Vertical Handoff Implementation of SIGMA

Abu Ahmed Sayeem Reaz, Mohammed Atiquzzaman

TR-OU-TNRL-06-106
July 2006

Telecommunication & Network Research Lab

School of Computer Science

THE UNIVERSITY OF OKLAHOMA

200 Felgar Street, Room 160, Norman, Oklahoma 73019-6151
(405)-325-0582, atiq@ou.edu, www.cs.ou.edu/~netlab

SIGMA Vertical Handoff

Abu Ahmed Sayeem Reaz
Telecommunications and Networking Research Laboratory
School of Computer Science
University of Oklahoma,
Norman, OK 73019-6151, USA.
July 13, 2006

I. Vertical Handoff

Handoff taking place during an inter-network movement of Mobile Host (MH) is defined as vertical handoff. In other words, when MH moves out of a particular type of network and moves into another type and the connection is handed over from one to another; that is vertical handoff. It is called vertical because usually networks have different coverage and usually are overlapping. So from a topological view, this handover is taking place vertically to a higher or lower coverage network (HCN and LCN respectively).

Vertical handoff takes place usually for overlay networks. A very simple example would be IrDA → WLAN → CDMA. These three different types of network are in overlay topology. Whenever the MH moves out of the coverage of one, it moves into another one.

Now, if this handoff is taking place between similar networks, its horizontal handoff. This handoff is like moving from one coverage area to another one hence the nomenclature is derived. WLAN → WLAN handoff is an example of horizontal handoff.

WLAN → WLAN → CDMA can be an example of combination of horizontal and vertical handoff. The following figure illustrates the procedure of both the handoffs:

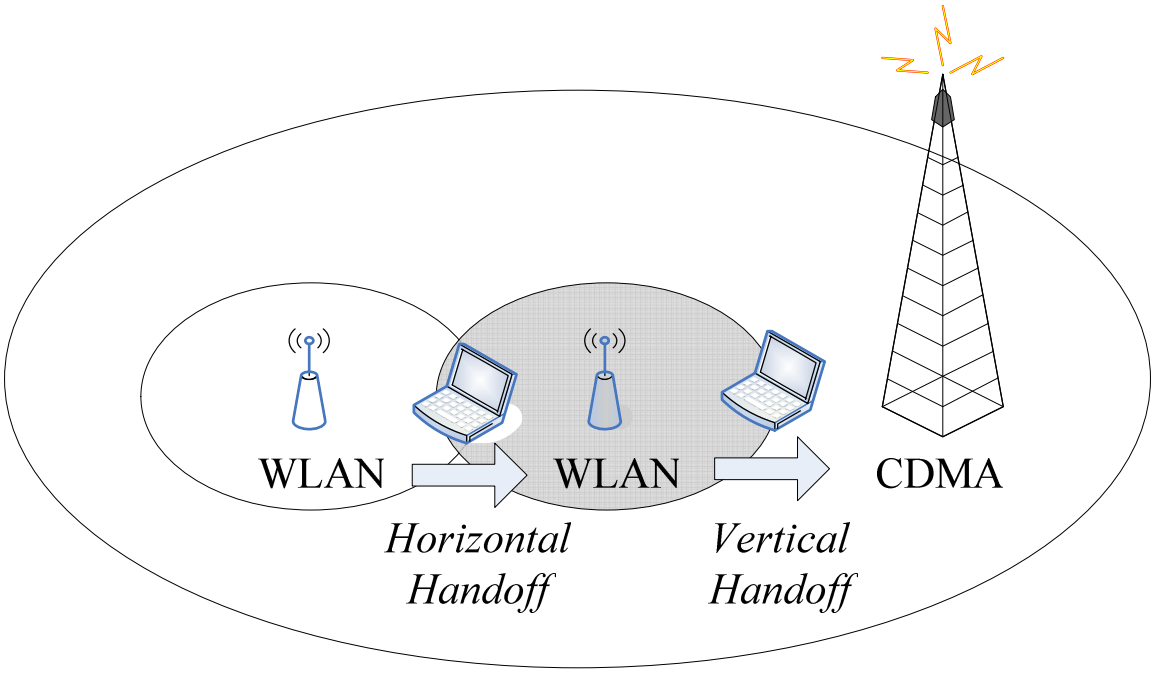

Figure 1: Vertical and horizontal handoff

II. Fundamentals of Vertical Handoff

The real challenge of vertical handoff is to decide when to perform the handoff. Usually the HCNs are lower speed network. So the basic idea is to try to use LCNs.

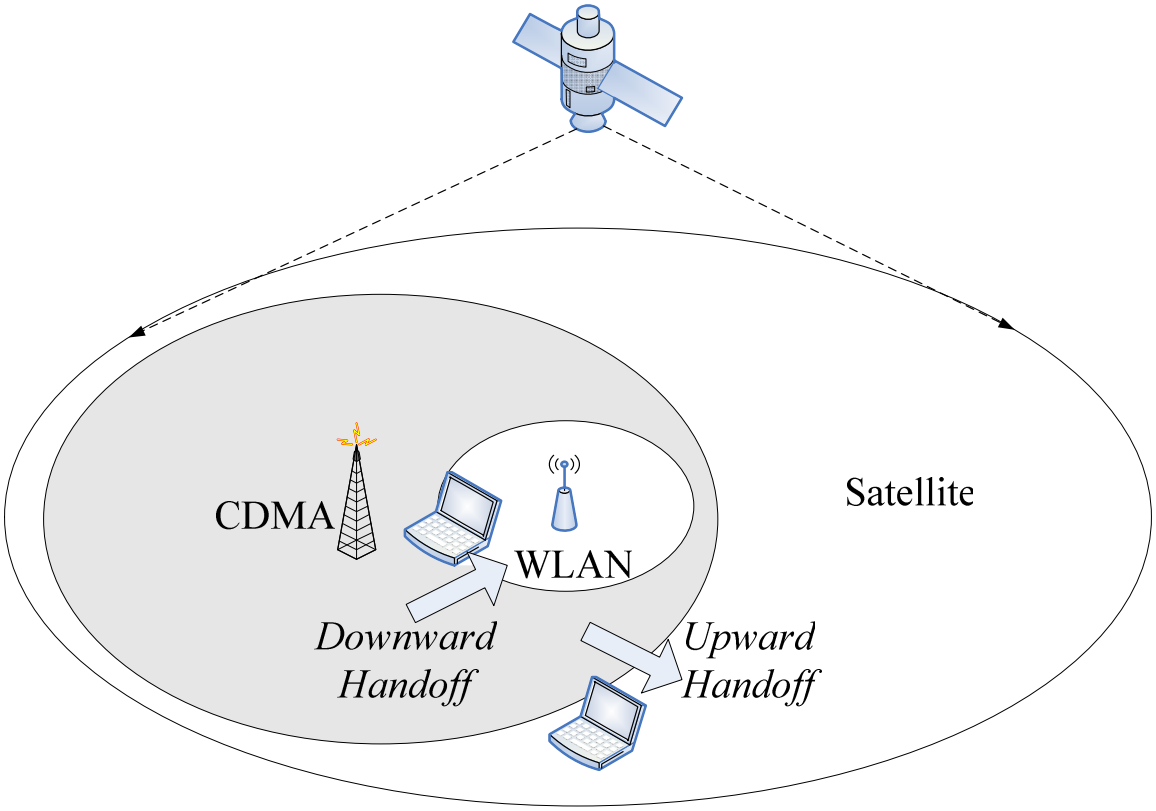

Figure 2: Vertical handoff classification

Switching from LCN to HCN (e.g. CDMA → Satellite) is henceforth called *upward handoff* and from HCN to LCN (e.g. WLAN ← CDMA) is called *downward handoff*. So the fundamental principle of doing vertical handoff is to always try to do downward handoff. Whenever a LCN is available, the system should try to move to that network. Upward handoffs are going to be performed only when MH is moving out of a LCN.

III. Handoff in SIGMA

Seamless IP diversity based Generalized Mobility Architecture (SIGMA) [1] is a mobility management technique which exploits IP diversity offered by multiple interfaces in mobile devices. When a MH moves into the coverage of a new subnet, it obtains a new IP address while retaining the old one in the overlapping area of the two subnets. The MH communicates through the old IP address while setting up a new connection through

the newly acquired IP address. When the signal strength of the old Access Point (AP) drops below a certain threshold, the connection is handed over to the new subnet and the new IP address is set to be the primary one. When the MH leaves the overlapping area, it releases the old IP address and only communicates over the new IP address. The duration of the MH in the overlapping area and the time during which the MH communicates over both IP addresses depend on the velocity of the MH and the power of the signals from the access points. Each time the MH hand to a new subnet, it updates the DNS with its new IP address.

IV. Vertical Handoff topology in SIGMA

The topology of the vertical handoff is illustrated in Figure 4.

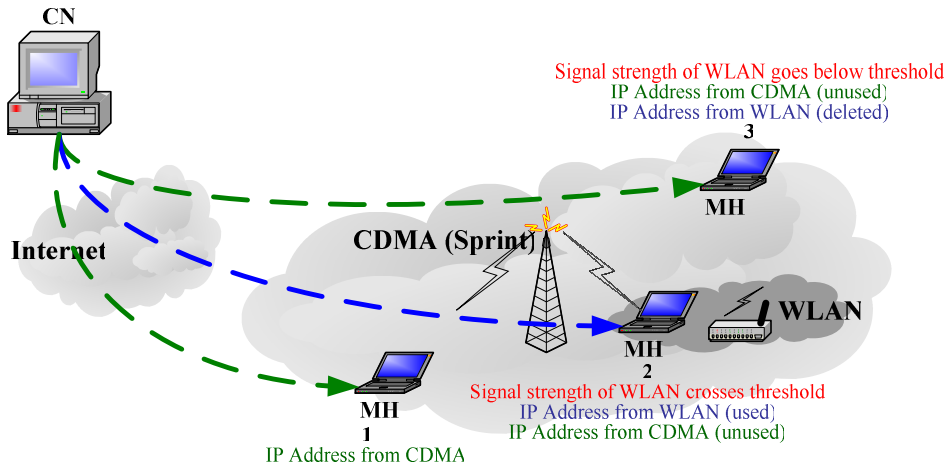

Figure 4: SIGMA Topology with Cellular Network

Coverage area of CDMA is determined by the coverage area of the Cell Phone service itself. It is assumed that the CDMA connection is always available. So the coverage of the WLAN falls within the coverage of CDMA and whenever MH moves out of WLAN, it goes right into CDMA. Therefore, it can be validly assumed that CDMA service is ubiquitous (similar assumptions made in literatures by Sharma et al.[2], Chakravorty et al.[3], Bernaschi et al.[4]).

The MH can initiate the communication either in WLAN or in CDMA. As WLAN is the higher speed one, we know that the MH always tries to connect to WLAN. So, MH connects to CDMA through upward handoff only when it moves out of WLAN. In the same way, whenever MH moves into WLAN, it would change attachment from to it from GPRS through downward handoff.

The vertical handoff principle is based on the fact that CDMA is assumed to be always available. So, only the signal strength of WLAN is measured for handoff decision. Whenever a signal is obtained from WLAN, an IP address is obtained. When it goes up the threshold level, connection is handed over to the WLAN from CDMA. Same principle is applied for other way. Whenever the signal strength goes below the threshold, the connection is handed over to CDMA. It is possible because the IP address from CDMA is always going to be there and all it requires is to set that address as the primary one (refer to the SCTP API documentation).

V. Vertical Handoff implementation in SIGMA

V. a. Installation of CDMA card

Only Sprint provides data service over CDMA in Norman, so Sprint's service is necessary to implement vertical handoff. They provide Sprint PC-5740 EVDO card. SIGMA test bed is implemented on Fedora Core 3 (FC3). To install and use the Sprint PC-5740 on FC3, the following steps were performed.

i. Load the `ohci-hcd` module with this command:

```
modprobe ohci-hcd
```

ii. Load usbserial module. To get it to work, it is required to specify exactly what card to found. For the PC-5740 model, the vendor id is 0x106c and the product is 0x3701.
An easy way to find that out is to look at what devices are installed before inserting the card, and after:

- ~$ cat /proc/bus/usb/devices > devices
- insert card
- ~$ diff /proc/bus/usb/devices devices | grep Vendor
  < P: Vendor=0000 ProdID=0000 Rev= 2.06
  < P: Vendor=0000 ProdID=0000 Rev= 2.06
  < P: Vendor=106c ProdID=3701 Rev=0.00

So when we inserted the card, the list of devices included one with a vendor ID of 106c and a product ID of 3701.
Then, load the module by
modprobe usbserial vendor=0x106c product=0x3701

iii. The system should have a file called /dev/ttyACM0.

iv. The following three scripts called sprint, sprint-connect and sprint-disconnect should be saved under `/etc/ppp/peers/` directory.

