路由器转发实验

武庆华

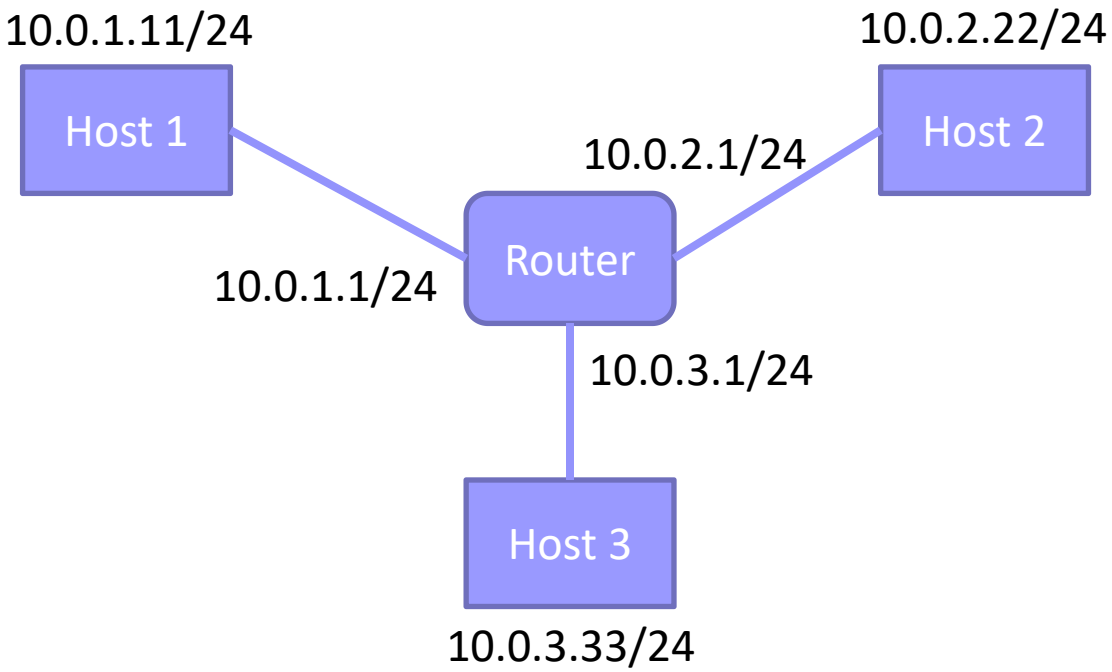
wuqinghua@ict.ac.cn

提纲

- 路由器转发
  - 路由表
  - 路由器转发流程
  - ARP协议与ARP缓存
  - ICMP协议格式
- 实验内容
- 附件文件列表

路由器转发实验

- 给定网络拓扑以及节点的路由表配置，实现路由器的转发功能，使得各节点之间能够连通并传送数据



路由表与最长前缀匹配

- 路由表保存网络（IP+Mask）到网关和端口的映射
- 路由表样例
  - `Dest/Mask -> GW, Iface`
  - `10.0.0.0/8 -> 1.2.0.1, eth0`
  - `10.0.0.0/16 -> 1.3.0.1, eth1`
  - `10.2.0.0/16 -> 1.4.0.1, eth2`
  - `Default -> 1.5.0.1, eth3`
- 在查询路由表时，数据包只包含目的地址
  - 使用最长前缀匹配方法来查找
    - (dst_ip & mask) == (dest & mask)，且掩码长度最长(mask值最大)

