# 1.ARP出现原因

ARP协议是“Address Resolution Protocol”（地址解析协议）的缩写。其作用是在以太网环境中，数据的传输所依懒的是MAC地址而非IP地址，而将已知IP地址转换为MAC地址的工作是由ARP协议来完成的。

在局域网中，网络中实际传输的是“帧”，帧里面是有目标主机的MAC地址的。在以太网中，一个主机和另一个主机进行直接通信，必须要知道目标主机的MAC地址。但这个目标MAC地址是如何获得的呢？它就是通过地址解析协议获得的。所谓“地址解析”就是主机在发送帧前将目标IP地址转换成目标MAC地址的过程。ARP协议的基本功能就是通过目标设备的IP地址，查询目标设备的MAC地址，以保证通信的顺利进行。

# 2. ARP映射方式

## 2.1. 静态映射

静态映射的意思是要手动创建一张ARP表，把逻辑（IP）地址和物理地址关联起来。这个ARP表储存在网络中的每一台机器上。例如，知道其机器的IP地址但不知道其物理地址的机器就可以通过查ARP表找出对应的物理地址。这样做有一定的局限性，因为物理地址可能发生变化：

（1）机器可能更换NIC（网络适配器），结果变成一个新的物理地址。

（2）在某些局域网中，每当计算机加电时，他的物理地址都要改变一次。

（3）移动电脑可以从一个物理网络转移到另一个物理网络，这样会时物理地址改变。

要避免这些问题出现，必须定期维护更新ARP表，此类比较麻烦而且会影响网络性能。

## 2.2. 动态映射

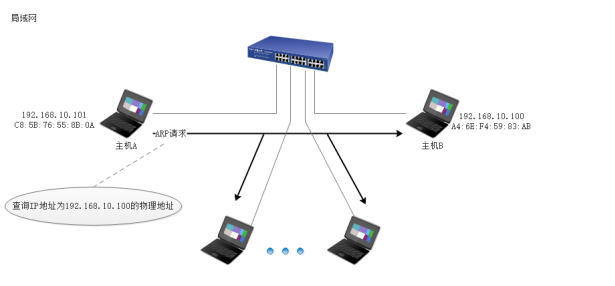
动态映射时，每次只要机器知道另一台机器的逻辑（IP）地址，就可以使用协议找出相对应的物理地址。已经设计出的实现了动态映射协议的有ARP和RARP两种。ARP把逻辑（IP）地址映射为物理地址。RARP把物理地址映射为逻辑（IP）地址。

# 3. ARP原理及流程

在任何时候，一台主机有IP数据报文发送给另一台主机，它都要知道接收方的逻辑（IP）地址。但是IP地址必须封装成帧才能通过物理网络。这就意味着发送方必须有接收方的物理（MAC）地址，因此需要完成逻辑地址到物理地址的映射。而ARP协议可以接收来自IP协议的逻辑地址，将其映射为相应的物理地址，然后把物理地址递交给数据链路层。

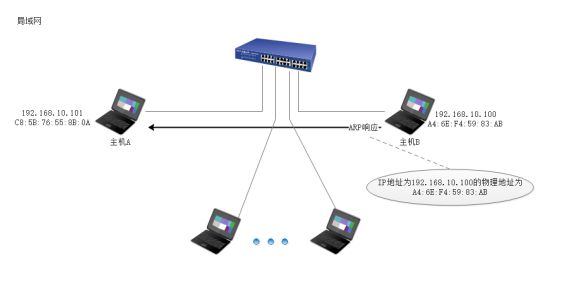
## 3.1.ARP请求

任何时候，当主机需要找出这个网络中的另一个主机的物理地址时，它就可以发送一个ARP请求报文，这个报文包好了发送方的MAC地址和IP地址以及接收方的IP地址。因为发送方不知道接收方的物理地址，所以这个查询分组会在网络层中进行广播。



## 3.2.ARP响应

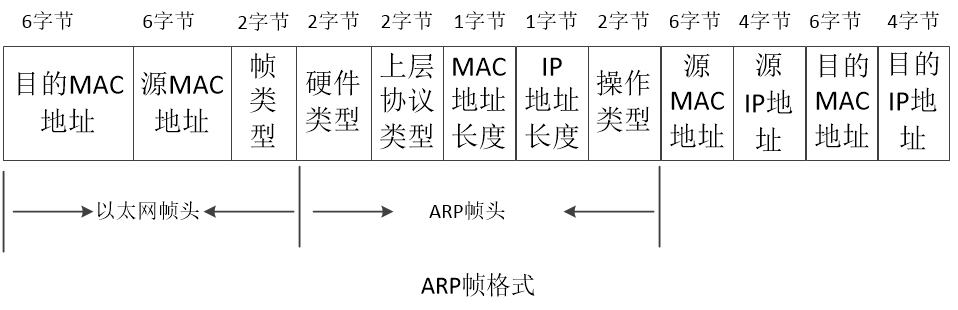
局域网中的每一台主机都会接受并处理这个ARP请求报文，然后进行验证，查看接收方的IP地址是不是自己的地址，只有验证成功的主机才会返回一个ARP响应报文，这个响应报文包含接收方的IP地址和物理地址。这个报文利用收到的ARP请求报文中的请求方物理地址以单播的方式直接发送给ARP请求报文的请求方。



