

```

/* -----
(c) Riduan M ABID, eMail: R.Abid@aui.ma
This code was implemented as a partial requirement for
a Ph.D degree in Computer Science at Auburn University
Supervisor: Dr. Biaz - eMail: biazsaa@auburn.edu,
Shelby Center for Engineering Technology- Auburn Unveristy - 2009,
----- */
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if.h>
#include <netinet/in.h>
#include <asm/types.h>
#include <sys/ioctl.h>
#include <linux/ip.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define MPP_COUNT 5
#define KEY 777777
#define SHARED_MEMORY_SIZE sizeof(struct MAP_Proxying_Table)
#define BROADCAST_ADDR {0xff, 0xff, 0xff, 0xff, 0xff, 0xff}

struct _802_11_s_frame_header fill_header(__u8 * dest, __u8 * src, __u8 *
dest_proxy, __u8 * src_proxy, __u8 * final_dest, __u8 * originator, __u8 * buffer);
void print_mac_addr(__u8 * mac);
int comp_mac_addr(__u8 * mac1, __u8 * mac2);
void print_frame_hdr(__u8 * hdr);

struct _802_11_s_frame_header {
    __u8 dest[6];
    __u8 src[6];
    __u8 type[2];
    __u8 eltID;
    __u8 dest_proxy[6];
    __u8 src_proxy[6];
    __u8 final_dest[6];
    __u8 originator[6];
};
struct MPP_Proxy_Entry {
    __u8 MPP_mac_addr[6];
    __u8 next_hop[6];
    __u32 seq_no;
    __u32 timeStamp;
    __u8 hop_count;
    double metric;
};

```

```

struct MAP_Proxying_Table {
    struct MPP_Proxy_Entry MPP_Entries[MPP_COUNT];
    int active_MPP; // subscript in the Array,
    int visible_MPPs;
};
__u8 LAN_ifr_mac_addr[6], Adhoc_ifr_mac_addr[6];
__u8 IP_packet[4096],src_mac_addr[6], dest_mac_addr[6] = BROADCAST_ADDR,
dest_mac[6], next_hop[6], temp_mac[6] = {0x0, 0x19, 0x7e, 0x46, 0x67, 0xcf},
client_addr[6]={0x00, 0x19, 0x7E, 0x56, 0x23, 0x49}, final_dest_mac[6];

int main(int argc, char* argv[]) {
    __u8 buffer[4096], buffer2[4096], eltID, *shm;
    struct sockaddr dest;
    int shmid, recievedBytes, dlen = 4096, cnt = 0;
    int LAN_sock, adhoc_sock, open_sock, fd, i, frame_size = sizeof(struct
_802_11_s_frame_header);
    key_t key;
    struct ifreq ifr, ifr2;
    struct sockaddr_ll sll0, sll;
    struct _802_11_s_frame_header frame_hdr;
    struct MAP_Proxying_Table MAP_proxying_table;
    struct MPP_Proxy_Entry MPP_Entry;

    // Checking for Arguments,
    if (argc < 3) {
        perror("\n Inussufficient Arguments ");
        perror("\n Usage: <executable> <LAN-ifr_Name> <Ad-Hoc-ifr_Name> \n");
        exit(-1);
    }
    //
-----
    // - Getting LAN-ifr MAC Address,
    fd = socket(AF_INET, SOCK_DGRAM, 0);
    ifr.ifr_addr.sa_family = AF_INET;
    strncpy(ifr.ifr_name, argv[1], IFNAMSIZ-1);
    ioctl(fd, SIOCGIFHWADDR, &ifr);
    close(fd);
    memcpy(LAN_ifr_mac_addr, ifr.ifr_hwaddr.sa_data, 6);
    // Finding LAN-ifr Index,
    if ((LAN_sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) < 0 ) {
        perror("socket");
        exit(-1);
    }
    memset(&ifr, 0, sizeof(ifr));
    strcpy(ifr.ifr_name, argv[1]);
    if ( ioctl(LAN_sock, SIOCGIFINDEX, &ifr) < 0) {
        perror("ioctl (SIOCGIFINDEX) ");
        exit(-1);
    }
    // Bind "sll0" to the LAN ifr,
    memset(&sll0, 0, sizeof(sll0));
    sll0.sll_family = AF_PACKET;
    sll0.sll_protocol = htons(ETH_P_ALL);