自动生成的路由表

- 主机节点

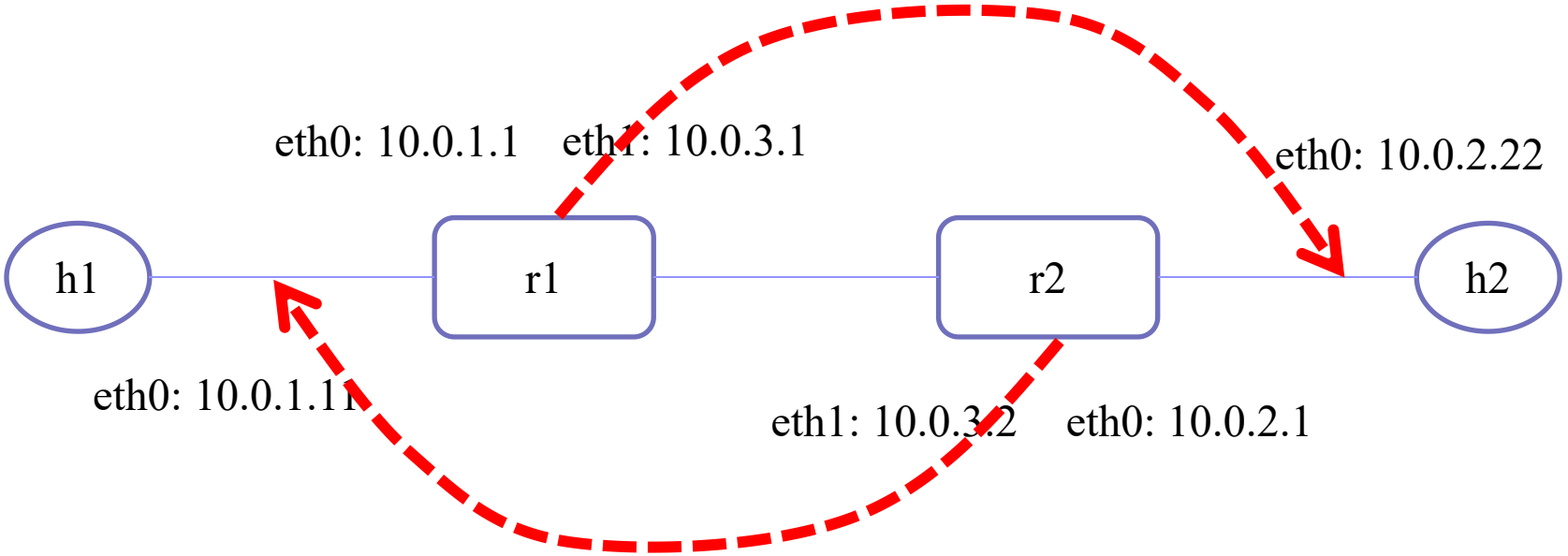
Dest	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.0.1.0	0.0.0.0	255.255.255.0	U	0	0	0	h1-eth0

- 路由器节点

Dest	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.0.1.0	0.0.0.0	255.255.255.0	U	0	0	0	r1-eth0
10.0.2.0	0.0.0.0	255.255.255.0	U	0	0	0	r1-eth1
10.0.3.0	0.0.0.0	255.255.255.0	U	0	0	0	r1-eth2

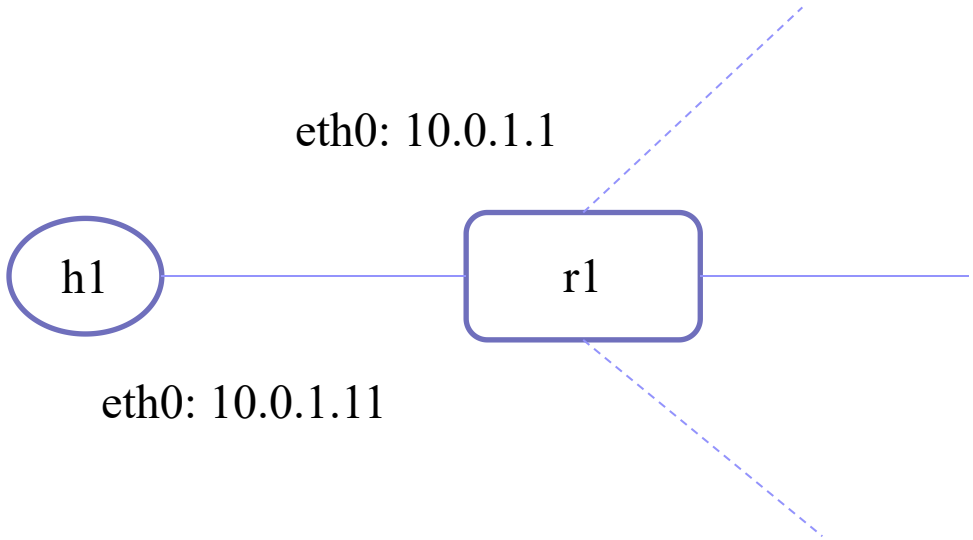
配置到其他网络的路由表

```
# route add -net 10.0.2.0 netmask \
  255.255.255.0 gw 10.0.3.2 dev r1-eth1
```




```
# route add -net 10.0.1.0 netmask \
  255.255.255.0 gw 10.0.3.1 dev r2-eth1
```