# 4. **ARP协议报文字段抓包解析**

## 4.1. **报文格式**





**硬件类型**：16位字段，用来定义运行ARP的网络类型。每个局域网基于其类型被指派一个整数。例如：以太网的类型为1。ARP可用在任何物理网络上。

**协议类型**：16位字段，用来定义使用的协议。例如：对IPv4协议这个字段是0800。ARP可用于任何高层协议。

**硬件长度**：8位字段，用来定义物理地址的长度，以字节为单位。例如：对于以太网的值为6。

**协议长度**：8位字段，用来定义逻辑地址的长度，以字节为单位。例如：对于IPv4协议的值为4。

**操作码**：16位字段，用来定义报文的类型。已定义的分组类型有两种：ARP请求（1），ARP响应。

**源硬件地址**：这是一个可变长度字段，用来定义发送方的物理地址。例如：对于以太网这个字段的长度是6字节。

**源逻辑地址**：这是一个可变长度字段，用来定义发送方的逻辑（IP）地址。例如：对于IP协议这个字段的长度是4字节。

**目的硬件地址**：这是一个可变长度字段，用来定义目标的物理地址，例如，对以太网来说这个字段位6字节。对于ARP请求报文，这个字段为全0，因为发送方并不知道目标的硬件地址。

**目的逻辑地址**：这是一个可变长度字段，用来定义目标的逻辑（IP）地址，对于IPv4协议这个字段的长度为4个字节。

# **5．Arp源代码**：

## 5.1 Arp\_receive.c

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

#include <unistd.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <netinet/if\_ether.h>

#include <net/if\_arp.h>

#include <net/ethernet.h>

/\* 以太网帧首部长度 \*/

#define ETHER\_HEADER\_LEN sizeof(struct ether\_header)

/\* 整个arp结构长度 \*/

#define ETHER\_ARP\_LEN sizeof(struct ether\_arp)

/\* 以太网 + 整个arp结构长度 \*/

#define ETHER\_ARP\_PACKET\_LEN ETHER\_HEADER\_LEN + ETHER\_ARP\_LEN

/\* IP地址长度 \*/

#define IP\_ADDR\_LEN 4

void Err\_exit(const char \*err\_msg)

{

perror(err\_msg);

exit(1);

}

int main(void)

{

struct ether\_arp \*arp\_packet;

char buf[ETHER\_ARP\_PACKET\_LEN];

int sock\_raw\_fd, ret\_len, i;

if ((sock\_raw\_fd = socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ARP))) == -1)

Err\_exit("socket()");

while (1)

{

bzero(buf, ETHER\_ARP\_PACKET\_LEN);

ret\_len = recv(sock\_raw\_fd, buf, ETHER\_ARP\_PACKET\_LEN, 0);

if (ret\_len > 0)

{

/\* 剥去以太头部 \*/

arp\_packet = (struct ether\_arp \*)(buf + ETHER\_HEADER\_LEN);

/\* arp操作码为2代表arp应答 \*/

if (ntohs(arp\_packet->arp\_op) == 2)

{

printf("==========================arp replay======================\n");

printf("from ip:");

for (i = 0; i < IP\_ADDR\_LEN; i++)

printf(".%u", arp\_packet->arp\_spa[i]);

printf("\nfrom mac");

for (i = 0; i < ETH\_ALEN; i++)

printf(":%02x", arp\_packet->arp\_sha[i]);

printf("\n");

}

}

}

close(sock\_raw\_fd);

return 0;

}

## 5.2 Arp\_request.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/ioctl.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <netinet/if\_ether.h>

#include <net/ethernet.h>

#include <net/if\_arp.h>

#include <net/if.h>

#include <netpacket/packet.h>

/\* 以太网帧首部长度 \*/

#define ETHER\_HEADER\_LEN sizeof(struct ether\_header)

/\* 整个arp结构长度 \*/

#define ETHER\_ARP\_LEN sizeof(struct ether\_arp)

/\* 以太网 + 整个arp结构长度 \*/

#define ETHER\_ARP\_PACKET\_LEN ETHER\_HEADER\_LEN + ETHER\_ARP\_LEN

/\* IP地址长度 \*/

#define IP\_ADDR\_LEN 4

/\* 广播地址 \*/

#define BROADCAST\_ADDR {0xff, 0xff, 0xff, 0xff, 0xff, 0xff}

void Err\_exit(const char \*err\_msg)

{

perror(err\_msg);

exit(1);