- sprint

  #the USB serial device of the EVDO PCMCIA card.
  ttyACM0
  #your login information
  user 4058244386@sprintpcs.com
  #230400 # speed
  57600 # speed
  #debug
  defaultroute # use the cellular network for the default route
  usepeerdns # use the DNS servers from the remote network
  -detach # keep pppd in the foreground
  crtscts # hardware flow control
  #lock # lock the serial port
  noauth # don't expect the modem to authenticate itself
  connect "/usr/sbin/chat -v -f /etc/ppp/peers/sprint-connect"
  disconnect "/usr/sbin/chat -v -f /etc/ppp/peers/sprint-disconnect"

- sprint-connect

  SAY 'Starting Sprint\n'
  TIMEOUT 120
  ABORT 'BUSY'
  ABORT 'NO ANSWER'
  ABORT 'NO CARRIER'
  " 'ATZ'
  'OK' 'AT&F0'
  'OK' 'ATE0v1'
  " 'AT+CSQ'
  'OK' 'ATDT#777'
  'CONNECT'

- sprint-dicconnect

  "" "\K"
  "" "+++ATH0"
  SAY "Disconnected from Sprint."

v. Connect to the Sprint by using `pppd call sprint` at the `/etc/ppp/peers/` directory.

V. b. Vertical Handoff architecture and algorithm

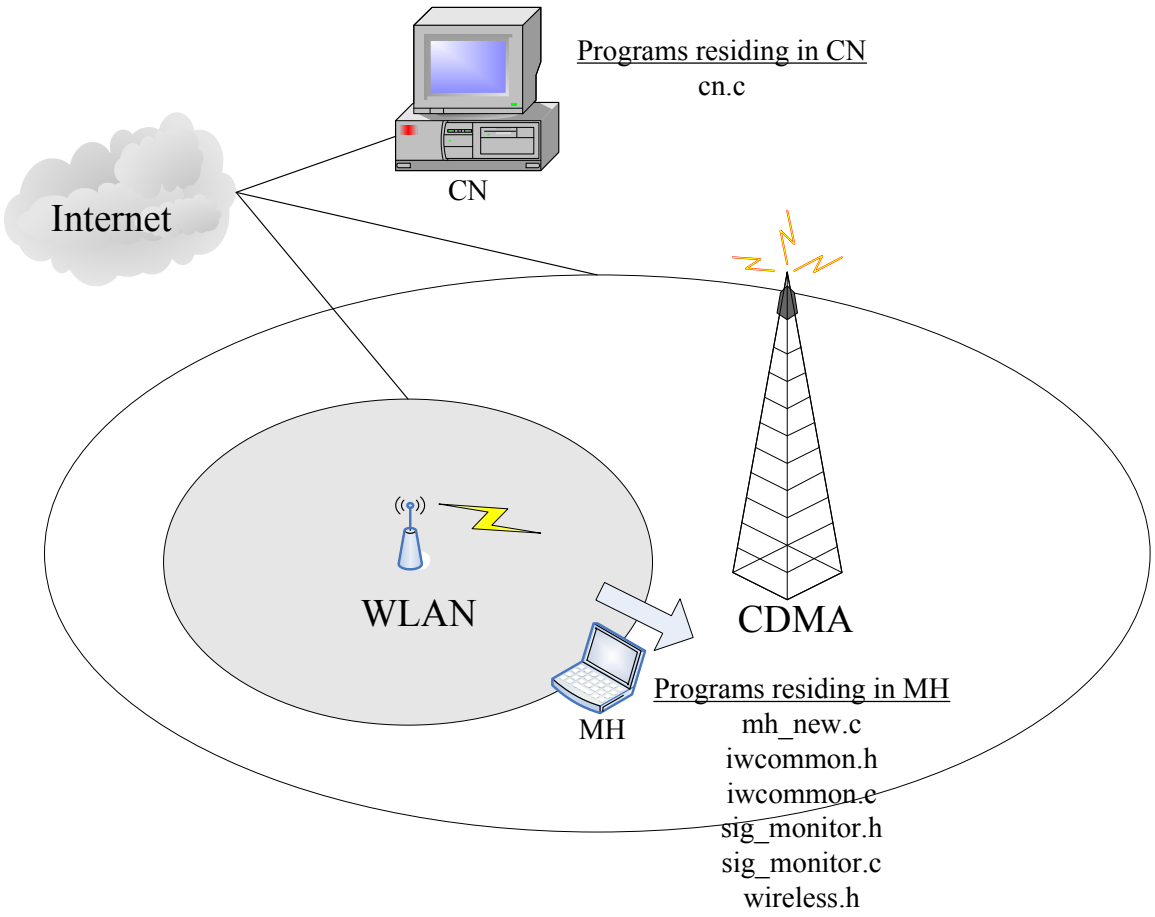

Figure 5: SIGMA test bed architecture for vertical handoff

Fig. 5 describes the architecture of SIGMA vertical handoff. As described in Sec. IV, only signal strength of WLAN is measured at MH. When MH moves out of WLAN coverage it hands off to CDMA and vice versa. The programs used to test vertical handoff are as follows:

1. cn.c runs on CN and is just a simple server program, which accepts an SCTP connection request from mh_new.c and sends a test packet.
2. mh_new.c runs on MH and is the client program corresponding to cn.c. It sends SCTP connection request to cn.c and receives test packets from cn.c.
3. iwcommon.h and iwcommon.c is required to obtain signaling information from Linux kernel.
4. sig_monitor.h and sig_monitor.c actually measures the signal of WLAN.
5. wireless.h is the supporting wireless header of Linux.

These programs are listed in the appendix.

The algorithm for the vertical handoff is as follows which is illustrated by fig 6.

```
SIGMA vertical handoff program
     connect to CDMA

     while (1)
        monitor WLAN signal strength S
        if S > threshold
           connect to WLAN and use WLAN
        if S < threshold
           use CDMA
```

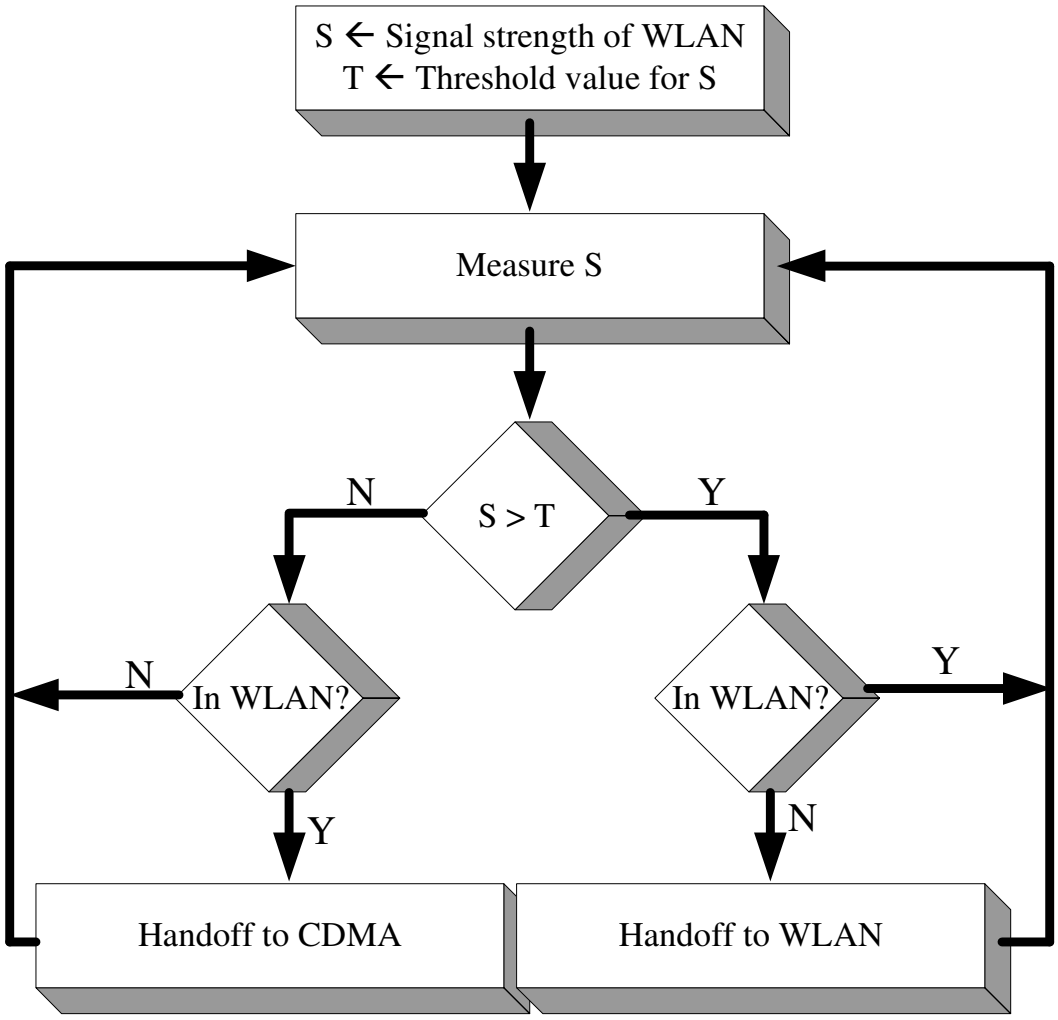

Figure 6: SIGMA vertical handoff algorithm

V. c. Test bed setup

Prerequisites to run cn.c and mh_new.c

1. CN and MN should be Linux machines
2. lksctp package must be installed on both CN and MN.
3. CN should be connected to the internet, and should not be behind any firewall so that cn.c can listen to a port and accept connection requests from mh_new.c
4. MN should be a laptop and must have two wireless interface cards: one for WLAN and the other one for Cellular network.

How cn.c works

cn.c is a simple server program that accepts a connection request, and sends test packets over the connection.

How mh_new.c works

1. mh_new.c assumes it is under an 802.11 network when launched, and a WLAN interface is connected to the WLAN Network (Network 1)
2. It also assumes that the other interface is always connected to a ubiquitous cellular network (Network 2).
3. mh_new.c constantly monitors the signal strength from the WLAN interface using iwconfig. When the signal strength drops below a given threshold value, the connection is switched to the CDMA interface, on the assumption that MH is already connected to Network 2.
4. When the connection is switched, the routing table of MH is changed so that all the subsequent packets go to the new network (Network 2).
5. Inversely, when the signal strength of WLAN goes above the threshold value, the connection is switched back to Network 1 and routing table is again modified.

Set of IP addresses used

Machine	**Interface**	**IP Address**
CN	Ethernet	129.15.78.139
MH	WLAN	10.1.8.5
MH	CDMA	70.2.159.60

How to Run cn.c and mn.c

1. Copy `cn.c` to CN
   1.1 cn.c listens to an available port where it can serve connection request
   1.2 Compile cn.c with the command:
   ```
   %cc –o cn cn.c –lm –lsctp –lpthread
   ```

1.3 Execute with the command:

```
%./cn LOCAL_IP LOCAL_PORT
```

2. Copy `mn.c` to the MN (Laptop):
   2.1 In order to compile and run mn.c, the following files needs to be in the same directory of mh_new.c:
   - `iwcommon.h`
   - `iwcommon.c`
   - `sig_monitor.h`
   - `sig_monitor.c`
   - `wireless.h`

   2.2 Compile mn.c with the command:

```
%cc -o mh mh_new.c sig_monitor.c iwcommon.c -lm -lsctp -lpthread
```

   2.3 Login as root user
   2.4 Make sure MN is in the coverage of Network 1 in the new test bed.
   2.5 Execute with the command:

```
%./mh PORT DESTINATION_IP DESTINATION_PORT
```

VI. Results

Figure 7 illustrates the throughput of the data communication.

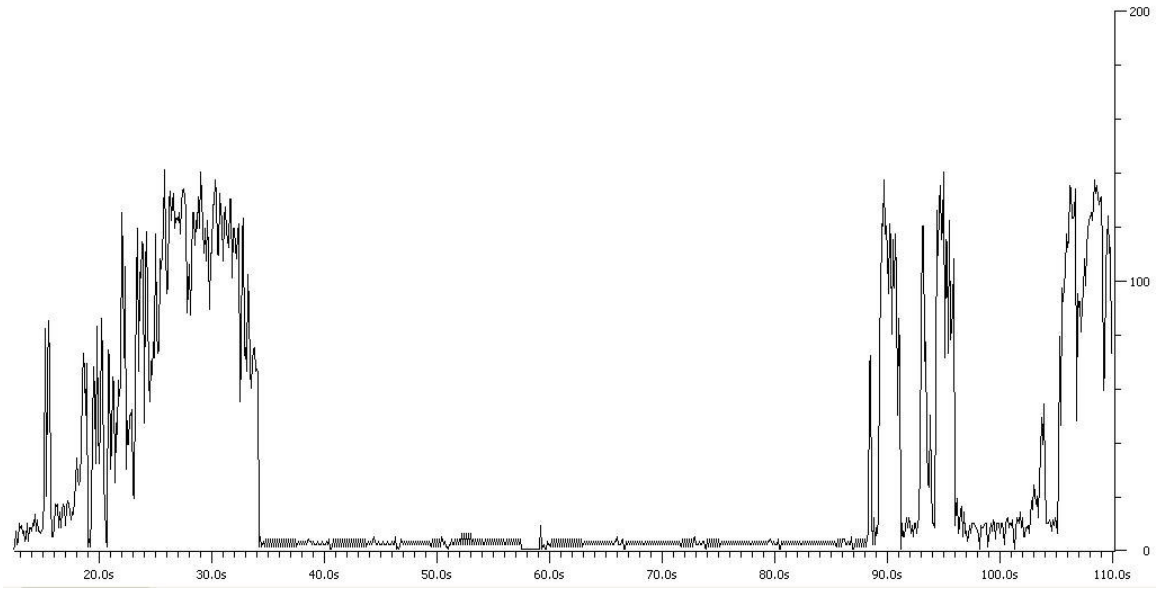

Figure 7: SIGMA vertical handoff algorithm

We can see that between 34.2512 and 34.4257 seconds, the handoff from WLAN to CDMA has occurred. This is more illustrated in figure 8.