```

```

sll0.sll_ifindex = ifr.ifr_ifindex;
if (bind(LAN_sock, (struct sockaddr *)&sll0, sizeof(sll0)) != 0) {
    perror("Error Binding LAN_sock");
    exit(-1);
}
//
-----
// - Getting Ad-Hoc-ifr MAC Address,
fd = socket(AF_INET, SOCK_DGRAM, 0);
ifr2.ifr_addr.sa_family = AF_INET;
strncpy(ifr2.ifr_name, argv[2], IFNAMSIZ);
ioctl(fd, SIOCGIFHWADDR, &ifr2);
close(fd);
memcpy(Adhoc_ifr_mac_addr, ifr2.ifr_hwaddr.sa_data, 6);
// Finding Ad-Hoc-ifr Index,
if ((adhoc_sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) < 0 ) {
    perror("socket error");
    exit(-1);
}
memset(&ifr2, 0, sizeof(ifr2));
strncpy(ifr2.ifr_name, argv[2], IFNAMSIZ);
if ( ioctl(adhoc_sock, SIOCGIFINDEX, &ifr2) < 0) {
    perror("ioctl (SIOCGIFINDEX) ");
    exit(-1);
}
// Bind "sll" to the wireless ifr,
memset(&sll, 0, sizeof(sll));
sll.sll_family = AF_PACKET;
sll.sll_protocol = htons(ETH_P_ALL);
sll.sll_ifindex = ifr2.ifr_ifindex;
if (bind(adhoc_sock, (struct sockaddr *)&sll, sizeof(sll)) != 0) {
    perror("Error Binding LAN_sock");
    exit(-1);
}

// open_sock - Recieving both LAN and adhoc frames,
if ((open_sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) < 0 ) {
    perror("socket error");
    exit(-1);
}

// Locating the shared memory for MPP_Proxying_Table,
key = KEY;
if ((shmid = shmget(key, SHARED_MEMORY_SIZE, IPC_CREAT)) < 0) {
    perror("shmget");
    exit(1);
}

// ----- Frame Processing, -----
while(1) {
    recievedBytes = recvfrom(open_sock,buffer,4096,0,&dest,&dlen);
    if (recievedBytes > 2100) {
        printf("\n ===== CHECK! LONG Packet ===== ");
        continue;
    }
}

```

```

    }
    memcpy(&eltID, buffer+12, 1);
    if (eltID == 13 || eltID == 5 || eltID == 77 || eltID == 11 || eltID ==
22 || eltID == 44 || eltID == 55) // a Routing Frame,
        continue;
    memcpy(dest_mac, buffer, 6);
    memcpy(src_mac_addr, buffer+6, 6);
    if (comp_mac_addr(src_mac_addr, LAN_ifr_mac_addr) ||
comp_mac_addr(src_mac_addr,
                Adhoc_ifr_mac_addr))
        continue; // Discarding Local messages,

    // --- Feedback,
    //printf("\n *** %d ---> Received %d bytes FROM:", cnt++,
recievedBytes);
    //print_mac_addr(src_mac_addr);

    // Checking Frame eltID,
    memcpy(&eltID, buffer+14, 1);
    if (eltID == 27) { // an Ad-hoc Frame,
        if (!comp_mac_addr(Adhoc_ifr_mac_addr, dest_mac))
            continue;
        printf("\n ----- Ad-hoc Packet ----- ");
        // Getting Frame Header,
        memcpy(&frame_hdr, buffer, frame_size);
        //print_frame_hdr((__u8 *)&frame_hdr);

        // Getting IP_Packet,
        memcpy(IP_packet, buffer+frame_size, recievedBytes-frame_size);

        // Formulating the Outgoing Frame,
        // Getting Final_Dest_MAC,
        memcpy(final_dest_mac, frame_hdr.final_dest, 6);

        /* Set the MAC address of the destination */
        memcpy(sll0.sll_addr, final_dest_mac, 6); // Broadcast Addr -
Broadcast the Frame in the LAN,
        //memcpy(sll0.sll_addr, dest_mac_addr, 6);
        sll0.sll_addr[6] = 0x00;
        sll0.sll_addr[7] = 0x00;
        /* Set the destination and source mac_addresses */
        memcpy(buffer2, final_dest_mac, 6);
        memcpy(buffer2+6, LAN_ifr_mac_addr, 6);
        buffer2[12] = buffer[12];
        buffer2[13] = buffer[13];

        // Constructing the Frame,
        memcpy(buffer2+14, IP_packet, recievedBytes - frame_size);
        if ((i = sendto(LAN_sock, buffer2, recievedBytes - frame_size +
14, 0x00, // To change to -2,
                (struct sockaddr *)&sll0, sizeof(sll0))) < 0 ) {
            perror("sendto");
            exit(-1);
        }
        printf("-- sent --> TO Wifi_LAN ", i);

