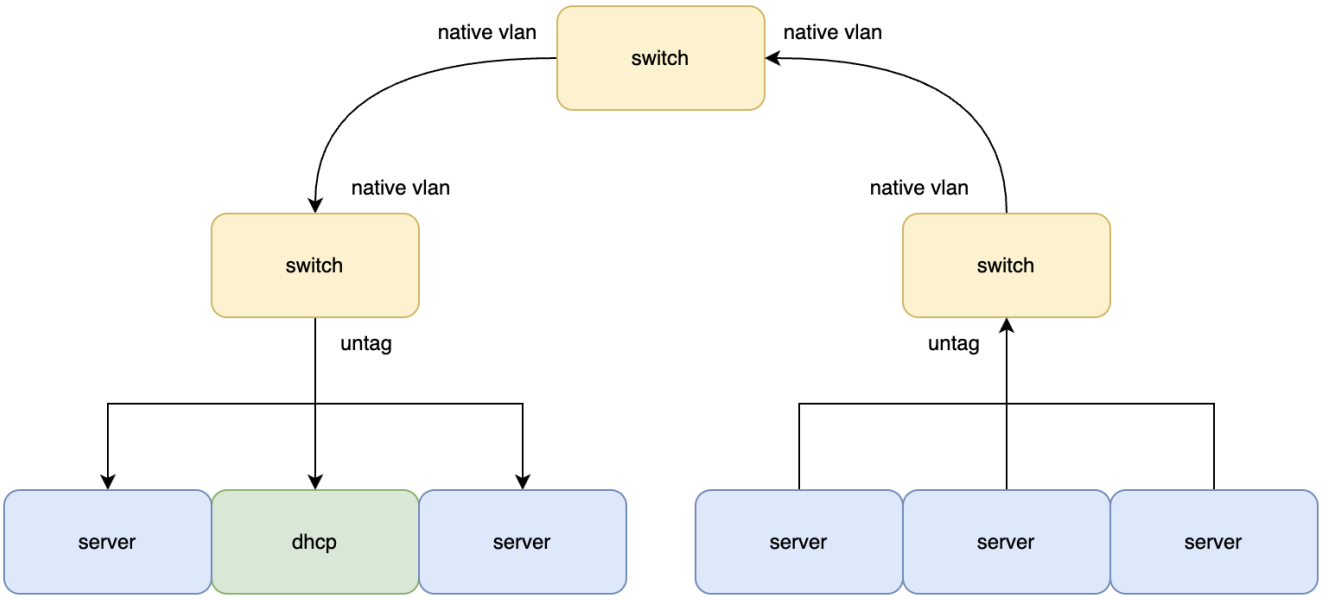
dhcp 协议工作流程

1. 机器发起 dhcp Discover 广播包获取 ip 地址。请求包是 udp 包，一般请求包自身端口 68，dhcp 服务器端口 67。请求包的源地址为 0.0.0.0:68，目的地址为 255.255.255.255:67。

这里目的地址 255.255.255.255:67 为广播包，这个包不是子网广播，怎么广播呢?