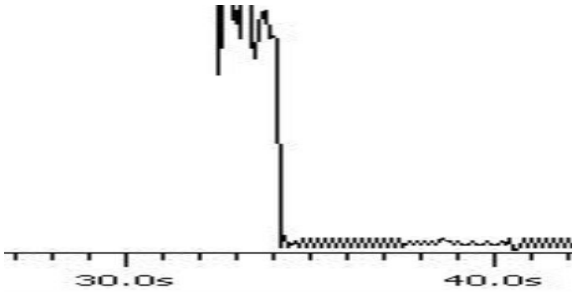

Figure 8: Throughput during upwards vertical handoff

CDMA has lower data rate than WLAN. It is reflected in fig. 8. But we can see that the throughput does not drop to zero. This handoff can be observed by watching the change of direction of packets. We can observe that packet from CN (129.15.78.139) coming to WLAN (10.1.8.5) is switched to CDMA (70.2.159.60). Figure 9(a) and (b) shows the handoff. The time taken for upward handoff is 0.17 seconds.

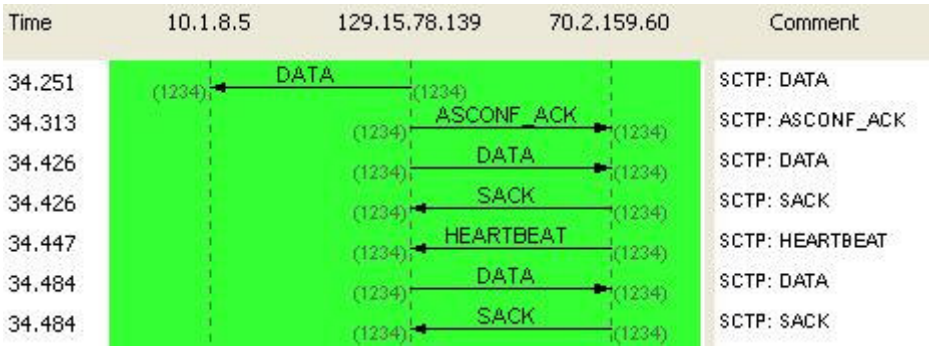

Figure 9 (a): Signaling diagram for upward handoff

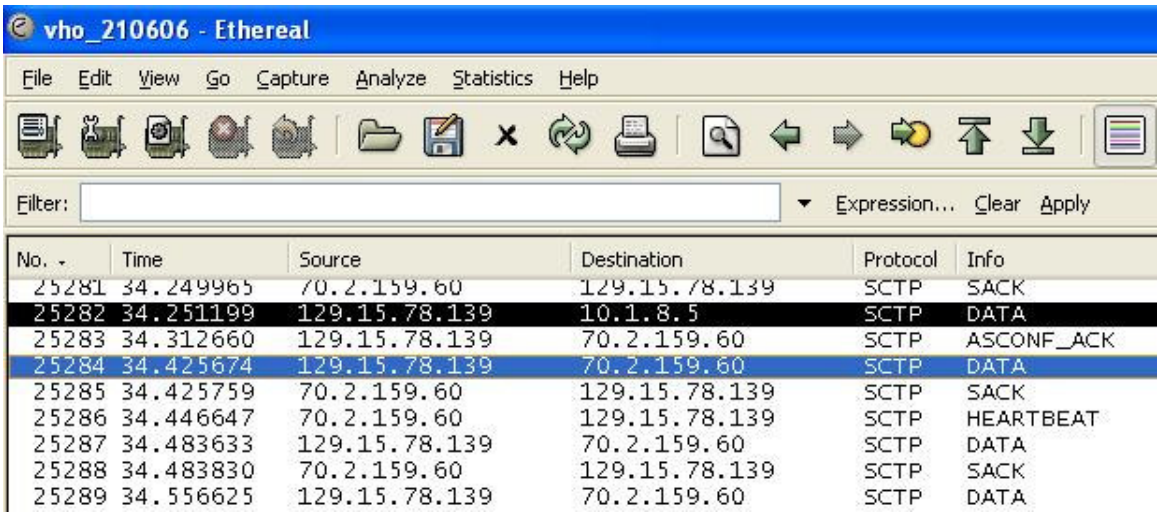

Figure 9 (b): Ethereal capture during upward handoff (black: WLAN, blue: CDMA)

We can observe from Fig. 7 that between time 88.4626 and 88.4944 a handoff from CDMA to WLAN has taken place. Fig. 10 illustrates this downward handoff.

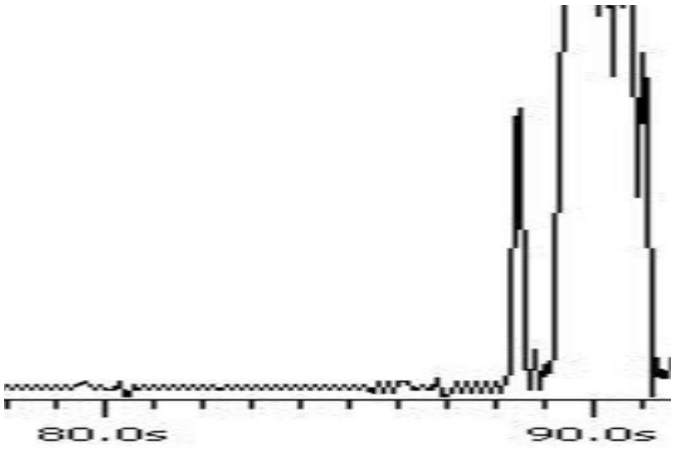

Figure 10: Throughput during downwards vertical handoff

When the downwards handoff is performed, we can see the jump in throughput. The handoff can be observed from fig. 11(a) and 11(b) where we can see the packets from CN is diverted from CDMA to WLAN. SIGMA takes about 0.03 seconds to perform a downward handoff.

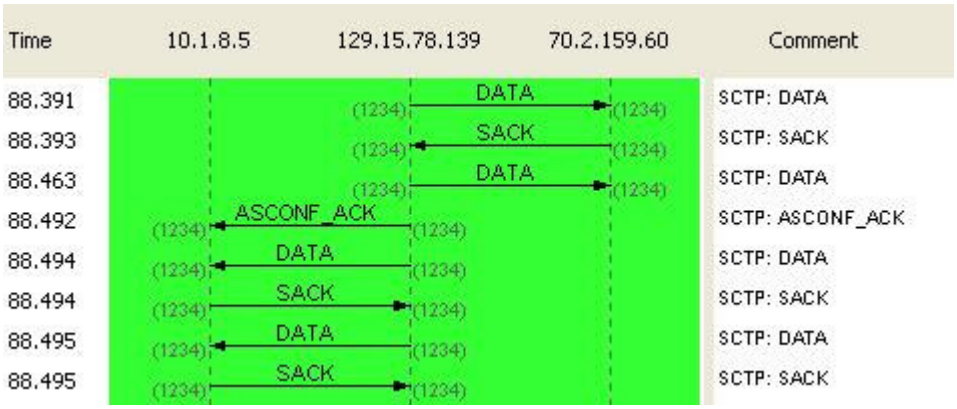

Figure 11 (a): Signaling diagram for downward handoff

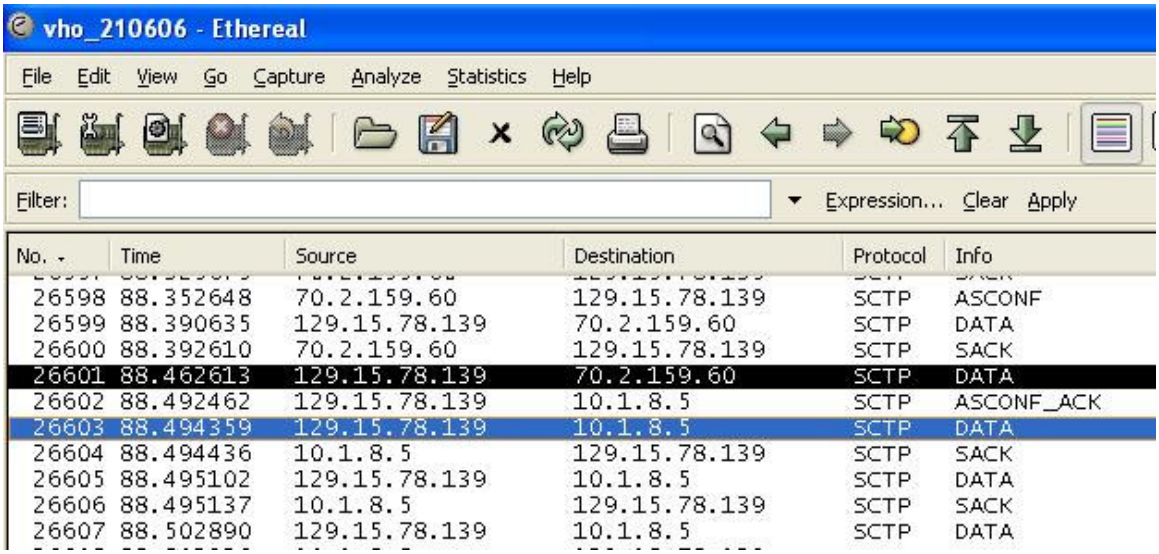

Figure 11 (b): Ethereal capture during upward handoff (black: CDMA, blue: WLAN)

