Dan Kass
Networking II CE-4960
3/24/14
Lab 2
ARP

Lab Experience:

I thought this lab was interesting to work on. One of the problems I had when combining the receive functionality with the sending functionality was that I was sending the Ethernet packet out and I was receiving the packet information back faster than I had a chance to look for it. I fixed that by setting up all of the receiving functions I need and then sent out the Ethernet packet right before I started looking for the response and this seemed to work. The other interesting part of the lab was how many different IP address you end up with when you move around. With this lab alone I had four different IP address that I had to keep straight because I moved locations so often. Other than it was a fun lab. I used a lot of recourses on line to help me out, but I still feel that I understand how the ARP packet and Ethernet Packet are created. I chose to hard code my MAC address and IP Address while taking the target IP Address as a command line argument. A note on the command line argument is that I do no checking on it, if it is not a correctly formated IP address the program will exit and fail.

Source Code for Lab 2:

```
/*****************************************************************************
 * Name: Dan Kass (kassd@msoe.edu)
 * Date: 3/24/2014
 * Class: Networking II CE-4960
 * Lab 2 Address Resolution Protocol (ARP)
 * Purpose of the this lab is to use raw sockets to create an ARP Ethernet
 *          Packet and receive a response from the the other computer.
 *****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>          // close()
#include <string.h>          // strcpy, memset(), and memcpy()
#include <netdb.h>           // struct addrinfo
#include <sys/types.h>       // needed for socket(), uint8_t, uint16_t
#include <sys/socket.h>      // needed for socket()
#include <netinet/in.h>      // IPPROTO_RAW, INET_ADDRSTRLEN
#include <netinet/ip.h>      // IP_MAXPACKET (which is 65535)
#include <arpa/inet.h>       // inet_pton() and inet_ntop()
#include <sys/ioctl.h>       // macro ioctl is defined
#include <bits/ioctls.h>     // defines values for argument "request" of ioctl.
#include <net/if.h>          // struct ifreq
#include <linux/if_ether.h>  // ETH_P_ARP = 0x0806
#include <linux/if_packet.h> // struct sockaddr_ll (see man 7 packet)
#include <net/ethernet.h>
#include <errno.h>           // errno, perror()


// Define a struct for ARP header
typedef struct _arp_hdr arp_hdr;
struct _arp_hdr {
        uint16_t htype; //Hardware Type Ethernet 0x0001
```

```
        uint16_t ptype; //Protocol Type IPv4 is 0x0800
        uint8_t hlen; //Hardware address length Ethernet is size 6
        uint8_t plen; //Protocol address length IPv4 is size 4
        uint16_t opcode; //Operation: 1 for request, 2 for replies
        uint8_t sender_mac[6]; //My MAC Address
        uint8_t sender_ip[4]; //My IP Address
        uint8_t target_mac[6]; //All 0's this is what we want to find
        uint8_t target_ip[4]; //user input this is the target IP address
};

// Define some constants.
#define ETH_HDRLEN 14      // Ethernet header length
#define IP4_HDRLEN 20      // IPv4 header length
#define ARP_HDRLEN 28      // ARP header length
#define ARPOP_REQUEST 1
#define ARPOP_REPLY 2

// Function prototypes
char *allocate_strmem(int);
uint8_t *allocate_ustrmem(int);

int main(int argc, char **argv) {
        int i, status, frame_length, sd, bytes;
        char *interface, *target, *src_ip;
        arp_hdr arphdr;
        arp_hdr *arphdr2;
        uint8_t *src_mac, *dst_mac, *ether_frame;
        struct addrinfo hints, *res;
        struct sockaddr_in *ipv4;
        struct sockaddr_ll device;

