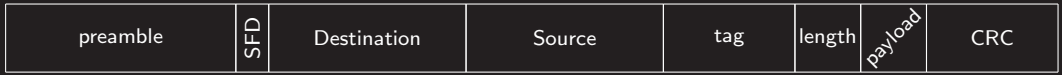
CS356: Low-Level Net. Prog.

W. Michael Petullo

University of Wisconsin–La Crosse

As of October 24, 2021

Firewalls: Ethernet Frames



Preamble (7 bytes) Alternating 0s and 1s; allows for synchronization

Start of frame delimiter (1 byte) Indicates upcoming bits start frame

Destination adderess (6 bytes) MAC address of destination

Source address (6 bytes) MAC address of source

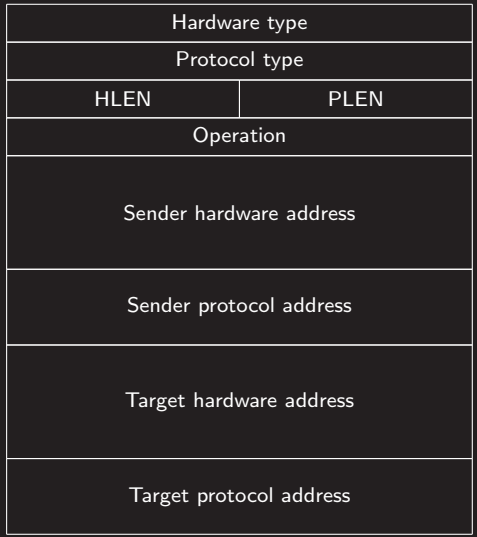
Tag (4 bytes) An optional tag

Length (2 bytes) Length of entire Ethernet frame

Payload (46–1500 bytes) Data carried by frame (e.g., IP packet)

Cyclic redundancy check (4 bytes) Hash of addresses, length, and data

ARP: Format



Hardware type (2 bytes) Ethernet is 1

Protocol type (2 bytes) IPv4 is 0x0800

HLEN (1 byte) Length in bytes of hardware address (6 for Ethernet)

PLEN (1 byte) Length in bytes of protocol address (4 for IPv4)

Operation (2 bytes) Request is 1, reply is 2

SHA (6 bytes) Request: hardware address of sender; reply: "answer" address

SPA (4 bytes) protocol address of sender

THA (6 bytes) Request: ignored; reply: host that made request

TPA (4 bytes) protocol address of receiver

Low-Level Net. Prog.: Headers

Headers necessary for crafting your own Ethernet/ARP messages:

```
#include <stdio.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if_arp.h>
```

Low-Level Net. Prog.: Raw Socket

Create a raw socket that requires programmer to write Ethernet header and so on.

```
int sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
```

Address argument to `connect`/`sendto`:

```
struct sockaddr_ll {
    unsigned short  sll_family;
    __be16          sll_protocol;
    int             sll_ifindex;
    unsigned short  sll_hatype;
    unsigned char   sll_pkttype;
    unsigned char   sll_halen;
    unsigned char   sll_addr[8];
};
```

`PF_FAMILY`	
`htons(ETH_P_ARP)`	
Next slide.	
`ARPHRD_ETHER`	
`PACKET_OTHERHOST`	
0	
Dest. MAC address (bytes 0–5)	

Low-Level Net. Prog.: Interface Index

Identify network interface index suitable for `sll_ifindex`:

```
struct ifreq ifr;
strncpy(ifr.ifr_name, DEVICE, IFNAMSIZ);
ret = ioctl(sock, SIOCGIFINDEX, &ifr);
if (-1 == ret) {
        perror("getting device index failed");
        exit(EXIT_FAILURE);
}
```

Low-Level Net. Prog.: `struct ethhdr`

```
struct ethhdr {
    unsigned char h_dest[ETH_ALEN];   /* destination eth addr */
    unsigned char h_source[ETH_ALEN]; /* source ether addr    */
    __be16        h_proto;            /* packet type ID field */
} __attribute__((packed));
```

Field `h_proto` is `htons(ETH_P_ARP)` for ARP messages.

Low-Level Net. Prog.: struct arphdr

```
struct arphdr {
    __be16          ar_hrd; /* format of hardware address   */
    __be16          ar_pro; /* format of protocol address   */
    unsigned char   ar_hln; /* length of hardware address   */
    unsigned char   ar_pln; /* length of protocol address   */
    __be16          ar_op;  /* ARP opcode (command)         */
};
```

For example:

```
htons(ETH_P_802_3)
htons(ETH_P_IP)
0x06
0x04
htons(ARPOP_REQUEST)
```

Missing:

```
struct arp_payload
{
    unsigned char sender_hardware_address[6];
    unsigned char sender_protocol_address[4];
    unsigned char target_hardware_address[6];
    unsigned char target_protocol_address[4];
};
```

Low-Level Net. Prog.: Assignments

Graded Homework Aquinas: arpspoof

Reading Read 0x440–0x443

https://www.flyn.org/courses/cs356/schedule