Appendix

mh.c

```
/* Compile: %cc -o mh mh_new.c sig_monitor.c iwcommon.c -lm -lsctp -
lpthread
*/


#include<pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>
#include <arpa/inet.h>
//#include <net/sock.h>
#include <netinet/in.h>
#include <net/if.h>
#include <netinet/sctp.h>
//#include <net/if_arp.h>
#include <pthread.h>
#include "sig_monitor.h"
#include "sys/time.h"
#include <unistd.h>
#include <sys/ioctl.h>


//#define SNR_HYST 0
#define ADD_THRESH 10
#define MAX_DATA_BUFFER 256
#define TRTIME 3
#define inaddrr(x) (*(struct in_addr *) &ifr->x[sizeof sa.sin_port])

#define IFRSIZE   ((int)(size * sizeof (struct ifreq)))

int count2=0;
int handover1=0;
int hyst1,hyst2;
sctp_assoc_t ass_id;
int SNR_HYST = 180;

int count=0,count1=0;
struct sock_lock
{
      pthread_mutex_t sock_mutex;
      int sockfd;
};


        float t,tt,t1;
int sd(int*);
```

```
int sd1(int*);
void handle_event(void *buf);
int display(char*, iwstats*);
int display1(char*, iwstats*);
void *handoff(void *args);
double gettime();
int local_port;
int val[200];
int val1[200];

char wlan_addr[16];
char cdma_addr[16];

void handle_event(void *buf)
{
struct sctp_assoc_change *sac;
sac = (struct sctp_assoc_change *)buf;
printf("the association  id is %x\n",sac->sac_assoc_id);
ass_id = sac->sac_assoc_id;
return;
}

void getaddy();

struct sctp_sndrcvinfo sri;
struct sctp_event_subscribe events;
int msg_flags;
int main(int argc, char **argv)
{
      int addrlen;
        struct timeval time;
      int connected;
      pthread_t hover_thrd;
      struct sock_lock sd = {PTHREAD_MUTEX_INITIALIZER, -1};
      struct sockaddr_in local_addr;
      struct sockaddr_in serv_addr;
      int bound;
        int port;
      int pid;
      FILE *ofp;
      int recved;
        int handover=0; //add by suren just to check the handover.
      char buf[MAX_DATA_BUFFER];
      int addrcnt = 1;



      float t2,t3;
                gettimeofday(&time, NULL);
                t=time.tv_sec/1.0;
                tt=time.tv_usec/1000000.0;
        printf("Starting time %f\n",t);
        printf("Starting time %f\n",tt);
      ofp = fopen("test.txt", "w");
      if(argc != 3)
      {
```

```
            fprintf(stderr, "Usage: mh DESTINATION_IP
DESTINATION_PORT\n");
            exit(1);
      }

      /* Create socket */
      sd.sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
      if(sd.sockfd < 0)
      {
            perror("Error:: Can not create socket:");
            exit(1);
      }

      local_port = atoi(argv[2]);

      getaddy();

      /* Initialize the local address */
      bzero(&local_addr, sizeof(struct sockaddr_in));
      local_addr.sin_family = AF_INET;
      //local_addr.sin_addr.s_addr = inet_addr(argv[1]);
      local_addr.sin_addr.s_addr = inet_addr(wlan_addr);
      local_addr.sin_port = htons(local_port);

      printf("WLAN addy: %s\n", wlan_addr);

      /* Bind address to local port */
//    bound = sctp_bindx(sd.sockfd, (struct sockaddr *) &local_addr,
addrcnt,
            //          SCTP_BINDX_ADD_ADDR);
        bound = bind(sd.sockfd, (struct sockaddr *)&local_addr,
sizeof(local_addr));
      if(bound < 0)
      {
            perror("Error:: Can not bind local IP address:");
            exit(1);
      }
        events.sctp_data_io_event=1;
        events.sctp_association_event = 1;
        events.sctp_address_event = 1;
        events.sctp_peer_error_event=1;
      int set = setsockopt(sd.sockfd, IPPROTO_SCTP, SCTP_EVENTS,
&events, sizeof(events));

      if(set < 0) perror("damn! setsockopt failed\n");
      printf("The value of the sockfd is %d\n",sd.sockfd);
      port = atoi(argv[2]);
      /* Initialize the destination address */
      bzero(&serv_addr, sizeof(struct sockaddr_in));
      serv_addr.sin_family = AF_INET;
      serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
      serv_addr.sin_port = htons(port);
      connected = connect(sd.sockfd, (struct sockaddr *) &serv_addr,
                        sizeof(serv_addr));

      if(connected < 0)
      {
```

```
            perror("Error:: Can not connect to server:");
            exit(1);
      }

      printf("Connected to server on port %d\n", port);

      pid = pthread_create(&hover_thrd, NULL, handoff, (void *) &sd);
        socklen_t len;
        size_t length;
        length = sizeof(buf);
        len = sizeof(struct sockaddr_in);

      while(strcmp(buf, "EOT") != 0)
      {
            recved = recv(sd.sockfd, (void *) buf, MAX_DATA_BUFFER, 0);
      }

      fclose(ofp);
      close(sd.sockfd);
      printf("Disconnecting from server\n");
      return 0;
}

#define MAX_ADAPTER_LEN 25
#define MAX_MAC_LEN 50
#define MAX_ESSID_LEN 100
//#define WLAN2_MAC "00:0D:88:A1:59:9F"
#define WLAN2_MAC "00:14:BF:E9:25:6D"
void *handoff(void *args)
{

        struct timeval time;
      struct sock_lock *sd = (struct sock_lock *) args;
        printf("The passed valus is %d\n",sd->sockfd);
      //  sleep(20);
      struct sockaddr_in new_addr;
      int addrlen = sizeof(struct sockaddr_in);
      struct sctp_setpeerprim new_prim;
//    struct sctp_setprim new_prim;
      int struct_setpeerprim_size = sizeof(struct sctp_setpeerprim);
//    int struct_setpeerprim_size = sizeof(new_prim);
      int addrcnt = 1;
      int bound;
      int sockopt_set;
      //char interfc_1[MAX_ADAPTER_LEN] = "eth0";
        char interfc_1[MAX_ADAPTER_LEN] = "wlan0";
      char interfc_2[MAX_ADAPTER_LEN] = "ppp0";
      char essid_1[MAX_ESSID_LEN];
      char essid_2[MAX_ESSID_LEN];
      char mac_1[MAX_MAC_LEN];
      char mac_2[MAX_MAC_LEN];
      char cur_ap_mac[MAX_MAC_LEN];
      int sig_strength_1;
      int sig_strength_2;
      int sock_1 = -1;
      int sock_2 = -1;
      int get_mac;
```

```
        int handover=0;
        double t2,t3;
      iwstats Stats;
        int SNR1, SNR2;
      int set1, set2, check1, check2;
      double time1, time2;
/*    char command_wlan_add[25];
      char command_wlan_del[25];
      char command_cdma_add[25];
      char command_cdma_del[25];


      bzero(command_wlan_add, 25);
      bzero(command_cdma_add, 25);
      bzero(command_cdma_del, 25);
      bzero(command_wlan_del, 25);

      strcpy(command_wlan_add, "./addIP ");
      strcpy(command_cdma_add, "./addIP ");
      strcpy(command_wlan_del, "./delIP ");
      strcpy(command_cdma_del, "./delIP ");


      strcat(command_wlan_add, wlan_addr);
      strcat(command_cdma_add, cdma_addr);
      strcat(command_wlan_del, wlan_addr);
      strcat(command_cdma_del, cdma_addr);
*/
      printf("Initializing sockets to monitor network activity\n");
      /* Open a socket to the kernel */
      if(SIG_MONITOR_FAILED == Sig_Initial_Socket(&sock_1))
            exit(1);

      printf("Sockets initialized\n");

      /* Get the name of our current AP */
      if(SIG_MONITOR_FAILED ==
Sig_Get_AP_MAC_Address(&sock_1,interfc_1,
                              mac_1))
      {
            printf("failed\n");
            exit(1);
      }


      printf("CDMA connection detected...\n");
                  /* New AP detected. Add static address to
                        association*/

                  bzero(&new_addr, addrlen);
                  new_addr.sin_family = AF_INET;
                  //new_addr.sin_addr.s_addr = inet_addr("10.1.6.2");
                  new_addr.sin_addr.s_addr = inet_addr(cdma_addr);
                  new_addr.sin_port = htons(local_port);
               //   sleep(20);
                  /* Lock the socket and bind new address */
                  pthread_mutex_lock(&sd->sock_mutex);
```

```
                  bound = sctp_bindx(sd->sockfd,
                        (struct sockaddr *) &new_addr, addrcnt,
                        SCTP_BINDX_ADD_ADDR);
                  pthread_mutex_unlock(&sd->sock_mutex);

                  if(bound < 0)
                  {
                        perror("Error:: Could not bind new address
2:");
                        exit(1);
                  }
                  else
                        printf("New address bound\n");
      printf("Im here\n");
        printf("The passed valus is %d\n",sd->sockfd);
     if(Sig_Get_CurrentAP_Stats(interfc_1, &Stats)<0)
         printf("Error in getting AP stats\n");
      else
      SNR1 = display(interfc_1, &Stats);
/*
               printf("performing add ip\n");
               system(command_wlan_add);

               printf("performing nslookup\n");
               system("nslookup trash1.trash.ou.edu");
*/

while(1)  {  //Loop for the ping pong movement
      /* Loop and compare signal strength of WLAN */

      printf("beginning of the loop.......\n");


      while(1)
      {

            check1 = 1;

            while(SNR1 <  SNR_HYST)
            {
                  if (check1 == 1)
                  {
                        //get time here
                        time1 = gettime();
                        check1 = 0;
                  }

                  // get time again
                  time2 = gettime();

                  //printf("time1:%f\ttime2:%f\ttimediff:%f\n", time1,
time2, (time2-time1)); // add by sayeem

                  if ((time2 - time1) > TRTIME)
                  {
                        set1 = 1;
                        break;
```

```
                  }

                        count2++;
                        if(count2==15000)
                  {
                              //printf("DATA THROUGH OLD PATH\n  "); //
add by suren
                              printf("In network 1 (waiting on
2)...SNR1:%d\tTimediff:%f\n", SNR1, (time2-time1)); // add by sayeem
                              count2=0;
                        }

                        if(Sig_Get_CurrentAP_Stats(interfc_1,
&Stats)<0)
                        printf("Error in getting AP1 stats/n");
                        else
                              SNR1 = display(interfc_1, &Stats);


            }

            if (set1 == 1)
            {
                  set1 = 0;
                  break;
            }



                    count2++;
                    if(count2==15000){
                  //printf("DATA THROUGH OLD PATH\n  "); // add by
suren
                  //printf("In network 1.........\n  "); // add by
suren
                        printf("In network 1...SNR1:%d\n", SNR1); //
add by sayeem
                        count2=0;
                    }




                  if(Sig_Get_CurrentAP_Stats(interfc_1, &Stats)<0)
                  printf("Error in getting AP1 stats/n");
                  else
                        SNR1 = display(interfc_1, &Stats);


      }


  struct sctp_status temp;
    socklen_t len4 = sizeof(temp);
    memset(&temp, 0, len4);
    temp.sstat_assoc_id = 12;
```

```
    if(getsockopt(sd->sockfd, IPPROTO_SCTP, SCTP_STATUS, &temp,
&len4)<0) {
     printf("assoc_id %x is not correct\n",ass_id);
     exit(1);
      }

      new_prim.sspp_assoc_id = NULL;
      memcpy(&new_prim.sspp_addr, &new_addr, addrlen);


      t2 = gettime();
      pthread_mutex_lock(&sd->sock_mutex);
      sockopt_set = setsockopt(sd->sockfd, IPPROTO_SCTP,
                        SCTP_SET_PEER_PRIMARY_ADDR,
                        (const void *) &new_prim,
                        struct_setpeerprim_size);
      pthread_mutex_unlock(&sd->sock_mutex);

      if(sockopt_set < 0)
      {
            perror("Error:: Unable to set primary:");
            exit(1);
      }
      else
      {


            printf("New primary set\n");

                handover1 = 1;
            system("route -n");
            /* Set the new route in the routing table
                  This will be added later after the set-peer-prim
                  crash on the cn is fixed */

