

```

/* -----
(c) Riduan M ABID, eMail: R.Abid@auburn.edu
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 University - 2009,
----- */
#include <stdio.h>
#include <stdlib.h>
#include <math.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 <string.h>
#include <time.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define HASH_TABLE_SIZE 255
#define MP_COUNT 10
#define AVERAGING_PERIOD 4
#define R_SINRs_SHARED_MEMORY_SIZE (MP_COUNT+1) * sizeof(struct reverse_SINR)
#define KEY_R_SINR 1331131

struct hash_entry {
    __u8 mac_addr[6];
    int last_rssi, avg_rssi, avg_rate, avg_length;
    __u32 rssi_sum, rate_sum, length_sum;
    __u32 arrivals;
};
struct reverse_SINR {
    __u8 mac_addr[6];
    double R_SINR, avg_rssi, avg_rate, avg_length;
    double interference_RSSI_sum;
};

struct linux_wlan_ng_val {
    __u32 did;
    __u16 status, len;
    __u32 data;
};
struct linux_wlan_ng_prism_hdr {
    __u32 msgcode;
    __u32 length;
    __u8 devname[16];
    struct linux_wlan_ng_val hosttime, mactime, channel, rssi, sq, signal, noise,
rate, istx, frmlen;
};

```

```

void print_prism_header(struct linux_wlan_ng_prism_hdr prism_hdr);
int get_fill_hash_entry(struct hash_entry* table, __u8* mac_addr);
void fill_hash_entry(struct hash_entry* table, __u8* mac_addr);
int is_neighbor(struct hash_entry* table, __u8* mac_addr);
void update_print_hash_table(struct hash_entry *table);
void print_mac_addr(__u8 * mac);
int comp_mac_addr(__u8 * mac1, __u8 * mac2);

__u8 Adhoc_ifr_mac_addr[6], src_mac_addr[6], dest_mac_addr[6], NULL_addr[6] = {0x00,
0x00, 0x00, 0x00, 0x00, 0x00}
,broadcast_addr[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff}, local_addr[6];

int main(int argc, char* argv[]) {
    __u8 buffer[4096], eltID, *shm;
    key_t key_R_SINRs;
    struct sockaddr dest, name;
    int shmid_R_SINRs, dlen, recievedBytes, len;
    int raw_sock, fd, i, j, cnt = 0, visible_MPs = 0, k = 0, frame_cnt = 0;
    __u32 timeStamp, c_time, refreshStamp;
    __u32 airtime_sum = 0;
    int airtime = 0;
    double CCI, Pr, lambda, tau;
    struct ifreq ifr;
    struct sockaddr_ll sll;
    struct linux_wlan_ng_prism_hdr prism_hdr;
    struct hash_entry hash_table[HASH_TABLE_SIZE], Neighbors[MP_COUNT], external;
    struct reverse_SINR R_SINRs[MP_COUNT+1];

    // Checking for Arguments,
    if (argc < 2) {
        perror("\n Inussuficient 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(Adhoc_ifr_mac_addr, ifr.ifr_hwaddr.sa_data, 6);
    memcpy(local_addr, Adhoc_ifr_mac_addr, 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);
}

// Initialize the Hash Table,
for (i=0; i < HASH_TABLE_SIZE; i++) {
    memcpy(hash_table[i].mac_addr, NULL_addr, 6);
    hash_table[i].rssi_sum = hash_table[i].rate_sum =
hash_table[i].length_sum =0;
    hash_table[i].arrivals = 0;
}
// Initialize Neighbors, SINR Entries,
for (i=0; i <= MP_COUNT; i++) {
    memcpy(Neighbors[i].mac_addr, NULL_addr, 6);
    Neighbors[i].rssi_sum = Neighbors[i].rate_sum = Neighbors[i].length_sum
=0;
    Neighbors[i].arrivals = 0;
    memcpy(R_SINRs[i].mac_addr, NULL_addr, 6);
}
// Initializing External Interferences entry,
memcpy(external.mac_addr, NULL_addr, 6);
external.rssi_sum = external.rate_sum = external.length_sum =
external.arrivals = 0;

// Creating R_SINRs Shared Memory,
key_R_SINRs = KEY_R_SINR;

if ((shmid_R_SINRs = shmget(key_R_SINRs, R_SINRs_SHARED_MEMORY_SIZE,
IPC_CREAT)) < 0) {
    perror("shmget");
    exit(1);
}
if ((shm = shmat(shmid_R_SINRs, NULL, 0)) == NULL) {
    perror("shmat");
    exit(1);
}

//
-----
-----
c_time = time(NULL);
while(1) {

