

```
/* -----  
    (c) Riduan M. ABID - eMail: R.Abid@aui.ma, abidmoh@auburn.edu,  
    This Work is relevant to a Ph.D Dissertation,  
    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/netlink.h>  
#include <linux/rtnetlink.h>  
#include <linux/ip.h>  
#include <string.h>  
#include <sys/ipc.h>  
#include <sys/shm.h>  
  
#define MP_COUNT 10  
#define SHARED_MEMORY_SIZE MP_COUNT * sizeof(struct Received_Probe)  
#define ETX_SHARED_MEMORY_SIZE MP_COUNT * sizeof(struct ETX)  
#define KEY 55555  
#define KEY_ETX 130508  
#define AVERAGING_PERIOD 10  
  
struct Received_Probe { // Received Probe Frames Per MP,  
    __u8 mac_addr[6];  
    __u8 value;  
};  
struct ETX {  
    __u8 mac_addr[6];  
    double df, dr, etx;  
};  
int get_received_probes_for_MP(struct Received_Probe * Ri, __u8 * mac_addr);  
void print_Received_Probes (struct Received_Probe *Ri);  
void print_ETXi (struct ETX *ETXi);  
void print_mac_addr(__u8 * mac) ;  
int comp_mac_addr(__u8 * mac1, __u8 * mac2);  
void print_mac_addr(__u8 * mac);  
  
int main(int argc, char* argv[]) {  
    double df, dr;  
    __u8 probe_frame[4096], *shm, *shm_ETX;  
    struct sockaddr dest;  
    int shmid, shmid_ETX, visible_MPs = 0, dlen, len, c_time, sec;  
    int raw_sock, fd, i, j, n, cnt = 0, mp_count, eltID, cursor;  
    key_t key, key_ETX;  
    struct ifreq ifr;
```

```

struct sockaddr_ll sll;
__u8 src_mac_addr[6], dest_mac[6], local_mac_addr[6], null_addr[6] =
{0x00,0x00,0x00,0x00,0x00,0x00};

struct Received_Probe Ri[MP_COUNT], Ri_in_Probe[MP_COUNT];
struct ETX ETXi[MP_COUNT];

// Checking for Arguments,
if (argc < 2) {
    perror("\n Inussufficient Arguments ");
    perror("\n Usage: <executable> <Ad-Hoc-ifr_Name> \n");
    exit(-1);
}

// - Getting Ad-Hoc-ifr MAC Address,
fd = socket(AF_INET, SOCK_DGRAM, 0);
ifr.ifr_addr.sa_family = AF_INET;
strncpy(ifr.ifr_name, argv[1], IFNAMSIZ);
ioctl(fd, SIOCGIFHWADDR, &ifr);
close(fd);
memcpy(local_mac_addr, ifr.ifr_hwaddr.sa_data, 6);

// - Getting Ad-Hoc-ifr Index,
if ((raw_sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) < 0 ) {
    perror("socket error");
    exit(-1);
}
memset(&ifr, 0, sizeof(ifr));
strncpy(ifr.ifr_name, argv[1], IFNAMSIZ);
if ( ioctl(raw_sock, SIOCGIFINDEX, &ifr) < 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 = ifr.ifr_ifindex;

if (bind(raw_sock, (struct sockaddr *)&sll, sizeof(sll)) != 0) {
    perror("Error Binding Raw_Sock");
    exit(-1);
}

// Creating Ri and ETXi entries Shared Memory,
key = KEY;
key_ETX = KEY_ETX;

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