                system("echo 0 > /proc/sys/net/ipv4/route/min_delay");
                system("echo 0 > /proc/sys/net/ipv4/route/max_delay");
                system("echo 1 > /proc/sys/net/ipv4/route/flush");


            system("route del -net 10.1.8.0 netmask 255.255.255.0 dev
wlan0");
            system("route del -net 169.254.0.0 netmask 255.255.0.0 dev
wlan0");
            system("route del -net 0.0.0.0 netmask 0.0.0.0 dev wlan0");
            system("route add -net 0.0.0.0 netmask 0.0.0.0 gw
68.28.177.69");


                system("echo 1 > /proc/sys/net/ipv4/route/flush");
                system("echo 2 > /proc/sys/net/ipv4/route/min_delay");
                system("echo 10 > /proc/sys/net/ipv4/route/max_delay");
           /*     system("ip route flush cache");
                system("ip route flush cache 129.15.78.0");
                system("ip route flush cache 129.15.78.0");
                system("ip route flush cache");
                system("ip route show cache");  */
```

```
                system("route -n");

/*          printf("performing add ip\n");
            system(command_cdma_add);

            printf("performing nslookup\n");
            system("nslookup trash1.trash.ou.edu");


            printf("performing delete ip\n");
                system(command_wlan_del);

                printf("performing nslookup\n");
                system("nslookup trash1.trash.ou.edu");
*/

              //  sleep(1);
            t3 = gettime();
                t1=t3-t2;
                printf("Time taken for handoff: %f\n",t2);
      }

             if(Sig_Get_CurrentAP_Stats(interfc_1, &Stats)<0)
         printf("Error in getting AP stats/n");
      else
      SNR1 = display(interfc_1, &Stats);



      while(1)
      {

            check2 = 1;

            while(SNR1 >  SNR_HYST)
            {
                  if (check2 == 1)
                  {
                        //get time here
                        time1 = gettime();
                        check2 = 0;
                  }

                  // get time again
                  time2 = gettime();

                  //printf("time1:%f\ttime2:%f\ttimediff:%f\n", time1,
time2, (time2-time1)); // add by sayeem

                  if ((time2 - time1) > TRTIME)
                  {
                        set2 = 1;
                        break;
                  }

                        count2++;
                        if(count2==15000)
```

```
                  {
                              //printf("DATA THROUGH OLD PATH\n  "); //
add by suren
                              printf("In network 2 (waiting on
1)...SNR1:%d\tTimediff:%f\n", SNR1, (time2-time1)); // add by sayeem
                              count2=0;
                        }

                        if(Sig_Get_CurrentAP_Stats(interfc_1,
&Stats)<0)
                        printf("Error in getting AP1 stats/n");
                        else
                              SNR1 = display(interfc_1, &Stats);


            }

            if (set2 == 1)
            {
                  set2 = 0;
                  break;
            }


                    count2++;
                    if(count2==15000){
                  //printf("DATA THROUGH OLD PATH\n  "); // add by
suren
                  //printf("In network 1.........\n  "); // add by
suren
                        printf("In network 2...SNR1:%d\n", SNR1); //
add by sayeem
                        count2=0;
                    }




                  if(Sig_Get_CurrentAP_Stats(interfc_1, &Stats)<0)
                  printf("Error in getting AP1 stats/n");
                  else
                        SNR1 = display(interfc_1, &Stats);


      }



#define MAX_ADAPTER_LEN 25
#define MAX_MAC_LEN 50
#define MAX_ESSID_LEN 100
//#define WLAN1_MAC "00:0D:88:88:41:A3"
#define WLAN1_MAC "00:11:95:52:FA:B1"

      if(SIG_MONITOR_FAILED ==
Sig_Get_AP_MAC_Address(&sock_1,interfc_1,
                              mac_1))
```

```
            exit(1);
//    while(1)
//    {
            get_mac = Sig_Get_AP_MAC_Address(&sock_1, interfc_1,
mac_1);
            if(get_mac != SIG_MONITOR_FAILED &&
               strcmp(mac_1, WLAN1_MAC) == 0)
            {

                  printf("WLAN AP detected\n");
                  /* New AP detected. Add static address to
                        association*/

                  bzero(&new_addr, addrlen);
                  new_addr.sin_family = AF_INET;
                  new_addr.sin_addr.s_addr = inet_addr(wlan_addr);
                  new_addr.sin_port = htons(local_port);

                  /* Lock the socket and bind new address */
        /*                 printf("Im stuck in the  mutex -1\n");
                  pthread_mutex_lock(&sd->sock_mutex);
                        printf("Im stuck in the  mutex -2\n");
                  bound = sctp_bindx(sd->sockfd,
                        (struct sockaddr *) &new_addr, addrcnt,
                        SCTP_BINDX_ADD_ADDR);
                  pthread_mutex_unlock(&sd->sock_mutex);

                  if(bound < 0)
                  {
                        perror("Error 2:: Could not bind new
address:");
                        exit(1);
                  }
                  else
                        printf("New-2 address bound\n");
                  break;*/
            }
//    }

      new_prim.sspp_assoc_id = NULL;
      memcpy(&new_prim.sspp_addr, &new_addr, addrlen);

      t2 = gettime();
      pthread_mutex_lock(&sd->sock_mutex);
      sockopt_set = setsockopt(sd->sockfd, IPPROTO_SCTP,
                        SCTP_SET_PEER_PRIMARY_ADDR,
                        (const void *) &new_prim,
                        struct_setpeerprim_size);
      pthread_mutex_unlock(&sd->sock_mutex);

      if(sockopt_set < 0)
      {
            perror("Error 2 :: Unable to set primary:");
            exit(1);
      }
      else
      {
```

```
            printf("New-2 Second primary set\n");
                handover1 = 0;
                system("route -n");
            /* Set the new route in the routing table
                  This will be added later after the set-peer-prim
                  crash on the cn is fixed */

                system("echo 0 > /proc/sys/net/ipv4/route/min_delay");
                system("echo 0 > /proc/sys/net/ipv4/route/max_delay");
                system("echo 1 > /proc/sys/net/ipv4/route/flush");


                system("route del -net 0.0.0.0 netmask 0.0.0.0 gw
68.28.177.69");
                system("route add -net 10.1.8.0 netmask 255.255.255.0
dev wlan0");
                system("route add -net 0.0.0.0 netmask 0.0.0.0 gw
10.1.8.1");


            system("echo 1 > /proc/sys/net/ipv4/route/flush");
                system("echo 2 > /proc/sys/net/ipv4/route/min_delay");
                system("echo 10 > /proc/sys/net/ipv4/route/max_delay");
           /*   system("ip route flush cache 129.15.78.0");
                system("ip route flush cache 129.15.78.0");
                system("ip route flush cache");
                system("ip route flush cache");
                system("ip route show cache"); */
                system("route -n");

               /* printf("performing add ip\n");
                system(command_wlan_add);

                printf("performing nslookup\n");
                system("nslookup trash1.trash.ou.edu");

            printf("performing delete ip\n");
                system(command_cdma_del);

                printf("performing nslookup\n");
                system("nslookup trash1.trash.ou.edu");

*/
            t3 = gettime();
                t1=t3-t2;
                printf("Time taken for handoff: %f\n",t1);
      }

} //main while
      return 0;
}


int display(char* InterfaceName, iwstats* Stats)
{
```

```
  int SNR;
// printf("Noise level %d dBm\n", Stats->qual.noise - 0x100);
// printf("The Signal to Noise Ratio is %d \n",((Stats->qual.level -
0x100) - (Stats->qual.noise - 0x100)));
SNR = (Stats->qual.level - 0x100) - (Stats->qual.noise - 0x100);
//printf("%d %d\n",count,Stats->qual.level - 0x100);
 val[count]=SNR;
count++;
 if(count%200 == 0) {sd(val);count=0;}
return(SNR);
}

int sd(int *val)
{
 double N=200,sum=0,mean=0,square=0;
 int i,j,k,l;
 double sd;
//val[0]=5;val[1]=6;val[2]=8;val[3]=9;
 for(i=0;i<N;i++)
 {
//   printf("Value %f\n",val[i]);
   sum = sum + val[i];
 }
//printf("The sum is %f\n",sum);
 mean = sum/N;
//printf("The mean is %f\n",mean);
 for(i=0;i<N;i++)
{
 square = square + ((val[i]-mean)*(val[i]-mean));
}
//printf("the sum of squares %f\n",square);
//hyst1 = sd = sqrt(square/N); //commented by suren on mqy19
hyst1 = (int)sqrt(square/N);
//printf("The standard deviation is %f\n",sd);
//printf("%d %f\n",count1,sd);
//count1++;
}



int display1(char* InterfaceName, iwstats* Stats)
{
  int SNR;
// printf("Noise level %d dBm\n", Stats->qual.noise - 0x100);
// printf("The Signal to Noise Ratio is %d \n",((Stats->qual.level -
0x100) - (Stats->qual.noise - 0x100)));
SNR = (Stats->qual.level - 0x100) - (Stats->qual.noise - 0x100);
//printf("%d %d\n",count,Stats->qual.level - 0x100);
 val1[count]=SNR;
count1++;
 if(count1%200 == 0) {sd1(val1);count1=0;}
return(SNR);
}

int sd1(int *val1)
{
 double N=200,sum=0,mean=0,square=0;
```

```
 int i,j,k,l;
 double sd;
//val[0]=5;val[1]=6;val[2]=8;val[3]=9;
 for(i=0;i<N;i++)
 {
//   printf("Value %f\n",val[i]);
   sum = sum + val[i];
 }
//printf("The sum is %f\n",sum);
 mean = sum/N;
//printf("The mean is %f\n",mean);
 for(i=0;i<N;i++)
{
 square = square + ((val[i]-mean)*(val[i]-mean));
}
//printf("the sum of squares %f\n",square);
hyst2 = (int)sqrt(square/N);
//printf("The standard deviation is %f\n",sd);
//printf("%d %f\n",count1,sd);
//count1++;
}

void getaddy()

{

  unsigned char      *u;

  int                sockfd, size  = 1;

  struct ifreq       *ifr;

  struct ifconf      ifc;

  struct sockaddr_in sa;

  char addy[11], init_addy[11], new_addy[11], temp[4];

  int rng_init, rng_end;

      FILE *fp, *fpd;

    int i;




  if (0 > (sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_IP))) {

   fprintf(stderr, "Cannot open socket.\n");

    exit(EXIT_FAILURE);

  }
```

```
ifc.ifc_len = IFRSIZE;
ifc.ifc_req = NULL;

do {
  ++size;
  /* realloc buffer size until no overflow occurs  */
  if (NULL == (ifc.ifc_req = realloc(ifc.ifc_req, IFRSIZE))) {
    fprintf(stderr, "Out of memory.\n");
    exit(EXIT_FAILURE);
  }
  ifc.ifc_len = IFRSIZE;
  if (ioctl(sockfd, SIOCGIFCONF, &ifc)) {
    perror("ioctl SIOCFIFCONF");
    exit(EXIT_FAILURE);
  }
} while  (IFRSIZE <= ifc.ifc_len);

ifr = ifc.ifc_req;
for (;(char *) ifr < (char *) ifc.ifc_req + ifc.ifc_len; ++ifr) {

  if (ifr->ifr_addr.sa_data == (ifr+1)->ifr_addr.sa_data) {
    continue;  /* duplicate, skip it */
  }

  if (ioctl(sockfd, SIOCGIFFLAGS, ifr)) {
    continue;  /* failed to get flags, skip it */
  }
```

```
            if (strcmp(ifr->ifr_name, "wlan0") == 0)

            {

                  printf("Interface:  %s\n", ifr->ifr_name);

                  strcpy(wlan_addr,
inet_ntoa(inaddrr(ifr_addr.sa_data)));
                  printf("IP Address: %s\n", wlan_addr);
            }

            else if (strcmp(ifr->ifr_name, "ppp0") == 0)

            {

                  printf("Interface:  %s\n", ifr->ifr_name);

                  strcpy(cdma_addr,
inet_ntoa(inaddrr(ifr_addr.sa_data)));
                  printf("IP Address: %s\n", cdma_addr);
            }


  }



  close(sockfd);

}


double gettime()
{
        struct timeval time;
        double t1, t2, t3;
        gettimeofday(&time, NULL);
        t2=(double)time.tv_sec;
        t3=(double)time.tv_usec/(double)1000000;
        t1=t2+t3;
        return t1;
}
```