配置默认路由表




```
# route add default gw 10.0.1.1 dev h1-eth0
```

默认路由表格式

Dest	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	10.0.1.1	0.0.0.0	UG	0	0	0	h1-eth0

路由表数据结构

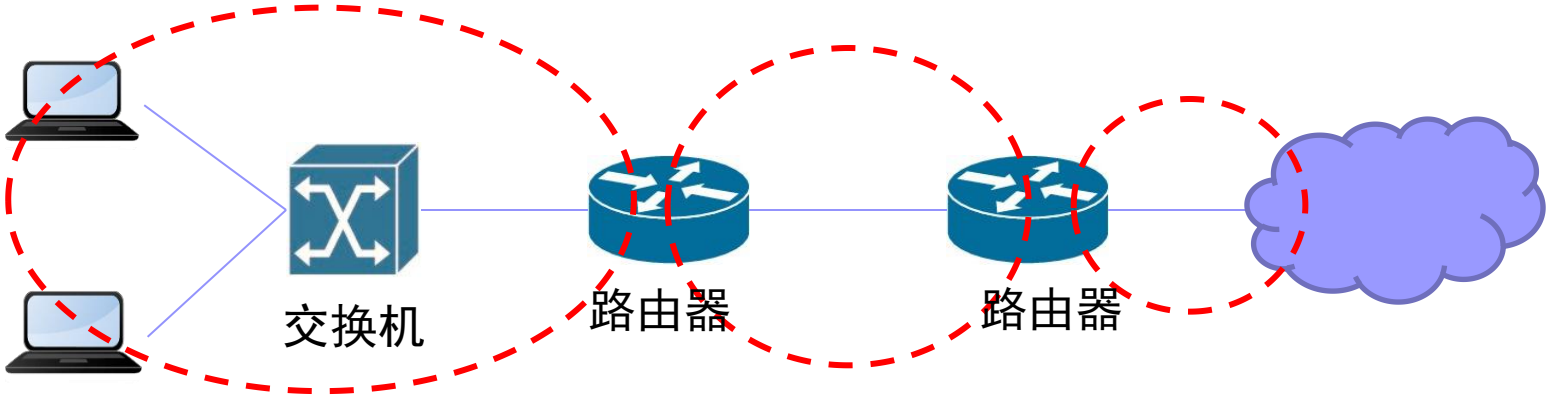
```
struct list_head rtable;    // 路由表，是一个链表头部伪节点


typedef struct {            // 路由表条目
    struct list_head list;  // 链表实现
    u32 dest;               // 目的网络地址
    u32 mask;               // 网络掩码
    u32 gw;                 // 下一跳网关地址
    int flags;              // 转发表条目标识（可忽略）
    char if_name[16];       // 转出端口名字，e.g. r1-eth0
    iface_info_t *iface;    // 转出端口
} rt_entry_t;
```

路由器路由查找流程

- 给定数据包，提取该数据包的目的IP地址
  - 注意进行字节序转换：数据包中的都是网络字节序，本地存储的数据结构都为本地字节序
- 遍历路由表（链表），使用最长前缀匹配查找相应条目
  - 如果设置默认路由，则肯定能查找到匹配路由条目
- 如果查找到相应条目，则将数据包从该条目对应端口转出，否则回复目的网络不可达(ICMP Dest Network Unreachable)