```

```

recievedBytes = recvfrom(raw_sock,buffer,4096,0,&dest,&dlen);
if (recievedBytes > 2100) {
    printf("\n !!!!!!!! HUGE Frame");
    continue;
}
memcpy(dest_mac_addr, buffer+148, 6);
memcpy(src_mac_addr, buffer+154, 6);

// Getting Prism Header,
memcpy(&prism_hdr, buffer, 144);
if (prism_hdr.rate.data == 0)
    continue;
frame_cnt++;
// Getting Frame eltID,
memcpy(&eltID, buffer+174, 1);
if (eltID == 13 || eltID == 5) { // i.e., an 802.11s Routing Frame from
a neighbor,
    if (!comp_mac_addr(src_mac_addr, local_addr)) {
        memcpy(dest_mac_addr, buffer+148, 6);
        memcpy(src_mac_addr, buffer+154, 6);
        i = get_fill_hash_entry(hash_table, src_mac_addr);
        hash_table[i].last_rssi = prism_hdr.rssi.data;
        hash_table[i].rssi_sum += hash_table[i].last_rssi;
        hash_table[i].rate_sum += prism_hdr.rate.data;
        hash_table[i].length_sum += prism_hdr.frmlen.data;
        hash_table[i].arrivals++;
    }
} else {
    i = is_neighbor(hash_table, src_mac_addr);
    if (i != 0) { // DATA Traffic in Local WMN,
        hash_table[i].last_rssi = prism_hdr.rssi.data;
        hash_table[i].rssi_sum += hash_table[i].last_rssi;
        hash_table[i].rate_sum += prism_hdr.rate.data;
        hash_table[i].length_sum += prism_hdr.frmlen.data;
        hash_table[i].arrivals++;
    } else { // a Non WMN Traffic: External Interference,
        external.last_rssi = prism_hdr.rssi.data;
        external.rssi_sum += external.last_rssi;
        external.rate_sum += prism_hdr.rate.data;
        external.length_sum += prism_hdr.frmlen.data;
        external.arrivals++;
    }
}
// For Feedback,
//print_prism_header(prism_hdr);

// ===== //
// Computing the airtime/transmission_time for the current frame,
airtime = prism_hdr.frmlen.data * 8 / prism_hdr.rate.data; // millionths
of seconds,
airtime_sum += airtime;
timeStamp = time(NULL);

// ===== Averaging, Updating Neighbors Shared Memory,

```

```

        if (timeStamp >= c_time + AVERAGING_PERIOD) {
            CCI = (double)airtime_sum / (AVERAGING_PERIOD * 1000000); //
Accounting for the Mb/sec - In Percentage,
            // Feedback,
            printf("\n
=====");
            printf("\n - %d * Cell Contention Indicator: %.2f", cnt++, CCI);
            printf("\n - TimeStamp Second: %d - Total Received Frames: %d",
timeStamp%1000, frame_cnt);
            printf("\n
=====");

            sleep(1); // Refreshing,
            // Updating averages,
            update_print_hash_table(hash_table);

            // Copying Neighbors Data,
            for (i=0, k=0; i < HASH_TABLE_SIZE; i++)
                if (!comp_mac_addr(hash_table[i].mac_addr, NULL_addr))
                    Neighbors[k++] = hash_table[i];
            // Computing Reverse_SINRs,
            for (i=0; i < MP_COUNT; i++) { // For every WMN Neighbor,
                if (comp_mac_addr(Neighbors[i].mac_addr, NULL_addr))
                    continue; // Skipping empty entries,
                memcpy(R_SINRs[i].mac_addr, Neighbors[i].mac_addr, 6);
                R_SINRs[i].interference_RSSI_sum = 1; // Initializing-
Accounting for Thermal Noise,
                R_SINRs[i].avg_rssi = Neighbors[i].avg_rssi;
                R_SINRs[i].avg_rate = Neighbors[i].avg_rate;
                R_SINRs[i].avg_length = Neighbors[i].avg_length;

                // Getting TAU first, i.e., the channel occupation time for
station i,
                printf("\n*"); print_mac_addr(Neighbors[i].mac_addr);
                tau = (double)Neighbors[i].length_sum * 8 /
Neighbors[i].avg_rate / 1000; // in ms,

                // Getting SINR shares from other stations,
                for (j = 0; j < MP_COUNT; j++) {
                    if (!comp_mac_addr(Neighbors[j].mac_addr,
Neighbors[i].mac_addr) && !comp_mac_addr(Neighbors[j].mac_addr, NULL_addr)) { //
Excluding desired station, i.e, 'i',
                        // Getting lambda - Average arrival rate,
                        lambda = (double)Neighbors[j].arrivals /
AVERAGING_PERIOD / 1000; // Per ms,
                        // Computing Pr,
                        Pr = 1 - exp(-lambda*tau); // Pr of having at
least one interfering packet,
                        R_SINRs[i].interference_RSSI_sum += Pr *
Neighbors[j].avg_rssi;
                    }
                }
                // Accounting for External (non-WMN) Interferences share,
                lambda = (double) external.arrivals / AVERAGING_PERIOD /