sig_monitor.c

```
/*
*************************************************************************
*****************
* sig_monitor.c version 2.0 - API to wireless signal quality based on
MAPI.c from  Moustafa A. Youssef
*
* Author Lu,Song
*
```

```
*
*
*************************************************************************
********************/

#include "sig_monitor.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIG_MONITOR_DEBUG

extern FILE *stderr;

int Sig_Initial_Socket(int * p_iSocket)
{

  if((*p_iSocket = sockets_open()) < 0)
  {
    fprintf(stderr, "Error Opening socket to the networking kernel!");
    return SIG_MONITOR_FAILED;
  }
  return SIG_MONITOR_SUCCESS;

}

int Sig_Monitor_Strength(int * p_iSocket, char * p_sInterfaceName)
{
  wireless_info Info;

      memset((char *) &Info, 0, sizeof(struct wireless_info));

      // Get ranges
      if(get_range_info(*p_iSocket, p_sInterfaceName, &(Info.range)) >=
0)
  {

      if(Sig_Get_CurrentAP_Stats(p_sInterfaceName, &(Info.stats)) ==
SIG_MONITOR_SUCCESS)
        return Info.stats.qual.level-0x100;
      else
        return SIG_MONITOR_FAILED;
  }
  else
    return SIG_MONITOR_FAILED;

}



int Sig_Get_AP_MAC_Address(int * p_iSocket, char * p_sInterfaceName,
                           char * p_sMac_Address)
{
      struct iwreq wrq;

      //memset((char *) Info, 0, sizeof(struct wireless_info));
```

```
      /* Get wireless name */
      strcpy(wrq.ifr_name, p_sInterfaceName);

      /* If no wireless name : no wireless extensions */
      if(ioctl(* p_iSocket, SIOCGIWNAME, &wrq) < 0)
            return SIG_MONITOR_FAILED;

      strcpy(wrq.ifr_name, p_sInterfaceName);

      /* Get AP address */
      if(ioctl(* p_iSocket, SIOCGIWAP, &wrq) >= 0)
      {
            sockaddr Addr;
            memcpy(&Addr, &wrq.u.ap_addr, sizeof(sockaddr));
            strcpy(p_sMac_Address,pr_ether((unsigned
char*)&Addr.sa_data));
            return SIG_MONITOR_SUCCESS;
      }
      return SIG_MONITOR_FAILED;

}


int Sig_Get_AP_ESSID(int * p_iSocket, char * p_sInterfaceName, char *
p_sEssid)
{
      struct iwreq wrq;
  wireless_info Info;
      strcpy(wrq.ifr_name, p_sInterfaceName);
      wrq.u.essid.pointer = (caddr_t) Info.essid;
      wrq.u.essid.length = 0;
      wrq.u.essid.flags = 0;
      if(ioctl(* p_iSocket, SIOCGIWESSID, &wrq) >= 0)
  {
            Info.has_essid = 1;
            Info.essid_on = wrq.u.data.flags;

    strncpy(p_sEssid, Info.essid, strlen(Info.essid)+1);
    return SIG_MONITOR_SUCCESS;
  }
  else
    return SIG_MONITOR_FAILED;
}

/*
int Sig_Layer_2_Handover(int * p_iSocket, char * p_sFrom, char * p_sTo)
{
    char s_New_Essid[255], s_Handover_Command[255];
    if (SIG_MONITOR_SUCCESS!=Sig_Get_AP_ESSID(*p_iSocket, p_sTo,
s_New_Essid))
      {
      printf("Error in Sig_Layer_2_Handover_1!\n");
      return SIG_MONITOR_FAILED;
    }
    sprintf(s_Handover_Command,"iwconfig %s essid %s", p_sFrom,
s_New_Essid);
    if  ( -1==system(s_Handover_Command))
```

```
      {
      printf("Error in Sig_Layer_2_Handover_2!\n");
      return SIG_MONITOR_FAILED;
    }
}
*/

int Sig_Get_CurrentAP_Stats(char* p_sInterfaceName, iwstats*
      Stats)
{
   FILE *   f=fopen("/proc/net/wireless","r");
   char           buf[256];
   char *   bp;
   int            t;
   if(f==NULL)
   return SIG_MONITOR_FAILED;
   /* Loop on all devices */
   while(fgets(buf,255,f))
   {
      bp=buf;
      while(*bp&&isspace(*bp))                              /* MOS:
skip blanks */
      bp++;
      /* Is it the good device ? */
      if(strncmp(bp, p_sInterfaceName, strlen(p_sInterfaceName))==0 &&
bp[strlen(p_sInterfaceName)]==':')
      {
        /* Skip ethX: */
        bp=strchr(bp,':');
        bp++;
        /* -- status -- */
        bp = strtok(bp, " ");                           /* MOS: tokenize
by space */
        sscanf(bp, "%X", &t);
        Stats->status = (unsigned short) t;
        /* -- link quality -- */
        bp = strtok(NULL, " ");
        if(strchr(bp,'.') != NULL)                      /* MOS: '.' means
updated */
            Stats->qual.updated |= 1;
        sscanf(bp, "%d", &t);
        Stats->qual.qual = (unsigned char) t;
        /* -- signal level -- */
        bp = strtok(NULL, " ");
        if(strchr(bp,'.') != NULL)
            Stats->qual.updated |= 2;
        sscanf(bp, "%d", &t);
        Stats->qual.level = (unsigned char) t;
        /* -- noise level -- */
        bp = strtok(NULL, " ");
        if(strchr(bp,'.') != NULL)
            Stats->qual.updated += 4;
        sscanf(bp, "%d", &t);
        Stats->qual.noise = (unsigned char) t;
        /* -- discarded packets -- */
        bp = strtok(NULL, " ");
        sscanf(bp, "%d", &Stats->discard.nwid);
```

```
        bp = strtok(NULL, " ");
        sscanf(bp, "%d", &Stats->discard.code);
        bp = strtok(NULL, " ");
        sscanf(bp, "%d", &Stats->discard.misc);
        fclose(f);
        return SIG_MONITOR_SUCCESS;
      }
  }
  fclose(f);
  return SIG_MONITOR_FAILED;
}


int Sig_Close_Socket(int * p_iSocket)
{
  close(* p_iSocket);
}
```