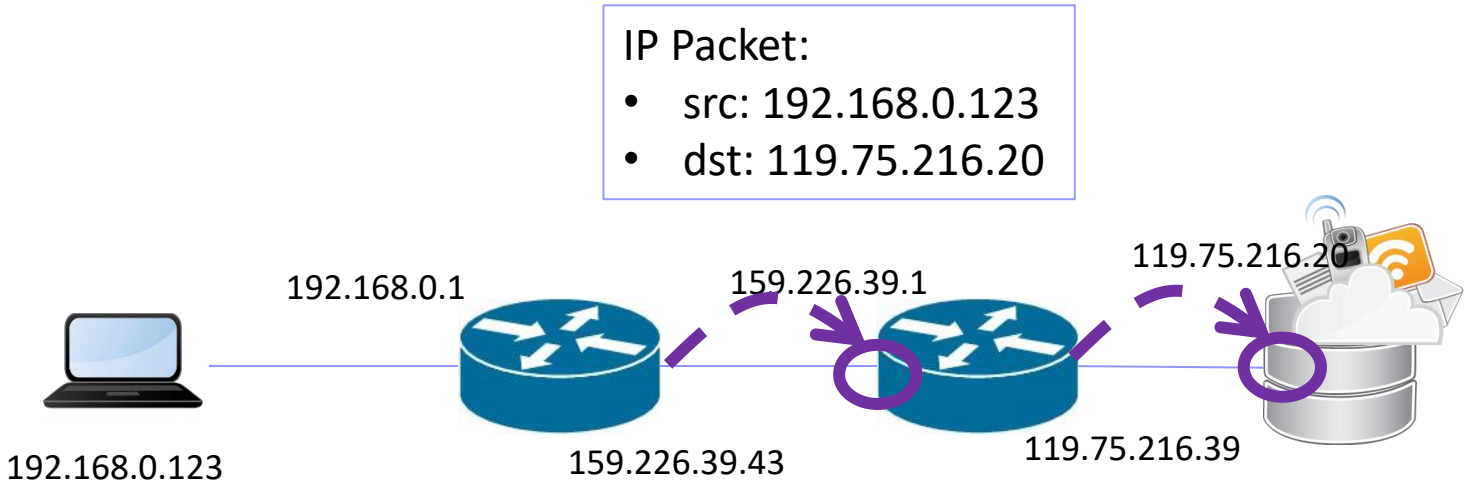
路由器转发数据包流程

- 转发数据包时
  1. 对IP头部的TTL值进行减一操作，如果该值 <= 0，则将该数据包丢弃，并回复ICMP信息
  2. IP头部数据已经发生变化，需要重新设置checksum
  3. 在新的网络内发送数据包
     - 将以太网头部的源MAC地址设置为转发端口的MAC地址
     - 将目的MAC地址设置为对应的MAC地址