        printf("Finding MAC address for %s \n", argv[1]);

        // Allocate memory for various arrays.
        src_mac = allocate_ustrmem(6);
        dst_mac = allocate_ustrmem(6);
        ether_frame = allocate_ustrmem(IP_MAXPACKET);
        interface = allocate_strmem(40);
        target = allocate_strmem(40);
        src_ip = allocate_strmem(INET_ADDRSTRLEN);
        // Interface to send packet through.
        strcpy(interface, "wlan0");

        // Submit request for a socket descriptor to look up interface.
        if ((sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
        perror("socket() failed to get socket descriptor for using ioctl() ");
        exit(EXIT_FAILURE);
        }
        close(sd);

        // Source IP Address
        //strcpy (src_ip, "155.92.78.114"); //school
        strcpy(src_ip, "10.162.22.119"); //Marquette
        //strcpy (src_ip, "192.168.0.105"); //home
```

```
// Destination IP Address
strcpy(target, argv[1]);

// Source MAC Address
src_mac[0] = 0x8C;
src_mac[1] = 0x70;
src_mac[2] = 0x5A;
src_mac[3] = 0x43;
src_mac[4] = 0xC8;
src_mac[5] = 0x0C;

// Find interface index from interface name and store index in
// struct sockaddr_ll device, which will be used as an argument of sendto().
memset(&device, 0, sizeof(device));
if ((device.sll_ifindex = if_nametoindex(interface)) == 0) {
perror("if_nametoindex() failed to obtain interface index ");
exit(EXIT_FAILURE);
}

// Set destination MAC address: broadcast address
memset(dst_mac, 0xff, 6 * sizeof(uint8_t));

// Fill out hints for getaddrinfo().
memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = hints.ai_flags | AI_CANONNAME;



// Source IP address
if ((status = inet_pton(AF_INET, src_ip, &arphdr.sender_ip)) != 1) {
fprintf(stderr,
        "inet_pton() failed for source IP address.\nError message: %s",
        strerror(status));
exit(EXIT_FAILURE);
}

// Resolve target using getaddrinfo().
if ((status = getaddrinfo(target, NULL, &hints, &res)) != 0) {
fprintf(stderr, "getaddrinfo() failed: %s\n", gai_strerror(status));
exit(EXIT_FAILURE);
}
ipv4 = (struct sockaddr_in *) res->ai_addr;
memcpy(&arphdr.target_ip, &ipv4->sin_addr, 4 * sizeof(uint8_t));
freeaddrinfo(res);

// Fill out sockaddr_ll.
device.sll_family = AF_PACKET;
memcpy(device.sll_addr, src_mac, 6 * sizeof(uint8_t));
device.sll_halen = htons(6);

// ARP header

// Hardware type (16 bits): 1 for ethernet
arphdr.htype = htons(1);
```

```
// Protocol type (16 bits): 2048 for IP
arphdr.ptype = htons(ETH_P_IP);

// Hardware address length (8 bits): 6 bytes for MAC address
arphdr.hlen = 6;

// Protocol address length (8 bits): 4 bytes for IPv4 address
arphdr.plen = 4;

// OpCode: 1 for ARP request
arphdr.opcode = htons(ARPOP_REQUEST);

// Sender hardware address (48 bits): MAC address
memcpy(&arphdr.sender_mac, src_mac, 6 * sizeof(uint8_t));

// Sender protocol address (32 bits)
// See getaddrinfo() resolution of src_ip.

// Target hardware address (48 bits): zero, since we don't know it yet.
memset(&arphdr.target_mac, 0, 6 * sizeof(uint8_t));

// Target protocol address (32 bits)
// See getaddrinfo() resolution of target.

// Fill out ethernet frame header.

// Ethernet frame length = ethernet header (MAC + MAC + ethernet type) +
//ethernet data (ARP header)
frame_length = 6 + 6 + 2 + ARP_HDRLEN;

// Destination and Source MAC addresses
memcpy(ether_frame, dst_mac, 6 * sizeof(uint8_t));
memcpy(ether_frame + 6, src_mac, 6 * sizeof(uint8_t));

// Next is ethernet type code (ETH_P_ARP for ARP).
// http://www.iana.org/assignments/ethernet-numbers
ether_frame[12] = ETH_P_ARP / 256;
ether_frame[13] = ETH_P_ARP % 256;

// Next is ethernet frame data (ARP header).

// ARP header
memcpy(ether_frame + ETH_HDRLEN, &arphdr, ARP_HDRLEN * sizeof(uint8_t));

