

```

#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 <sys/ioctl.h>
#include <asm/types.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define MPP_COUNT 5
#define MAP_COUNT 5
#define SHARED_MEMORY_SIZE  sizeof(struct MP_Forwarding_Table)

struct Proxy_Entry {
    __u8 mac_addr[6];
    __u8 next_hop[6];
    __u32 seq_no;
    __u8 hop_count;
    double metric;
};

struct MP_Forwarding_Table {
    struct Proxy_Entry MPP_Entries[MPP_COUNT];
    struct Proxy_Entry MAP_Entries[MAP_COUNT];
    int visible_MPPs;
    int visible_MAPs;
};

struct Proxy_Entry get_Proxy_Entry(struct MP_Forwarding_Table table, __u8 *
mac_addr);
void print_MP_Forwarding_Table(struct MP_Forwarding_Table table);
void print_mac_addr(__u8 * mac);
int comp_mac_addr(__u8 * mac1, __u8 * mac2);
void print_frame_hdr(__u8 * hdr);

__u8 Adhoc_ifr_mac_addr[6], src_mac_addr[6], dest_mac_addr[6], dest_mac[6],
dest_proxy[6], next_hop[6];

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];
};

int main(int argc, char* argv[]) {
    __u8 buffer[4096], buffer2[4096], IP_packet[4096], *shm;

```

```

    struct sockaddr dest, name;
    int shmid_MP_Table, dlen, recievedBytes, len, frame_size = sizeof(struct
_802_11_s_frame_header), eltID;
    int raw_sock, fd, i;
    long int cnt = 0;
    key_t key_MP_Table;
    struct ifreq ifr, ifr2;
    struct sockaddr_ll sll;
    struct _802_11_s_frame_header frame_hdr;
    struct MP_Forwarding_Table MP_forwarding_table;
    struct Proxy_Entry proxy_entry;

// Checking for Arguments,
if (argc < 2) {
    perror("\n Inussuficient Arguments ");
    perror("\n Usage: <executable> <Ad-Hoc-ifr_Name> \n");
    exit(-1);
}
// - Getting Adhoc-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(Adhoc_ifr_mac_addr, ifr.ifr_hwaddr.sa_data, 6);
// Finding Adhoc-ifr Index,
if ((raw_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(raw_sock, SIOCGIFINDEX, &ifr) < 0) {
    perror("ioctl (SIOCGIFINDEX) ");
    exit(-1);
}
// Bind "sll" to the LAN ifr,
memset(&sll, 0, sizeof(sll));
sll.sll_family = AF_PACKET;
sll.sll_protocol = htons(ETH_P_ALL);
sll.sll_ifindex = ifr.ifr_ifindex;
if (bind(raw_sock, (struct sockaddr *)&sll, sizeof(sll)) != 0) {
    perror("Error Binding Raw_Sock");
    exit(-1);
}

// Locating Shared Memories for MP_Forwarding_Table,
key_MP_Table = 131305;

if ((shmid_MP_Table = shmget(key_MP_Table, SHARED_MEMORY_SIZE, IPC_CREAT)) <
0) {
    perror("shmget");
    exit(1);
}