```

```

    } else { // an 802.11 LAN Frame,
        if (!comp_mac_addr(src_mac_addr, client_addr)) // For Security
Reasons,
            continue;
        printf("\n --- a Wifi_Frame");
        memcpy(IP_packet, buffer+14, recievedBytes-14);
        // Getting next_hop from Shared Memory,
        if ((shm = shmat(shmid, NULL, 0)) == NULL) {
            perror("shmat");
            exit(1);
        }
        // Getting the MAP Proxying Table,
        memcpy(&MAP_proxying_table, shm, SHARED_MEMORY_SIZE); // ONE Entry
per MPP,
        // Getting the best MPP Entry,
        MPP_Entry =
MAP_proxying_table.MPP_Entries[MAP_proxying_table.active_MPP];
        printf(" --- NEXT HOP --> "); print_mac_addr(MPP_Entry.next_hop);

        // Setting the 802_11.s Frame header,
        frame_hdr = fill_header(MPP_Entry.next_hop, Adhoc_ifr_mac_addr,
MPP_Entry.MPP_mac_addr, Adhoc_ifr_mac_addr, dest_mac, src_mac_addr, buffer);
        memcpy(buffer2, &frame_hdr, frame_size);
        //print_frame_hdr((__u8 *) &frame_hdr);

        /* Set the MAC address of the destination */
        memcpy(sll.sll_addr, MPP_Entry.next_hop, 6);
        sll.sll_addr[6] = 0x00;
        sll.sll_addr[7] = 0x00;

        // Setting IP_packet
        memcpy(buffer2+frame_size, IP_packet, recievedBytes-14);
        // Sending the Frame,
        if ((i = sendto(adhoc_sock, buffer2, recievedBytes + frame_size -
14, 0x00, // To change to -2,
            (struct sockaddr *)&sll, sizeof(sll))) < 0 ) {
            perror("sendto");
            exit(-1);
        }
        printf("---%d BYTES--- sent",i);
//print_mac_addr(MPP_Entry.next_hop);
    }
}

struct _802_11_s_frame_header fill_header(__u8 * dest, __u8 * src, __u8 *
dest_proxy, __u8 * src_proxy, __u8 * final_dest, __u8 * originator, __u8 * buffer) {
    struct _802_11_s_frame_header hdr;
    memcpy(hdr.dest, dest, 6);
    memcpy(hdr.src, src, 6);
    memcpy(hdr.type, buffer+12, 2);
    hdr.elitID = 27;
    memcpy(hdr.dest_proxy, dest_proxy, 6);

```

```

        memcpy(hdr.src_proxy, src_proxy, 6);
        memcpy(hdr.final_dest, final_dest, 6);
        memcpy(hdr.originator, originator, 6);
        return hdr;
    }
void print_mac_addr(__u8 * mac) {
    int i;
    for (i = 0; i < 6; i++)
        printf("%2x:", mac[i]);
}
int comp_mac_addr(__u8 * mac1, __u8 * mac2) {
    int i;
    for (i = 0; i < 6; i++)
        if (mac1[i] != mac2[i])
            return 0;
    return 1;
}
void print_frame_hdr(__u8 * hdr) {
    printf("\n Dest: ");
    print_mac_addr(hdr);
    printf("\n Src: ");
    print_mac_addr(hdr+6);
    printf("\n Dest_Proxy: ");
    print_mac_addr(hdr+15);
    printf("\n Src_Proxy: ");
    print_mac_addr(hdr+21);
    printf("\n final_dest: ");
    print_mac_addr(hdr+27);
    printf("\n originator: ");
    print_mac_addr(hdr+33);
}

```