// Submit request for a raw socket descriptor.
if ((sd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) < 0) {
perror("socket() failed ");
exit(EXIT_FAILURE);
}
```

```
        //now we have to receive the incoming packet information and send it

        // Listen for incoming ethernet frame from socket sd.
        // We expect an ARP ethernet frame of the form:
        //     MAC (6 bytes) + MAC (6 bytes) + ethernet type (2 bytes)
        //     + ethernet data (ARP header) (28 bytes)
        // Keep at it until we get an ARP reply.
        arphdr2 = (arp_hdr *) (ether_frame + 6 + 6 + 2);

        //we need to send the frame right before we look for it because i was
        //getting the response
        // Send ethernet frame to socket.
        if ((bytes = sendto(sd, ether_frame, frame_length, 0,
                        (struct sockaddr *) &device, sizeof(device))) <= 0) {
                perror("sendto() failed");
                exit(EXIT_FAILURE);
        }
        //waiting for the ethernet frame response.
        while (((((ether_frame[12]) << 8) + ether_frame[13]) != ETH_P_ARP)
                || (ntohs(arphdr2->opcode) != ARPOP_REPLY)) {
        if ((status = recv(sd, ether_frame, IP_MAXPACKET, 0)) < 0) {
                if (errno == EINTR) {
                        memset(ether_frame, 0, IP_MAXPACKET * sizeof(uint8_t));
                        continue; // Something weird happened, but let's try again.
                } else {
                        perror("recv() failed:");
                        exit(EXIT_FAILURE);
                }
        }
        }
        close(sd);

        //printing out the response and its MAC address
        printf("%s's MAC address is: ", argv[1]);
        for (i = 0; i < 4; i++) {
        printf("%02x:", arphdr2->sender_mac[i]);
        }
        printf("%02x\n", arphdr2->sender_mac[5]);

        // Free allocated memory.
        free(src_mac);
        free(dst_mac);
        free(ether_frame);
        free(interface);
        free(target);
        free(src_ip);

        return (EXIT_SUCCESS);

}

// Allocate memory for an array of chars.
char * allocate_strmem(int len) {
        void *tmp;
```

```
        if (len <= 0) {
        fprintf(
                        stderr,
                        "ERROR: Cannot allocate memory because len = %i in allocate_strmem().\n",len);
        exit(EXIT_FAILURE);
        }

        tmp = (char *) malloc(len * sizeof(char));
        if (tmp != NULL) {
        memset(tmp, 0, len * sizeof(char));
        return (tmp);
        } else {
        fprintf(stderr,
                        "ERROR: Cannot allocate memory for array allocate_strmem().\n");
        exit(EXIT_FAILURE);
        }
}

// Allocate memory for an array of unsigned chars.
uint8_t * allocate_ustrmem(int len) {
        void *tmp;

        if (len <= 0) {
        fprintf(
                        stderr,
                        "ERROR: Cannot allocate memory because len = %i in allocate_ustrmem().\n",len);
        exit(EXIT_FAILURE);
        }

        tmp = (uint8_t *) malloc(len * sizeof(uint8_t));
        if (tmp != NULL) {
        memset(tmp, 0, len * sizeof(uint8_t));
        return (tmp);
        } else {
        fprintf(
                        stderr,
                        "ERROR: Cannot allocate memory for array allocate_ustrmem().\n");
        exit(EXIT_FAILURE);
        }
}
```

Recourses:

Wikipedia's article on the Address Resolution Protocol
http://en.wikipedia.org/wiki/Address_Resolution_Protocol
Raw Ethernet Programming
http://aschauf.landshut.org/fh/linux/udp_vs_raw/ch01s03.html
C Language Examples of IPv4 and IPv6 Raw Sockets for Linux
http://www.pdbuchan.com/rawsock/rawsock.html