```

```

}
if ((shm = shmat(shmid_MP_Table, NULL, 0)) == NULL) {
    perror("shmat");
    exit(1);
}

while(1) {
    recievedBytes = recvfrom(raw_sock,buffer,4096,0,&dest,&dlen);
    memcpy(dest_mac, buffer, 6);
    memcpy(src_mac_addr, buffer+6, 6);
    if (!comp_mac_addr(dest_mac, Adhoc_ifr_mac_addr)) {
        continue; // Discarding messages not meant for this station,
    }

    memcpy(&eltID, buffer+12, 1);
    if (eltID == 13 || eltID == 5 || eltID == 77 || eltID == 11 || eltID ==
22 || eltID == 44 || eltID == 55) // a Routing Frame - Discard,
        continue;
    memcpy(&eltID, buffer+14, 1);
    if (eltID != 27) // Not an ad-hoc frame,
        continue;
    if (comp_mac_addr(src_mac_addr, Adhoc_ifr_mac_addr))
        continue; // Discarding Local messages,
    printf("\n - %d - from:", cnt++);
    print_mac_addr(src_mac_addr);

    // Getting MP_Forwarding Table,
    memcpy(&MP_forwarding_table, shm, SHARED_MEMORY_SIZE);
    //print_MP_Forwarding_Table(MP_forwarding_table);

    // Getting Frame Header,
    memcpy(&frame_hdr, buffer, frame_size);
    // Getting Destintation Proxy Address (to_MAP OR to_MPP?)
    memcpy(dest_proxy, frame_hdr.dest_proxy, 6);

    // Getting Destination Proxy Entry in MP_Forwarding Table,
    proxy_entry = get_Proxy_Entry(MP_forwarding_table, dest_proxy);
    if (proxy_entry.seq_no == 0)
        continue; // NULL_entry;

    // Updating the new 802.11s header,
    memcpy(frame_hdr.dest, proxy_entry.next_hop, 6);
    memcpy(frame_hdr.src, Adhoc_ifr_mac_addr, 6);
    // Getting IP_Packet,
    memcpy(&IP_packet, buffer+frame_size, recievedBytes-frame_size);
    /* Set the MAC address of source and destination */
    memcpy(sll.sll_addr, proxy_entry.next_hop, 6);
    sll.sll_addr[6] = 0x00;
    sll.sll_addr[7] = 0x00;
    memcpy(buffer2, &frame_hdr, frame_size);
    memcpy(buffer2+frame_size, &IP_packet, recievedBytes - frame_size);
    // Ya RBI,
    if ((i = sendto(raw_sock, buffer2, recievedBytes, 0x00, // To change to

```

```

        (struct sockaddr *)&sll, sizeof(sll))) < 0 ) {
            perror("sendto");
            exit(-1);
        }
        printf(" * to ---> ", i); print_mac_addr(proxy_entry.next_hop);
    }
}
struct Proxy_Entry get_Proxy_Entry(struct MP_Forwarding_Table table, __u8 *
mac_addr) {
    int i;
    struct Proxy_Entry NULL_entry;
    NULL_entry.seq_no = 0;

    for (i = 0; i < table.visible_MPPs; i++)
        if (comp_mac_addr(table.MPP_Entries[i].mac_addr, mac_addr))
            return table.MPP_Entries[i];
    for (i = 0; i < table.visible_MAPs; i++)
        if (comp_mac_addr(table.MAP_Entries[i].mac_addr, mac_addr))
            return table.MAP_Entries[i];
    printf("\n ERROR - No Entry found in MP Forwarding Table ");
    return NULL_entry;
}

void print_MP_Forwarding_Table(struct MP_Forwarding_Table table) {
    int i;

    printf("\n ----- MP Forwarding Table ----- ");
    printf("\n - Visible MPPs: %d", table.visible_MPPs);
    printf("\n - Visible MAPs: %d", table.visible_MAPs);
    printf("\n ----- MPP Entries ----- ");
    for (i = 0; i < table.visible_MPPs; i++) {
        printf("\n ----- MPP: %d -----", i);
        printf("\n ----- MAC addr: ");
        print_mac_addr(table.MPP_Entries[i].mac_addr);
        printf("\n ----- Next hop: ");
        print_mac_addr(table.MPP_Entries[i].next_hop);
        printf("\n ----- Hop count: %d", table.MPP_Entries[i].hop_count);
        printf("\n ----- Metric: %.2f", table.MPP_Entries[i].metric);
    }
    printf("\n ----- MAP Entries ----- ");
    for (i = 0; i < table.visible_MAPs; i++) {
        printf("\n ----- MAP: %d -----", i);
        printf("\n ----- MAC addr: ");
        print_mac_addr(table.MAP_Entries[i].mac_addr);
        printf("\n ----- Next hop: ");
        print_mac_addr(table.MAP_Entries[i].next_hop);
        printf("\n ----- Hop count: %d", table.MAP_Entries[i].hop_count);
        printf("\n ----- Metric: %.2f", table.MAP_Entries[i].metric);
    }
}

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+12);
    printf("\n Src_Proxy: ");
    print_mac_addr(hdr+18);
    printf("\n Final_Dest: ");
    print_mac_addr(hdr+24);
    printf("\n Originator: ");
    print_mac_addr(hdr+30);
}
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;
}
```