```

```

1000; // Per ms,
// Getting the Pr,
Pr = 1 - exp(-lambda*tau);
R_SINRs[i].interference_RSSI_sum += Pr * external.rssi_sum /
external.arrivals;
R_SINRs[i].R_SINR = R_SINRs[i].avg_rssi /
R_SINRs[i].interference_RSSI_sum;
printf(" ---> R_SINR: %.2f", R_SINRs[i].R_SINR);
}

// Writing R_SINRs Shared Memory,
R_SINRs[MP_COUNT].R_SINR = CCI;
memcpy(shm, R_SINRs, R_SINRs_SHARED_MEMORY_SIZE);

// Re-initialize Neighborss - R_SINRs,
for (i=0; i < MP_COUNT; i++) {
    memcpy(Neighbors[i].mac_addr, NULL_addr,
6);memcpy(R_SINRs[i].mac_addr,NULL_addr, 6);
    Neighbors[i].rssi_sum = Neighbors[i].rate_sum =
Neighbors[i].length_sum =0;
    Neighbors[i].arrivals = 0;
    R_SINRs[i].R_SINR = 0;
}
// Re-initialize Hash Table;
for (i=0; i < HASH_TABLE_SIZE; i++) {
    memcpy(hash_table[i].mac_addr, NULL_addr, 6);
    hash_table[i].rssi_sum = hash_table[i].rate_sum =
hash_table[i].length_sum = 0;
    hash_table[i].arrivals = 0;
}
// Re-initialize the external interference entry,
external.rssi_sum = external.rate_sum = external.length_sum =
external.arrivals = 0;
// Reinitializing Timer,
c_time = time(NULL);
airtime_sum = frame_cnt = 0;
}
}

int get_fill_hash_entry(struct hash_entry* table, __u8* mac_addr) {
    int i;
    i = mac_addr[5] % 255;
    memcpy(table[i].mac_addr, mac_addr, 6);
    return i;
}
void fill_hash_entry(struct hash_entry* table, __u8* mac_addr) {
    int i;
    i = mac_addr[5] % 255;
    memcpy(table[i].mac_addr, mac_addr, 6);
}
int is_neighbor(struct hash_entry* table, __u8* mac_addr) {
    int i;
    i = mac_addr[5] % 255;

```

```

        if (comp_mac_addr(table[i].mac_addr, NULL_addr))
            return 0;
        if (!comp_mac_addr(table[i].mac_addr, mac_addr)) // Avoiding Collision in the
Hash Table,
            return 0;
        return i;
    }

void update_print_hash_table(struct hash_entry *hash_table) {
    int i;
    printf("\n ----- Neighbors HASH TABLE ----- ");
    for (i = 0; i < HASH_TABLE_SIZE; i++) {
        if (!comp_mac_addr(hash_table[i].mac_addr, NULL_addr)) {
            // Updating,
            hash_table[i].avg_rssi = hash_table[i].rssi_sum /
hash_table[i].arrivals;
            hash_table[i].avg_rate = (double)hash_table[i].rate_sum /
hash_table[i].arrivals;
            hash_table[i].avg_length = (double)hash_table[i].length_sum /
hash_table[i].arrivals;
            // Printing,
            printf("\n *"); print_mac_addr(hash_table[i].mac_addr);
            printf("| RSSI:%d", hash_table[i].avg_rssi);
            printf("| Rate:%d", hash_table[i].avg_rate);
            printf("| Length:%d", hash_table[i].avg_length);
            printf("| Arrivals:%d", hash_table[i].arrivals);
            //printf("| Noise: %d", hash_table[i].noise);
        }
    }
    printf("\n ----- ");
}

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 = 1; i < 6; i++) // Staring from 1 instead of 0 in order to acarrials
for the difference in Addr between ath0
        if (mac1[i] != mac2[i]) // and ath1,
            return 0;
    return 1;
}

void print_prism_header(struct linux_wlan_ng_prism_hdr prism_hdr) {
    printf("\n -----");
    printf("\n MsgCode: %d", prism_hdr.msgcode);
    printf("\n Length: %d", prism_hdr.length);
    printf("\n Device Name: %s", prism_hdr.devname);
    printf("\n channel: %d", prism_hdr.channel.data);
    printf("\n rssi: %d", prism_hdr.rssi.data);
    printf("\n sq: %d", prism_hdr.sq.data);
    printf("\n Signal: %d", prism_hdr.signal.data);
    printf("\n Noise: %d", prism_hdr.noise.data);
}

```

```
printf("\n Rate: %d", prism_hdr.rate.data);  
printf("\n istx: %d", prism_hdr.istx.data);  
printf("\n Frame Length: %d", prism_hdr.frmlen.data);  
}
```