sig_monitor.h

```
/*
**************************************************************************
*****************
* sig_monitor.c version 3.0 - API to monitor wireless signal quality
* and implement layer2 handover  based on MAPI.c from  Moustafa A.
Youssef
*
* Author Lu,Song lusong@ou.ed
* CopyRight All Right Reserved.
*
* Acknowledgement: I had learned lots of things from Mr. Moustafa A.
Youssef's code, mapi.h
*   Thanks a lot.
*
**************************************************************************
********************/

#ifndef SIG_MONITOR_H
#define SIG_MONITOR_H

#define SIG_MONITOR_SUCCESS  0
#define SIG_MONITOR_FAILED   -1

#ifdef __cplusplus
extern "C" {
#endif

// wireless extension interface
#include "iwcommon.h"

/**************************************************************************
****
Function: Sig_Initial_Socket
```

```
Return Value: int
      SIG_MONITOR_SUCCESS for Success
      SIG_MONITOR_FAILED for failed
Parameters:
      int * p_iSocket: Pointer to Socket descripter
Description: Initial Socket, the very first step.
**********************************************************************
****/
int Sig_Initial_Socket(int * p_iSocket);

/**********************************************************************
****
Function: Sig_Monitor_Strength
Return Value: int
      Non SIG_MONITOR_FAILED: signal strength
      SIG_MONITOR_FAILED for failed
Parameters:
      int * p_iSocket: Pointer to Socket descripter
      char * p_sInterfaceName: Interface Name
Description: Get the interface's signal strength

**********************************************************************
****/
int Sig_Monitor_Strength(int * p_iSocket, char * p_sInterfaceName);

/**********************************************************************
****
Function: Sig_Get_AP_MAC_Address
Return Value: int
      SIG_MONITOR_SUCCESS for Success
      SIG_MONITOR_FAILED for failed
Parameters:
      int * p_iSocket: Pointer to Socket descripter
      char * p_sInterfaceName: Interface Name
      char * p_sMac_Address: string of MAC address
Description: fill the MAC address of current AP connected
      into string p_sMac_Address
**********************************************************************
****/
int Sig_Get_AP_MAC_Address(int * p_iSocket, char * p_sInterfaceName,
char * p_sMac_Address);

/**********************************************************************
****
Function: Sig_Get_AP_ESSID
Return Value: int
      SIG_MONITOR_SUCCESS for Success
      SIG_MONITOR_FAILED for failed
Parameters:
      int * p_iSocket: Pointer to Socket descripter
      char * p_sInterfaceName: Interface Name
      char * p_sEssid: string of ESSID to be filled
Description: fill the ESSID of current AP connected
      into string p_sEssid
**********************************************************************
****/
```

```
int Sig_Get_AP_ESSID(int * p_iSocket, char * p_sInterfaceName, char *
p_sEssid);

/*********************************************************************
****
Function: Sig_Layer_2_Handover
Return Value: int
      SIG_MONITOR_SUCCESS for Success
      SIG_MONITOR_FAILED for failed
Parameters:
      int * p_iSocket: Pointer to Socket descripter
      char * p_sFrom: Interface which should handover from
      char * p_sTo: Interface which should handover to
Description: let interface p_sFrom handover to another domain monitored
by
      p_sTo.
*********************************************************************
****/
int Sig_Layer_2_Handover(int * p_iSocket, char * p_sFrom, char *
p_sTo);


/*********************************************************************
****
Function: Sig_Get_CurrentAP_Stats
Return Value: int
      SIG_MONITOR_SUCCESS for Success
      SIG_MONITOR_FAILED for failed
Parameters:
      char * p_sFrom: Interface which should handover from
      char * p_sTo: Interface which should handover to
Description: Get current AP status which is connecting the interface by
      p_sInterfaceName.
      This function is directly copied from mapi.c by Moustafa A.
Youssef.
*********************************************************************
****/
int Sig_Get_CurrentAP_Stats(char* p_sInterfaceName, iwstats*
      Stats);


/*********************************************************************
****
Function: Sig_Close_Socket
Return Value: int
      SIG_MONITOR_SUCCESS for Success
      SIG_MONITOR_FAILED for failed
Parameters:
      int * p_iSocket: Pointer to Socket descripter

*********************************************************************
****/
int Sig_Close_Socket(int * p_iSocket);

#endif

#ifdef __cplusplus
```

```
}
#endif
```

iwcommon.c

```
/*
 *    Wireless Tools
 *
 *          Jean II - HPLB '99
 *
 * Common header for the wireless tools...
 */

#ifndef IWCOMMON_H
#define IWCOMMON_H

#ifdef __cplusplus
extern "C" {
#endif

/************************** DOCUMENTATION **************************/
/*
 * None ? Todo...
 */

/* ---------------------------- HISTORY ---------------------------- */
/*
 * wireless 16 :        (Jean Tourrilhes)
 * -----------
 *    o iwconfig, iwpriv & iwspy
 *
 * wireless 17 :        (Justin Seger)
 * -----------
 *    o Compile under glibc fix
 *    o merge iwpriv in iwconfig
 *    o Add Wavelan roaming support
 *    o Update man page of iwconfig
 *
 * wireless 18 :
 * -----------
 *          (From Andreas Neuhaus <andy@fasta.fh-dortmund.de>)
 *    o Many fix to remove "core dumps" in iwconfig
 *    o Remove useless headers in iwconfig
 *    o CHAR wide private ioctl
 *          (From Jean Tourrilhes)
 *    o Create iwcommon.h and iwcommon.c
 *    o Separate iwpriv again for user interface issues
 *      The folllowing didn't make sense and crashed :
 *          iwconfig eth0 priv sethisto 12 15 nwid 100
 *    o iwspy no longer depend on net-tools-1.2.0
 *    o Reorganisation of the code, cleanup
```

```
 *    o Add ESSID stuff in iwconfig
 *    o Add display of level & noise in dBm (stats in iwconfig)
 *    o Update man page of iwconfig and iwpriv
 *    o Add xwireless (didn't check if it compiles)
 *          (From Dean W. Gehnert <deang@tpi.com>)
 *    o Minor fixes
 *          (Jan Rafaj <rafaj@cedric.vabo.cz>)
 *    o Cosmetic changes (sensitivity relative, freq list)
 *    o Frequency computation on double
 *    o Compile clean on libc5
 *          (From Jean Tourrilhes)
 *    o Move listing of frequencies to iwspy
 *    o Add AP address stuff in iwconfig
 *    o Add AP list stuff in iwspy
 *
 * wireless 19 :
 * -----------
 *          (From Jean Tourrilhes)
 *    o Allow for sensitivity in dBm (if < 0) [iwconfig]
 *    o Formatting changes in displaying ap address in [iwconfig]
 *    o Slightly improved man pages and usage display
 *    o Add channel number for each frequency in list [iwspy]
 *    o Add nickname... [iwconfig]
 *    o Add "port" private ioctl shortcut [iwpriv]
 *    o If signal level = 0, no range or dBms [iwconfig]
 *    o I think I now got set/get char strings right in [iwpriv]
 *          (From Thomas Ekstrom <tomeck@thelogic.com>)
 *    o Fix a very obscure bug in [iwspy]
 *
 * wireless 20 :
 * -----------
 *          (From Jean Tourrilhes)
 *    o Remove all #ifdef WIRELESS ugliness, but add a #error :
 *          we require Wireless Extensions 9 or nothing !  [all]
 *    o Switch to new 'nwid' definition (specific -> iw_param)
[iwconfig]
 *    o Rewriten totally the encryption support [iwconfig]
 *          - Multiple keys, through key index
 *          - Flexible/multiple key size, and remove 64bits upper limit
 *          - Open/Restricted modes
 *          - Enter keys as ASCII strings
 *    o List key sizes supported and all keys in [iwspy]
 *    o Mode of operation support (ad-hoc, managed...) [iwconfig]
 *    o Use '=' to indicate fixed instead of ugly '(f)' [iwconfig]
 *    o Ability to disable RTS & frag (off), now the right way
[iwconfig]
 *    o Auto as an input modifier for bitrate [iwconfig]
 *    o Power Management support [iwconfig]
 *          - set timeout or period and its value
 *          - Reception mode (unicast/multicast/all)
 *    o Updated man pages with all that ;-)
 */

/* ---------------------------- TODO ---------------------------- */
/*
 * One day, maybe...
 *
```

```
 * iwconfig :
 * --------
 *    Make disable a per encryption key modifier if some hardware
 *    requires it.
 *    Should not mention "Access Point" but something different when
 *    in ad-hoc mode.
 *
 * iwpriv :
 * ------
 *    Remove 'port' and 'roam' cruft now that we have mode in iwconfig
 *
 * iwspy :
 * -----
 *    ?
 *
 * Doc & man pages :
 * ---------------
 *    Update main doc.
 *
 * Other :
 * -----
 *    What about some graphical tools ?
 */

/*************************** INCLUDES ***************************/

/* Standard headers */
#include <sys/types.h>
#include <sys/ioctl.h>
#include <stdio.h>
#include <math.h>
#include <errno.h>
#include <fcntl.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>            /* gethostbyname, getnetbyname */

/* This is our header selection. Try to hide the mess and the misery :-
(
 * Please choose only one of the define...
 */
/* Kernel headers 2.0.X + Glibc 2.0 - Debian 2.0, RH5
 * Kernel headers 2.2.X + Glibc 2.1 - Debian 2.2, RH6.1 */
#define GLIBC_HEADERS

/* Kernel headers 2.2.X + Glibc 2.0 - Debian 2.1 */
#undef KLUDGE_HEADERS

/* Kernel headers 2.0.X + libc5 - old systems */
#undef LIBC5_HEADERS

#ifdef KLUDGE_HEADERS
#include <socketbits.h>
#endif      /* KLUDGE_HEADERS */
```

```
#if defined(KLUDGE_HEADERS) || defined(GLIBC_HEADERS)
#include <linux/if_arp.h>     /* For ARPHRD_ETHER */
#include <linux/socket.h>     /* For AF_INET & struct sockaddr */
/* mos
#include <linux/in.h>
*/
#endif      /* KLUDGE_HEADERS || GLIBC_HEADERS */

#ifdef LIBC5_HEADERS
#include <sys/socket.h>       /* For AF_INET & struct sockaddr &
socket() */
#include <linux/if_arp.h>     /* For ARPHRD_ETHER */
#include <linux/in.h>         /* For struct sockaddr_in */
#endif      /* LIBC5_HEADERS */

/* Wireless extensions */
// modified by Lusong
//#include "/usr/src/linux/include/linux/wireless.h"
//#include "/lib/modules/2.6.9-1.667/build/include/linux/wireless.h"
#include "wireless.h"

#if WIRELESS_EXT < 8
#error "Wireless Extension v9 or newer required :-(\n\
Use Wireless Tools v19 or update your kernel headers"
#endif

/**************************** DEBUG ****************************/


/********************** CONSTANTS & MACROS **********************/

/* Some usefull constants */
#define KILO      1e3
#define MEGA      1e6
#define GIGA      1e9

/***************************** TYPES *****************************/

/* Shortcuts */
typedef struct iw_statistics  iwstats;
typedef struct iw_range       iwrange;
typedef struct iw_param       iwparam;
typedef struct iw_freq        iwfreq;
typedef struct iw_priv_args   iwprivargs;
typedef struct sockaddr       sockaddr;

/* Structure for storing all wireless information for each device */
typedef struct wireless_info
{
  char            name[IFNAMSIZ];         /* Wireless/protocol name */
  int       has_nwid;
  iwparam   nwid;             /* Network ID */
  int       has_freq;
  float           freq;             /* Frequency/channel */
  int       has_sens;
  iwparam   sens;             /* sensitivity */
  int       has_key;
```

```
  unsigned char   key[IW_ENCODING_TOKEN_MAX];   /* Encoding key used */
  int       key_size;         /* Number of bytes */
  int       key_flags;        /* Various flags */
  int       has_essid;
  int       essid_on;
  char            essid[IW_ESSID_MAX_SIZE + 1]; /* ESSID (extended
network) */
  int       has_nickname;
  char            nickname[IW_ESSID_MAX_SIZE + 1]; /* NickName */
  int       has_ap_addr;
  sockaddr  ap_addr;          /* Access point address */
  int       has_bitrate;
  iwparam   bitrate;          /* Bit rate in bps */
  int       has_rts;
  iwparam   rts;              /* RTS threshold in bytes */
  int       has_frag;
  iwparam   frag;             /* Fragmentation threshold in bytes */
  int       has_mode;
  int       mode;             /* Operation mode */
  int       has_power;
  iwparam   power;                /* Power management parameters */

  /* Stats */
  iwstats   stats;
  int       has_stats;
  iwrange   range;
  int       has_range;
} wireless_info;

/************************** PROTOTYPES **************************/
/*
 * All the functions in iwcommon.c
 */
/* ---------------------- SOCKET SUBROUTINES ----------------------*/
int
      sockets_open(void);
/* --------------------- WIRELESS SUBROUTINES ---------------------*/
int
      get_range_info(int            skfd,
                     char *         ifname,
                     iwrange *  range);
int
      get_priv_info(int       skfd,
                    char *            ifname,
                    iwprivargs *      priv);
/* -------------------- FREQUENCY SUBROUTINES -------------------- */
void
      float2freq(double in,
                 iwfreq * out);
double
      freq2float(iwfreq *     in);
/* --------------------- ADDRESS SUBROUTINES ---------------------- */
int
      check_addr_type(int     skfd,
                      char *      ifname);
char *
      pr_ether(unsigned char *ptr);
```

```
int
      in_ether(char *bufp, struct sockaddr *sap);
int
      in_inet(char *bufp, struct sockaddr *sap);
int
      in_addr(int       skfd,
            char *            ifname,
            char *            bufp,
            struct sockaddr *sap);
/* ------------------------ MISC SUBROUTINES ------------------------ */
int
      byte_size(int           args);

/**************************** VARIABLES ****************************/

#ifdef __cplusplus
}
#endif


#endif      /* IWCOMMON_H */
```