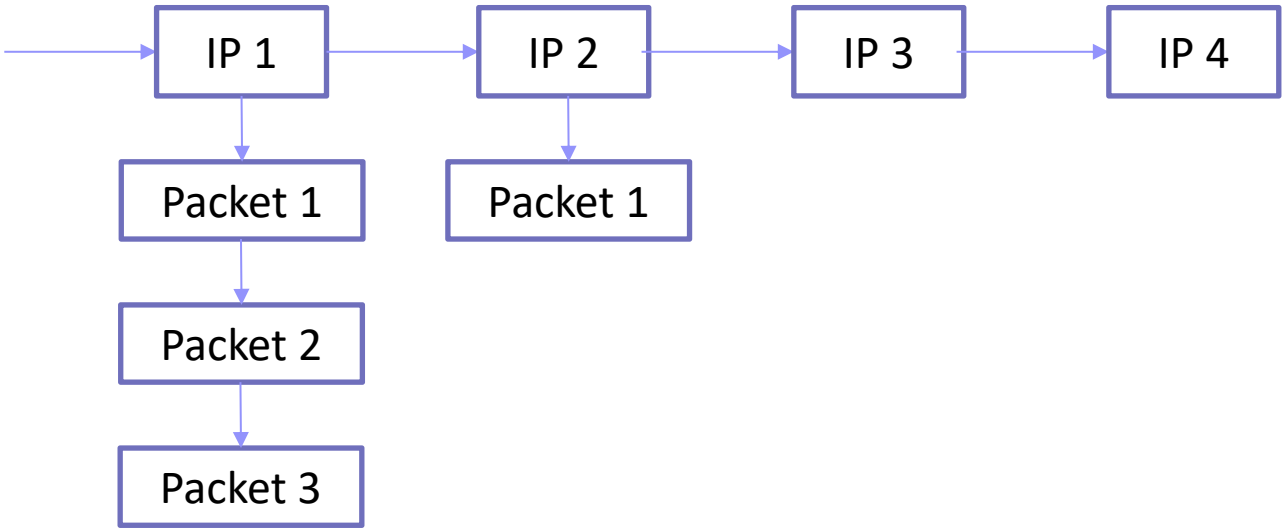
查询IP地址对应的MAC地址

- 给定数据包，路由器查询哪个IP地址对应的MAC地址？
  - MAC地址在局域网内转发数据包时起作用
  - 应查询下一跳IP地址对应的MAC地址
    - 网关地址还是目的主机地址？
    - 当查询到的路由条目中gw为0时，说明数据包已到达目的主机



ARP查询

- 使用ARP机制查询下一跳IP地址对应的MAC地址
  - 路由器维护一个缓存ARP相关内容的数据结构：arpcache
  - arpcache缓存两类数据：
    - IP->MAC映射条目
    - 查找不到相应条目而等待ARP应答的数据包



ARP相关数据结构

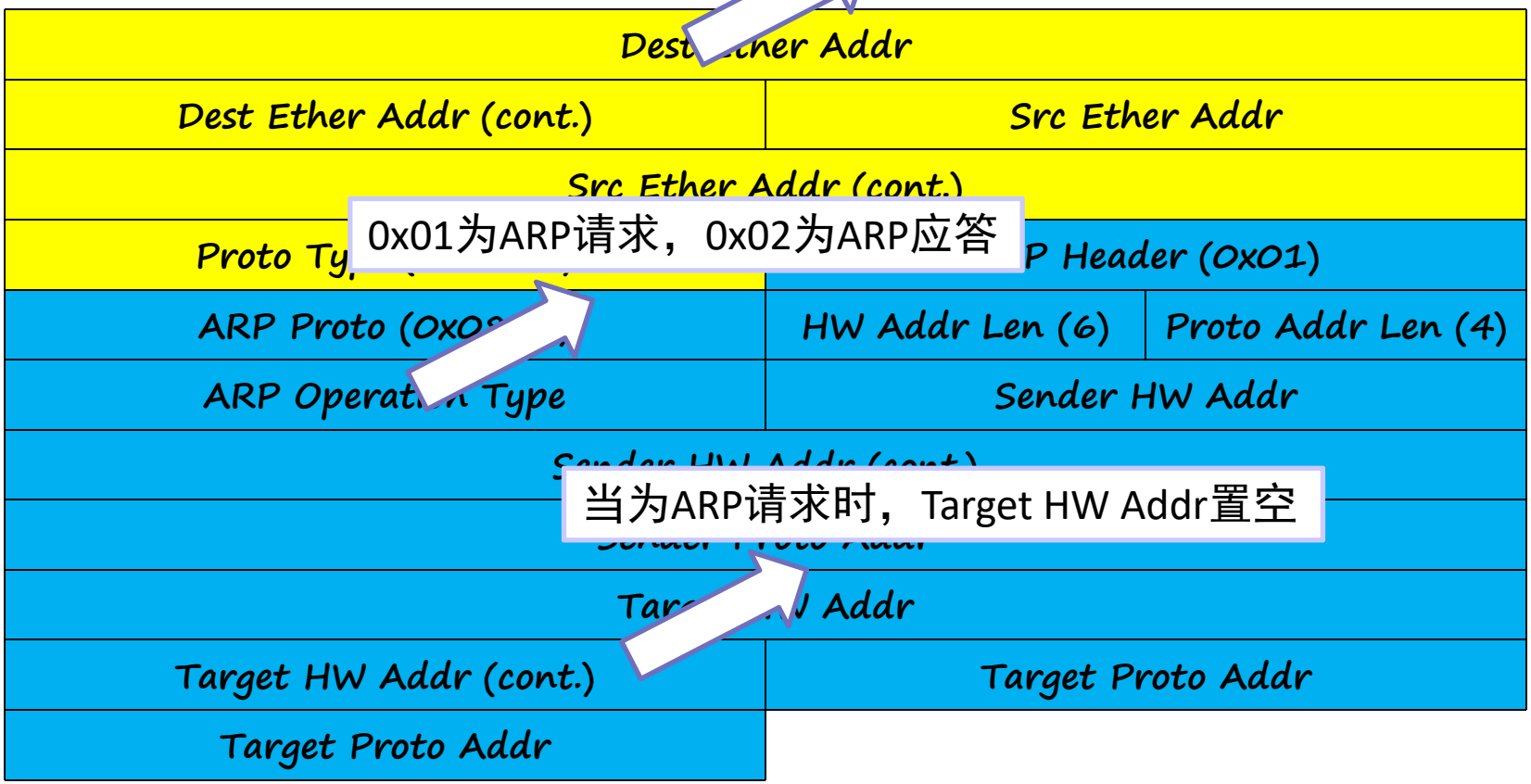
```
typedef struct {
    struct arp_cache_entry entries[32]; // IP->MAC映射，最多存储32条
    struct list_head req_list;          // 等待ARP回复的IP列表，指向arp_req
    pthread_mutex_t lock;               // arpcache查询、更新操作锁
    pthread_t thread;                   // 老化操作对应的线程
} arpcache_t;

struct arp_cache_entry {
    u32 ip4;                    // IP地址，本地字节序
    u8 mac[ETH_ALEN];           // IP地址对应的MAC地址
    time_t added;               // 添加时间
    int valid;                  // 是否继续有效
};

struct arp_req {
    struct list_head list;      // 用于串联不同arp_seq的链表节点
    iface_info_t *iface;        // 转发数据包的端口
    u32 ip4;                    // 等待ARP回复的IP地址
    time_t sent;                        // ARP发送时间
    int retries;                        // 重试次数
    struct list_head cached_packets;  // 目的地址为该IP地址的数据包列表
};
```