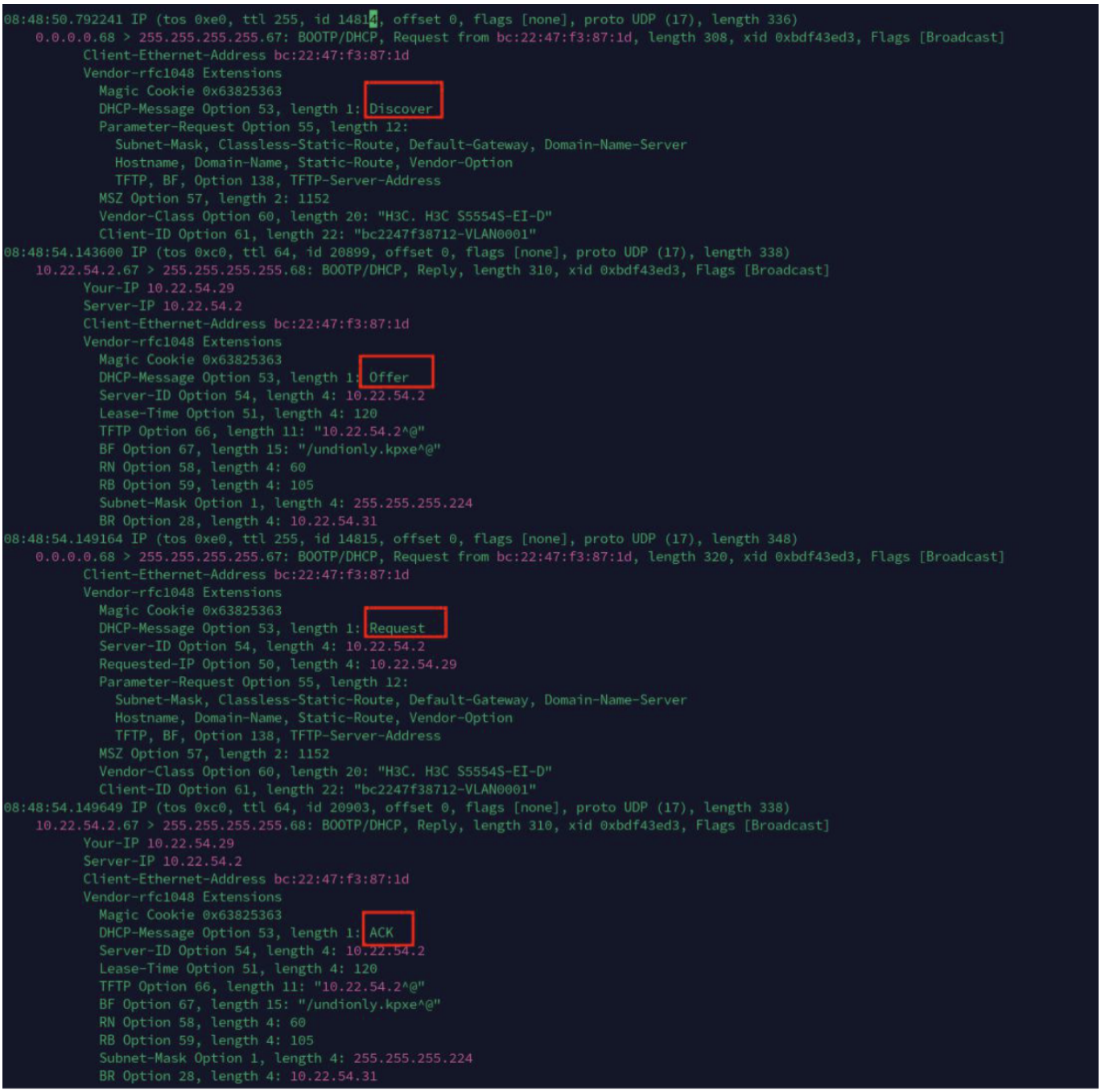
通过 vlan 隔离，交换机会将这个包转发到同一 vlan 下的机器，对于 dhcp 请求包，只有 dhcp 服务器会回应该请求包，其它服务器发现目的地址不是自己将丢弃该包，路由器不会转发 dhcp 包。

既然通过 vlan 隔离，可以设置 native vlan 打通交换机。示意图如下：



2. dhcp 服务器收到 Discover 包，回应客户端 Offer 包。数据报源 ip 地址是自己，目的 ip 地址为 0.0.0.0:68，这个包是广播包，只有机器收到这个包之后。
3. 机器根据 Offer 包中自己可以获取的 ip，发送请求广播包。注意这里为什么还是广播包，是想让同 vlan 下的其它 dhcp 知道自己请求的地址。如果其它 dhcp 发现请求的地址不是自己分配的则不继续响应该包。
4. dhcp 服务器发 Offer 提供机器 ip 地址。

抓包查看 dhcp 过程如下：



dhcp 组件

dnsmasq 是一个开源的，提供 dns 域名解析和 dhcp 服务的软件。

部署 dnsmasq 作为 dhcp 服务器，查看其日志输出如下：

```
dnsmasq-dhcp: 373865281 available DHCP range: 10.xx.xxx.10 --
10.xx.xxx.250
dnsmasq-dhcp: 373865281 client provides name: bmh6
dnsmasq-dhcp: 373865281 DHCPDISCOVER(bond0) 58:25:75:6f:14:a0
dnsmasq-dhcp: 373865281 tags: bond0
dnsmasq-dhcp: 373865281 DHCPOFFER(bond0) 10.xx.xxx.96 58:25:75:6f:14:a0
dnsmasq-dhcp: 373865281 requested options: 1:netmask, 2:time-offset,
6:dns-server, 12:hostname,
dnsmasq-dhcp: 373865281 requested options: 15:domain-name, 26:mtu,
28:broadcast, 121:classless-static-route,
```

```
dnsmasq-dhcp: 373865281 requested options: 3:router, 33:static-route,
40:nis-domain,
dnsmasq-dhcp: 373865281 requested options: 41:nis-server, 42:ntp-server,
119:domain-search,
dnsmasq-dhcp: 373865281 requested options: 249, 252, 17:root-path
dnsmasq-dhcp: 373865281 bootfile name: /undionly.kpxe
dnsmasq-dhcp: 373865281 server name: 10.xx.xxx.2
dnsmasq-dhcp: 373865281 next server: 10.xx.xxx.2
dnsmasq-dhcp: 373865281 sent size:  1 option: 53 message-type  2
dnsmasq-dhcp: 373865281 sent size:  4 option: 54 server-identifier
10.22.xxx.2
dnsmasq-dhcp: 373865281 sent size:  4 option: 51 lease-time  2m
dnsmasq-dhcp: 373865281 sent size:  4 option: 58 T1  1m
dnsmasq-dhcp: 373865281 sent size:  4 option: 59 T2  1m45s
dnsmasq-dhcp: 373865281 sent size:  4 option:  1 netmask  255.255.255.0
dnsmasq-dhcp: 373865281 sent size:  4 option: 28 broadcast  10.xx.xxx.255
```

请求的 ip 是需要续约，不用的话需要释放，这点有点类似于 kubernetes 中的 lease。