}

/\* 填充arp包 \*/

struct ether\_arp \*fill\_arp\_packet(const unsigned char \*src\_mac\_addr, const char \*src\_ip, const char \*dst\_ip)

{

struct ether\_arp \*arp\_packet;

struct in\_addr src\_in\_addr, dst\_in\_addr;

unsigned char dst\_mac\_addr[ETH\_ALEN] = BROADCAST\_ADDR;

/\* IP地址转换 \*/

inet\_pton(AF\_INET, src\_ip, &src\_in\_addr);

inet\_pton(AF\_INET, dst\_ip, &dst\_in\_addr);

/\* 整个arp包 \*/

arp\_packet = (struct ether\_arp \*)malloc(ETHER\_ARP\_LEN);

arp\_packet->arp\_hrd = htons(ARPHRD\_ETHER);

arp\_packet->arp\_pro = htons(ETHERTYPE\_IP);

arp\_packet->arp\_hln = ETH\_ALEN;

arp\_packet->arp\_pln = IP\_ADDR\_LEN;

arp\_packet->arp\_op = htons(ARPOP\_REQUEST);

memcpy(arp\_packet->arp\_sha, src\_mac\_addr, ETH\_ALEN);

memcpy(arp\_packet->arp\_tha, dst\_mac\_addr, ETH\_ALEN);

memcpy(arp\_packet->arp\_spa, &src\_in\_addr, IP\_ADDR\_LEN);

memcpy(arp\_packet->arp\_tpa, &dst\_in\_addr, IP\_ADDR\_LEN);

return arp\_packet;

}

/\* arp请求 \*/

void Arp\_request(const char \*if\_name, const char \*dst\_ip)

{

struct sockaddr\_ll saddr\_ll;

struct ether\_header \*eth\_header;

struct ether\_arp \*arp\_packet;

struct ifreq ifr;

char buf[ETHER\_ARP\_PACKET\_LEN];

unsigned char src\_mac\_addr[ETH\_ALEN];

unsigned char dst\_mac\_addr[ETH\_ALEN] = BROADCAST\_ADDR;

char \*src\_ip;

int sock\_raw\_fd, ret\_len, i;

if ((sock\_raw\_fd = socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ARP))) == -1)

Err\_exit("socket()");

bzero(&saddr\_ll, sizeof(struct sockaddr\_ll));

bzero(&ifr, sizeof(struct ifreq));

/\* 网卡接口名 \*/

memcpy(ifr.ifr\_name, if\_name, strlen(if\_name));

/\* 获取网卡接口索引 \*/

if (ioctl(sock\_raw\_fd, SIOCGIFINDEX, &ifr) == -1)

Err\_exit("ioctl() get ifindex");

saddr\_ll.sll\_ifindex = ifr.ifr\_ifindex;

saddr\_ll.sll\_family = PF\_PACKET;

/\* 获取网卡接口IP \*/

if (ioctl(sock\_raw\_fd, SIOCGIFADDR, &ifr) == -1)

Err\_exit("ioctl() get ip");

src\_ip = inet\_ntoa(((struct sockaddr\_in \*)&(ifr.ifr\_addr))->sin\_addr);

printf("local ip:%s\n", src\_ip);

/\* 获取网卡接口MAC地址 \*/

if (ioctl(sock\_raw\_fd, SIOCGIFHWADDR, &ifr))

Err\_exit("ioctl() get mac");

memcpy(src\_mac\_addr, ifr.ifr\_hwaddr.sa\_data, ETH\_ALEN);

printf("local mac");

for (i = 0; i < ETH\_ALEN; i++)

printf(":%02x", src\_mac\_addr[i]);

printf("\n");

bzero(buf, ETHER\_ARP\_PACKET\_LEN);

/\* 填充以太首部 \*/

eth\_header = (struct ether\_header \*)buf;

memcpy(eth\_header->ether\_shost, src\_mac\_addr, ETH\_ALEN);

memcpy(eth\_header->ether\_dhost, dst\_mac\_addr, ETH\_ALEN);

eth\_header->ether\_type = htons(ETHERTYPE\_ARP);

/\* arp包 \*/

arp\_packet = fill\_arp\_packet(src\_mac\_addr, src\_ip, dst\_ip);

memcpy(buf + ETHER\_HEADER\_LEN, arp\_packet, ETHER\_ARP\_LEN);

/\* 发送请求 \*/

ret\_len = sendto(sock\_raw\_fd, buf, ETHER\_ARP\_PACKET\_LEN, 0, (struct sockaddr \*)&saddr\_ll, sizeof(struct sockaddr\_ll));

if ( ret\_len > 0)

printf("sendto() ok!!!\n");

close(sock\_raw\_fd);

}

int main(int argc, const char \*argv[])

{

if (argc != 3)

{

printf("usage:%s device\_name dst\_ip\n", argv[0]);

exit(1);

}

Arp\_request(argv[1], argv[2]);

return 0;

}

# 6.实现效果图