ARP缓存操作

- 查找IP->MAC映射
  - 如果在arp缓存中找到相应映射，则填充数据包的目的MAC地址，并转发该数据包
  - 否则，将该数据包缓存在arpcache->req_list中，并发送ARP请求
- 收到新的IP->MAC映射
  - 将该映射写入arp缓存中，如果缓存已满（最多32条），则随机替换掉其中一个
  - 将在缓存中等待该映射的数据包，依次填写目的MAC地址，转发出去，并删除掉相应缓存数据包
- 每1秒钟，运行arpcache_sweep操作
  - 如果一个缓存条目在缓存中已存在超过了15秒，将该条目清除
  - 如果一个IP对应的ARP请求发出去已经超过了1秒，重新发送ARP请求
    - 如果发送超过5次仍未收到ARP应答，则对该队列下的数据包依次回复ICMP（Destination Host Unreachable）消息，并删除等待的数据包

ARP协议格式

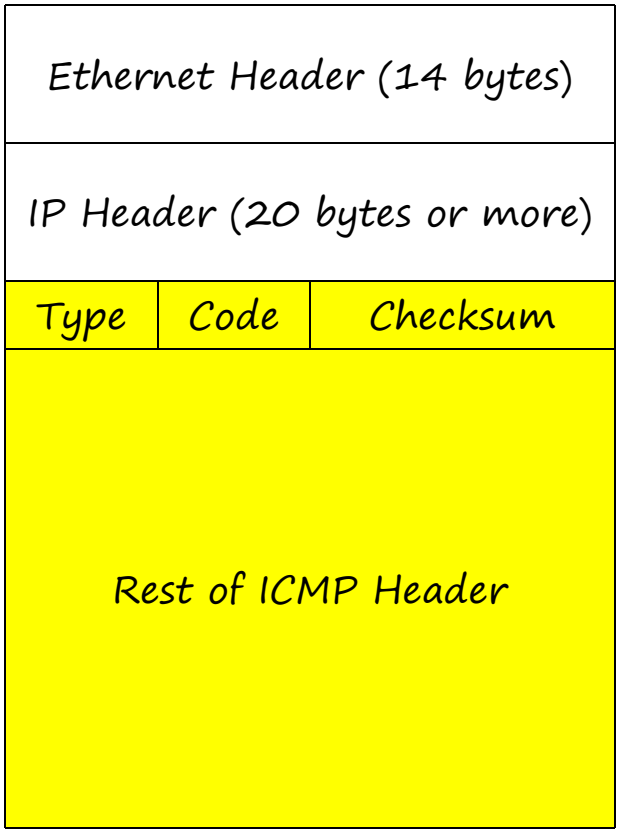
当目的MAC地址不可知时，写FF: FF: FF: FF: FF: FF，为广播包



0x01为ARP请求，0x02为ARP应答

当为ARP请求时，Target HW Addr置空

ICMP数据包格式



路由表查找失败

Type: 3, Code: 0, Rest of ICMP Header: 前4字节设置为0，接着拷贝收到数据包的IP头部（>= 20字节）和随后的8字节

ARP查询失败

Type: 3, Code: 1, Rest of ICMP Header: 同上

TTL值减为0

Type: 11, Code: 0, Rest of ICMP Header:同上

收到Ping本端口的数据包（ Type为8 ）

Type: 0, Code: 0, Rest of ICMP Header: 拷贝Ping包中的相应字段

路由器实现

- 处理ARP请求和应答
  - 收到ARP请求时，如果Target Proto Addr为本端口地址，则ARP应答
  - 转发数据包时，如果ARP缓存中没有相应条目，则发送ARP请求
- ARP缓存管理
  - 进行ARP查询、更新等操作
- IP地址查找和IP数据包转发
  - 收到数据包后，查找对应的转发端口；更新IP头部，转发数据包
- 发送ICMP数据包
  - 路由表查找失败；ARP查询失败； TTL值为0；收到ping本端口的包

实验内容一

- 在主机上安装arptables, iptables，用于禁止每个节点的相应功能
  - sudo apt install arptables iptables
- 运行给定网络拓扑(router_topo.py)
  - 路由器节点r1上执行脚本(disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh)，禁止协议栈的相应功能
  - 终端节点h1-h3上执行脚本disable_offloading.sh

实验内容一（续）

- 在r1上执行路由器程序
  - 在r1中运行./router，进行数据包的处理
- 在h1上进行ping实验
  - Ping 10.0.1.1 (r1)，能够ping通
  - Ping 10.0.2.22 (h2)，能够ping通
  - Ping 10.0.3.33 (h3)，能够ping通
  - Ping 10.0.3.11，返回ICMP Destination Host Unreachable
  - Ping 10.0.4.1，返回ICMP Destination Net Unreachable

实验内容二

- 构造一个包含多个路由器节点组成的网络
  - 手动配置每个路由器节点的路由表
  - 有两个终端节点，通过路由器节点相连，两节点之间的跳数不少于3跳，手动配置其默认路由表
- 连通性测试
  - 终端节点ping每个路由器节点的入端口IP地址，能够ping通
- 路径测试
  - 在一个终端节点上traceroute另一节点，能够正确输出路径上每个节点的IP信息

附件文件列表

- arp.c # 发送ARP请求和应答
- arpcache.c # ARP缓存相关操作
- device_internal.c # 网口管理等内部实现
- icmp.c # ICMP数据包
- ip_base.c # IP前缀查找和发送IP数据包
- rtable.c # 路由表相关
- rtable_internal.c # 从协议栈中读取路由条目
- include
- ip.c # 处理IP数据包，包括转发
- main.c
- Makefile
- router-reference # 参考实现
- router_topo.py # Mininet topo脚本
- scripts # 禁止Linux协议栈的相关功能

libipstack.a