```

```

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

if ((shmid_ETX = shmget(key_ETX, ETX_SHARED_MEMORY_SIZE, IPC_CREAT)) < 0) {
    perror("shmget");
    exit(1);
}

if ((shm_ETX = shmat(shmid_ETX, NULL, 0)) == NULL) {
    perror("shmat");
    exit(1);
}

// Initializing Ri - Received Probes,
for (i = 0; i < MP_COUNT; i++) {
    Ri[i].value = 0;
    memcpy(&Ri[i].mac_addr, null_addr, 6);
}
// Writing Ri to shared memory - Essential to etx_broadcast,
memcpy(shm, Ri, SHARED_MEMORY_SIZE);

// Get Current Time,
c_time = time(NULL);

while(1) {
    i = recvfrom(raw_sock, probe_frame, 4096, 0, &dest, &dlen);
    // Checking if it is an ETX control frame,
    memcpy(&eltID, probe_frame+12, 1);
    if (eltID != 77)
        continue; // Discard the frame,

    memcpy(dest_mac, probe_frame, 6);
    memcpy(src_mac_addr, probe_frame+6, 6);
    if (comp_mac_addr(src_mac_addr, local_mac_addr))
        continue; // Discard Local messages,

    // ----- Updating Corresponding Ri[i],
    i = get_received_probes_for_MP(Ri, src_mac_addr);
    if (i == -1 && visible_MPs < MP_COUNT) { // No Entry,
        memcpy(Ri[visible_MPs].mac_addr, src_mac_addr, 6); // Fill Entry,
        Ri[visible_MPs].value++;
        memcpy(ETXi[visible_MPs].mac_addr, src_mac_addr, 6); // Fill ETXi
Entry,
        ETXi[visible_MPs].df = ETXi[visible_MPs].dr = 0;
        visible_MPs++;
    } else
        Ri[i].value++;

    // Reading Ri_in_Probe[i] contained in the Probe,
    cursor = 14;
    for (i = 0; i < MP_COUNT; i++) {
        memcpy(Ri_in_Probe[i].mac_addr, probe_frame+cursor, 6);
    }
}

```

```

        memcpy(&Ri_in_Probe[i].value, probe_frame+cursor+6, 1);
        if (comp_mac_addr(local_mac_addr, Ri_in_Probe[i].mac_addr)) { //
Getting Probe of Local Station,
        // Computing the Forward Delivery Ratio using the reported
received probes from local station,
        df = (double)Ri_in_Probe[i].value / 10;
        // Updating "df" in the corresponding entry in ETXi,
        for (j = 0; j < MP_COUNT; j++)
            if (comp_mac_addr(src_mac_addr, ETXi[j].mac_addr)) {
                ETXi[j].df = df;
                break;
            }
        break;
    }
    cursor += 7;
}

// Checking for the AVERAGING_PERIOD Seconds Period,
if (time(NULL) >= c_time + AVERAGING_PERIOD) {
    // Computing the Reverse Delivery Ratio for ALL visible stations,
    for (i = 0; i < MP_COUNT; i++) {
        ETXi[i].dr = (double)Ri[i].value / AVERAGING_PERIOD;
        if (ETXi[i].dr == 0 || ETXi[i].df == 0) // Avoiding division
by Zero,
            ETXi[i].etx = 1000; // Arbitrary: MAX,
        else
            ETXi[i].etx = 1 / (ETXi[i].dr * ETXi[i].df);
    }

    // Update/Write Ri[] to Shared Memory,
    memcpy(shm, Ri, SHARED_MEMORY_SIZE);
    // Update/Write ETXi[] to Shared Memory,
    memcpy(shm_ETX, ETXi, ETX_SHARED_MEMORY_SIZE);
    // --- Feedback,
    printf("\n *** ETX Values:");
    print_ETXi(ETXi);
    c_time = time(NULL);

    // Re-Initializing Ri[],
    for (i = 0; i < MP_COUNT; i++)
        Ri[i].value = 0;
    printf("\n\n ----- Collecting for a %d Seconds Period : \n",
AVERAGING_PERIOD);
    }
    fflush(stdout); printf("..");
}
}

void print_mac_addr(__u8 * mac) {
    int i;
    printf("\n");
    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;
}
int get_received_probes_for_MP(struct Received_Probe *Ri, __u8 *mac_addr) {
    int i;
    for (i = 0; i < MP_COUNT; i++)
        if (comp_mac_addr(Ri[i].mac_addr, mac_addr))
            return i;
    return -1;
}

void print_Received_Probes (struct Received_Probe *Ri) {
    int i;

    for (i = 0; i < MP_COUNT; i++) {
        printf("\n-----");
        print_mac_addr(Ri[i].mac_addr);
        printf(" --- Probes: %d", Ri[i].value);
    }
}

void print_ETXi (struct ETX *ETXi) {
    int i;
    for (i = 0; i < MP_COUNT; i++) {
        if (ETXi[i].etx != 1000) {

            printf("\n-----");
            print_mac_addr(ETXi[i].mac_addr);
            printf(" - df: %.2f", ETXi[i].df);
            printf(" - dr: %.2f", ETXi[i].dr);
            printf(" - ETX: %.2f", ETXi[i].etx);
        }
    }
    printf("\n-----");
}

```