wireless.h

```
/*
 * This file define a set of standard wireless extensions
 *
 * Version :      16    2.4.03
 *
 * Authors :      Jean Tourrilhes - HPL - <jt@hpl.hp.com>
 * Copyright (c) 1997-2002 Jean Tourrilhes, All Rights Reserved.
 */

#ifndef _LINUX_WIRELESS_H
#define _LINUX_WIRELESS_H

/************************** DOCUMENTATION **************************/
/*
 * Initial APIs (1996 -> onward) :
 * -----------------------------
 * Basically, the wireless extensions are for now a set of standard
ioctl
 * call + /proc/net/wireless
 *
 * The entry /proc/net/wireless give statistics and information on the
 * driver.
 * This is better than having each driver having its entry because
 * its centralised and we may remove the driver module safely.
 *
 * Ioctl are used to configure the driver and issue commands.  This is
 * better than command line options of insmod because we may want to
 * change dynamically (while the driver is running) some parameters.
 *
 * The ioctl mechanimsm are copied from standard devices ioctl.
```

```
 * We have the list of command plus a structure descibing the
 * data exchanged...
 * Note that to add these ioctl, I was obliged to modify :
 *    # net/core/dev.c (two place + add include)
 *    # net/ipv4/af_inet.c (one place + add include)
 *
 * /proc/net/wireless is a copy of /proc/net/dev.
 * We have a structure for data passed from the driver to
/proc/net/wireless
 * Too add this, I've modified :
 *    # net/core/dev.c (two other places)
 *    # include/linux/netdevice.h (one place)
 *    # include/linux/proc_fs.h (one place)
 *
 * New driver API (2002 -> onward) :
 * -------------------------------
 * This file is only concerned with the user space API and common
definitions.
 * The new driver API is defined and documented in :
 *    # include/net/iw_handler.h
 *
 * Note as well that /proc/net/wireless implementation has now moved in
:
 *    # include/linux/wireless.c
 *
 * Wireless Events (2002 -> onward) :
 * --------------------------------
 * Events are defined at the end of this file, and implemented in :
 *    # include/linux/wireless.c
 *
 * Other comments :
 * --------------
 * Do not add here things that are redundant with other mechanisms
 * (drivers init, ifconfig, /proc/net/dev, ...) and with are not
 * wireless specific.
 *
 * These wireless extensions are not magic : each driver has to provide
 * support for them...
 *
 * IMPORTANT NOTE : As everything in the kernel, this is very much a
 * work in progress. Contact me if you have ideas of improvements...
 */

/************************** INCLUDES **************************/

/* To minimise problems in user space, I might remove those headers
 * at some point. Jean II */
#include <linux/types.h>            /* for "caddr_t" et al        */
#include <linux/socket.h>           /* for "struct sockaddr" et al
      */
#include <linux/if.h>               /* for IFNAMSIZ and co... */

/************************** VERSION **************************/
/*
 * This constant is used to know the availability of the wireless
 * extensions and to know which version of wireless extensions it is
 * (there is some stuff that will be added in the future...)
```

```
 * I just plan to increment with each new version.
 */
#define WIRELESS_EXT    16
#define IFNAMSIZ  16
/*
 * Changes :
 *
 * V2 to V3
 * --------
 *    Alan Cox start some incompatibles changes. I've integrated a bit
more.
 *    - Encryption renamed to Encode to avoid US regulation problems
 *    - Frequency changed from float to struct to avoid problems on old
386
 *
 * V3 to V4
 * --------
 *    - Add sensitivity
 *
 * V4 to V5
 * --------
 *    - Missing encoding definitions in range
 *    - Access points stuff
 *
 * V5 to V6
 * --------
 *    - 802.11 support (ESSID ioctls)
 *
 * V6 to V7
 * --------
 *    - define IW_ESSID_MAX_SIZE and IW_MAX_AP
 *
 * V7 to V8
 * --------
 *    - Changed my e-mail address
 *    - More 802.11 support (nickname, rate, rts, frag)
 *    - List index in frequencies
 *
 * V8 to V9
 * --------
 *    - Support for 'mode of operation' (ad-hoc, managed...)
 *    - Support for unicast and multicast power saving
 *    - Change encoding to support larger tokens (>64 bits)
 *    - Updated iw_params (disable, flags) and use it for NWID
 *    - Extracted iw_point from iwreq for clarity
 *
 * V9 to V10
 * ---------
 *    - Add PM capability to range structure
 *    - Add PM modifier : MAX/MIN/RELATIVE
 *    - Add encoding option : IW_ENCODE_NOKEY
 *    - Add TxPower ioctls (work like TxRate)
 *
 * V10 to V11
 * ----------
 *    - Add WE version in range (help backward/forward compatibility)
 *    - Add retry ioctls (work like PM)
```

```
 *
 * V11 to V12
 * ----------
 *    - Add SIOCSIWSTATS to get /proc/net/wireless programatically
 *    - Add DEV PRIVATE IOCTL to avoid collisions in SIOCDEVPRIVATE
space
 *    - Add new statistics (frag, retry, beacon)
 *    - Add average quality (for user space calibration)
 *
 * V12 to V13
 * ----------
 *    - Document creation of new driver API.
 *    - Extract union iwreq_data from struct iwreq (for new driver
API).
 *    - Rename SIOCSIWNAME as SIOCSIWCOMMIT
 *
 * V13 to V14
 * ----------
 *    - Wireless Events support : define struct iw_event
 *    - Define additional specific event numbers
 *    - Add "addr" and "param" fields in union iwreq_data
 *    - AP scanning stuff (SIOCSIWSCAN and friends)
 *
 * V14 to V15
 * ----------
 *    - Add IW_PRIV_TYPE_ADDR for struct sockaddr private arg
 *    - Make struct iw_freq signed (both m & e), add explicit padding
 *    - Add IWEVCUSTOM for driver specific event/scanning token
 *    - Add IW_MAX_GET_SPY for driver returning a lot of addresses
 *    - Add IW_TXPOW_RANGE for range of Tx Powers
 *    - Add IWEVREGISTERED & IWEVEXPIRED events for Access Points
 *    - Add IW_MODE_MONITOR for passive monitor
 *
 * V15 to V16
 * ----------
 *    - Increase the number of bitrates in iw_range to 32 (for 802.11g)
 *    - Increase the number of frequencies in iw_range to 32 (for
802.11b+a)
 *    - Reshuffle struct iw_range for increases, add filler
 *    - Increase IW_MAX_AP to 64 for driver returning a lot of
addresses
 *    - Remove IW_MAX_GET_SPY because conflict with enhanced spy
support
 *    - Add SIOCSIWTHRSPY/SIOCGIWTHRSPY and "struct iw_thrspy"
 *    - Add IW_ENCODE_TEMP and iw_range->encoding_login_index
 */

/************************** CONSTANTS **************************/

/* ------------------------- IOCTL LIST ------------------------- */

/* Wireless Identification */
#define SIOCSIWCOMMIT   0x8B00            /* Commit pending changes to
driver */
#define SIOCGIWNAME     0x8B01            /* get name == wireless
protocol */
/* SIOCGIWNAME is used to verify the presence of Wireless Extensions.
```

```
 * Common values : "IEEE 802.11-DS", "IEEE 802.11-FH", "IEEE
802.11b"...
 * Don't put the name of your driver there, it's useless. */

/* Basic operations */
#define SIOCSIWNWID     0x8B02          /* set network id (pre-
802.11) */
#define SIOCGIWNWID     0x8B03          /* get network id (the cell)
*/
#define SIOCSIWFREQ     0x8B04          /* set channel/frequency (Hz)
*/
#define SIOCGIWFREQ     0x8B05          /* get channel/frequency (Hz)
*/
#define SIOCSIWMODE     0x8B06          /* set operation mode */
#define SIOCGIWMODE     0x8B07          /* get operation mode */
#define SIOCSIWSENS     0x8B08          /* set sensitivity (dBm) */
#define SIOCGIWSENS     0x8B09          /* get sensitivity (dBm) */

/* Informative stuff */
#define SIOCSIWRANGE    0x8B0A          /* Unused */
#define SIOCGIWRANGE    0x8B0B          /* Get range of parameters */
#define SIOCSIWPRIV     0x8B0C          /* Unused */
#define SIOCGIWPRIV     0x8B0D          /* get private ioctl
interface info */
#define SIOCSIWSTATS    0x8B0E          /* Unused */
#define SIOCGIWSTATS    0x8B0F          /* Get /proc/net/wireless
stats */
/* SIOCGIWSTATS is strictly used between user space and the kernel, and
 * is never passed to the driver (i.e. the driver will never see it).
*/

/* Spy support (statistics per MAC address - used for Mobile IP
support) */
#define SIOCSIWSPY      0x8B10          /* set spy addresses */
#define SIOCGIWSPY      0x8B11          /* get spy info (quality of
link) */
#define SIOCSIWTHRSPY   0x8B12          /* set spy threshold (spy
event) */
#define SIOCGIWTHRSPY   0x8B13          /* get spy threshold */

/* Access Point manipulation */
#define SIOCSIWAP 0x8B14            /* set access point MAC addresses
*/
#define SIOCGIWAP 0x8B15            /* get access point MAC addresses
*/
#define SIOCGIWAPLIST   0x8B17          /* Deprecated in favor of
scanning */
#define SIOCSIWSCAN     0x8B18          /* trigger scanning (list
cells) */
#define SIOCGIWSCAN     0x8B19          /* get scanning results */

/* 802.11 specific support */
#define SIOCSIWESSID    0x8B1A          /* set ESSID (network name)
*/
#define SIOCGIWESSID    0x8B1B          /* get ESSID */
#define SIOCSIWNICKN    0x8B1C          /* set node name/nickname */
#define SIOCGIWNICKN    0x8B1D          /* get node name/nickname */
```

```
/* As the ESSID and NICKN are strings up to 32 bytes long, it doesn't
fit
 * within the 'iwreq' structure, so we need to use the 'data' member to
 * point to a string in user space, like it is done for RANGE... */

/* Other parameters useful in 802.11 and some other devices */
#define SIOCSIWRATE     0x8B20            /* set default bit rate (bps)
*/
#define SIOCGIWRATE     0x8B21            /* get default bit rate (bps)
*/
#define SIOCSIWRTS      0x8B22            /* set RTS/CTS threshold
(bytes) */
#define SIOCGIWRTS      0x8B23            /* get RTS/CTS threshold
(bytes) */
#define SIOCSIWFRAG     0x8B24            /* set fragmentation thr
(bytes) */
#define SIOCGIWFRAG     0x8B25            /* get fragmentation thr
(bytes) */
#define SIOCSIWTXPOW    0x8B26            /* set transmit power (dBm)
*/
#define SIOCGIWTXPOW    0x8B27            /* get transmit power (dBm)
*/
#define SIOCSIWRETRY    0x8B28            /* set retry limits and
lifetime */
#define SIOCGIWRETRY    0x8B29            /* get retry limits and
lifetime */

/* Encoding stuff (scrambling, hardware security, WEP...) */
#define SIOCSIWENCODE   0x8B2A            /* set encoding token & mode
*/
#define SIOCGIWENCODE   0x8B2B            /* get encoding token & mode
*/
/* Power saving stuff (power management, unicast and multicast) */
#define SIOCSIWPOWER    0x8B2C            /* set Power Management
settings */
#define SIOCGIWPOWER    0x8B2D            /* get Power Management
settings */

/* -------------------- DEV PRIVATE IOCTL LIST -------------------- */

/* These 16 ioctl are wireless device private.
 * Each driver is free to use them for whatever purpose it chooses,
 * however the driver *must* export the description of those ioctls
 * with SIOCGIWPRIV and *must* use arguments as defined below.
 * If you don't follow those rules, DaveM is going to hate you (reason
:
 * it make mixed 32/64bit operation impossible).
 */
#define SIOCIWFIRSTPRIV 0x8BE0
#define SIOCIWLASTPRIV  0x8BFF
/* Previously, we were using SIOCDEVPRIVATE, but we now have our
 * separate range because of collisions with other tools such as
 * 'mii-tool'.
 * We now have 32 commands, so a bit more space ;-).
 * Also, all 'odd' commands are only usable by root and don't return
the
```

```
 * content of ifr/iwr to user (but you are not obliged to use the
set/get
 * convention, just use every other two command).
 * And I repeat : you are not obliged to use them with iwspy, but you
 * must be compliant with it.
 */

/* ------------------------ IOCTL STUFF ------------------------ */

/* The first and the last (range) */
#define SIOCIWFIRST     0x8B00
#define SIOCIWLAST      SIOCIWLASTPRIV          /* 0x8BFF */

/* Even : get (world access), odd : set (root access) */
#define IW_IS_SET(cmd)  (!((cmd) & 0x1))
#define IW_IS_GET(cmd)  ((cmd) & 0x1)

/* ----------------------- WIRELESS EVENTS ----------------------- */
/* Those are *NOT* ioctls, do not issue request on them !!! */
/* Most events use the same identifier as ioctl requests */

#define IWEVTXDROP      0x8C00            /* Packet dropped to
excessive retry */
#define IWEVQUAL  0x8C01            /* Quality part of statistics
(scan) */
#define IWEVCUSTOM      0x8C02            /* Driver specific ascii
string */
#define IWEVREGISTERED  0x8C03            /* Discovered a new node (AP
mode) */
#define IWEVEXPIRED     0x8C04            /* Expired a node (AP mode)
*/

#define IWEVFIRST 0x8C00

/* ------------------------ PRIVATE INFO ------------------------ */
/*
 * The following is used with SIOCGIWPRIV. It allow a driver to define
 * the interface (name, type of data) for its private ioctl.
 * Privates ioctl are SIOCIWFIRSTPRIV -> SIOCIWLASTPRIV
 */

#define IW_PRIV_TYPE_MASK     0x7000      /* Type of arguments */
#define IW_PRIV_TYPE_NONE     0x0000
#define IW_PRIV_TYPE_BYTE     0x1000      /* Char as number */
#define IW_PRIV_TYPE_CHAR     0x2000      /* Char as character */
#define IW_PRIV_TYPE_INT      0x4000      /* 32 bits int */
#define IW_PRIV_TYPE_FLOAT    0x5000      /* struct iw_freq */
#define IW_PRIV_TYPE_ADDR     0x6000      /* struct sockaddr */

#define IW_PRIV_SIZE_FIXED    0x0800      /* Variable or fixed number
of args */

#define IW_PRIV_SIZE_MASK     0x07FF      /* Max number of those args
*/

/*
 * Note : if the number of args is fixed and the size < 16 octets,
```

```
 * instead of passing a pointer we will put args in the iwreq struct...
 */

/* ---------------------- OTHER CONSTANTS ---------------------- */

/* Maximum frequencies in the range struct */
#define IW_MAX_FREQUENCIES    32
/* Note : if you have something like 80 frequencies,
 * don't increase this constant and don't fill the frequency list.
 * The user will be able to set by channel anyway... */

/* Maximum bit rates in the range struct */
#define IW_MAX_BITRATES       32

/* Maximum tx powers in the range struct */
#define IW_MAX_TXPOWER        8
/* Note : if you more than 8 TXPowers, just set the max and min or
 * a few of them in the struct iw_range. */

/* Maximum of address that you may set with SPY */
#define IW_MAX_SPY            8

/* Maximum of address that you may get in the
   list of access points in range */
#define IW_MAX_AP       64

/* Maximum size of the ESSID and NICKN strings */
#define IW_ESSID_MAX_SIZE     32

/* Modes of operation */
#define IW_MODE_AUTO    0     /* Let the driver decides */
#define IW_MODE_ADHOC   1     /* Single cell network */
#define IW_MODE_INFRA   2     /* Multi cell network, roaming, ... */
#define IW_MODE_MASTER  3     /* Synchronisation master or Access Point
*/
#define IW_MODE_REPEAT  4     /* Wireless Repeater (forwarder) */
#define IW_MODE_SECOND  5     /* Secondary master/repeater (backup) */
#define IW_MODE_MONITOR 6     /* Passive monitor (listen only) */

/* Maximum number of size of encoding token available
 * they are listed in the range structure */
#define IW_MAX_ENCODING_SIZES 8

/* Maximum size of the encoding token in bytes */
#define IW_ENCODING_TOKEN_MAX 32    /* 256 bits (for now) */

/* Flags for encoding (along with the token) */
#define IW_ENCODE_INDEX       0x00FF      /* Token index (if needed) */
#define IW_ENCODE_FLAGS       0xFF00      /* Flags defined below */
#define IW_ENCODE_MODE        0xF000      /* Modes defined below */
#define IW_ENCODE_DISABLED    0x8000      /* Encoding disabled */
#define IW_ENCODE_ENABLED     0x0000      /* Encoding enabled */
#define IW_ENCODE_RESTRICTED  0x4000      /* Refuse non-encoded packets
*/
#define IW_ENCODE_OPEN        0x2000      /* Accept non-encoded packets
*/
```

```
#define IW_ENCODE_NOKEY       0x0800  /* Key is write only, so not
present */
#define IW_ENCODE_TEMP        0x0400  /* Temporary key */

/* Power management flags available (along with the value, if any) */
#define IW_POWER_ON           0x0000      /* No details... */
#define IW_POWER_TYPE         0xF000      /* Type of parameter */
#define IW_POWER_PERIOD       0x1000      /* Value is a period/duration
of  */
#define IW_POWER_TIMEOUT      0x2000      /* Value is a timeout (to go
asleep) */
#define IW_POWER_MODE         0x0F00      /* Power Management mode */
#define IW_POWER_UNICAST_R    0x0100      /* Receive only unicast
messages */
#define IW_POWER_MULTICAST_R  0x0200      /* Receive only multicast
messages */
#define IW_POWER_ALL_R        0x0300      /* Receive all messages
though PM */
#define IW_POWER_FORCE_S      0x0400      /* Force PM procedure for
sending unicast */
#define IW_POWER_REPEATER     0x0800      /* Repeat broadcast messages
in PM period */
#define IW_POWER_MODIFIER     0x000F      /* Modify a parameter */
#define IW_POWER_MIN          0x0001      /* Value is a minimum  */
#define IW_POWER_MAX          0x0002      /* Value is a maximum */
#define IW_POWER_RELATIVE     0x0004      /* Value is not in
seconds/ms/us */

/* Transmit Power flags available */
#define IW_TXPOW_TYPE         0x00FF      /* Type of value */
#define IW_TXPOW_DBM          0x0000      /* Value is in dBm */
#define IW_TXPOW_MWATT        0x0001      /* Value is in mW */
#define IW_TXPOW_RANGE        0x1000      /* Range of value between
min/max */

/* Retry limits and lifetime flags available */
#define IW_RETRY_ON           0x0000      /* No details... */
#define IW_RETRY_TYPE         0xF000      /* Type of parameter */
#define IW_RETRY_LIMIT        0x1000      /* Maximum number of
retries*/
#define IW_RETRY_LIFETIME     0x2000      /* Maximum duration of
retries in us */
#define IW_RETRY_MODIFIER     0x000F      /* Modify a parameter */
#define IW_RETRY_MIN          0x0001      /* Value is a minimum  */
#define IW_RETRY_MAX          0x0002      /* Value is a maximum */
#define IW_RETRY_RELATIVE     0x0004      /* Value is not in
seconds/ms/us */

/* Scanning request flags */
#define IW_SCAN_DEFAULT       0x0000      /* Default scan of the driver
*/
#define IW_SCAN_ALL_ESSID     0x0001      /* Scan all ESSIDs */
#define IW_SCAN_THIS_ESSID    0x0002      /* Scan only this ESSID */
#define IW_SCAN_ALL_FREQ      0x0004      /* Scan all Frequencies */
#define IW_SCAN_THIS_FREQ     0x0008      /* Scan only this Frequency
*/
#define IW_SCAN_ALL_MODE      0x0010      /* Scan all Modes */
```

```
#define IW_SCAN_THIS_MODE     0x0020      /* Scan only this Mode */
#define IW_SCAN_ALL_RATE      0x0040      /* Scan all Bit-Rates */
#define IW_SCAN_THIS_RATE     0x0080      /* Scan only this Bit-Rate */
/* Maximum size of returned data */
#define IW_SCAN_MAX_DATA      4096  /* In bytes */

/* Max number of char in custom event - use multiple of them if needed
*/
#define IW_CUSTOM_MAX         256   /* In bytes */

/****************************** TYPES ******************************/

/* --------------------------- SUBTYPES --------------------------- */
/*
 *    Generic format for most parameters that fit in an int
 */
struct      iw_param
{
  __s32           value;            /* The value of the parameter
itself */
  __u8            fixed;            /* Hardware should not use auto
select */
  __u8            disabled;   /* Disable the feature */
  __u16           flags;            /* Various specifc flags (if any)
*/
};

/*
 *    For all data larger than 16 octets, we need to use a
 *    pointer to memory allocated in user space.
 */
struct      iw_point
{
  caddr_t   pointer;    /* Pointer to the data  (in user space) */
  __u16           length;           /* number of fields or size in
bytes */
  __u16           flags;            /* Optional params */
};

/*
 *    A frequency
 *    For numbers lower than 10^9, we encode the number in 'm' and
 *    set 'e' to 0
 *    For number greater than 10^9, we divide it by the lowest power
 *    of 10 to get 'm' lower than 10^9, with 'm'= f / (10^'e')...
 *    The power of 10 is in 'e', the result of the division is in 'm'.
 */
struct      iw_freq
{
      __s32       m;          /* Mantissa */
      __s16       e;          /* Exponent */
      __u8        i;          /* List index (when in range struct) */
      __u8        pad;        /* Unused - just for alignement */
};

/*
 *    Quality of the link
```

```
 */
struct      iw_quality
{
      __u8        qual;       /* link quality (%retries, SNR,
                                 %missed beacons or better...) */
      __u8        level;          /* signal level (dBm) */
      __u8        noise;          /* noise level (dBm) */
      __u8        updated;    /* Flags to know if updated */
};

/*
 *    Packet discarded in the wireless adapter due to
 *    "wireless" specific problems...
 *    Note : the list of counter and statistics in net_device_stats
 *    is already pretty exhaustive, and you should use that first.
 *    This is only additional stats...
 */
struct      iw_discarded
{
      __u32       nwid;       /* Rx : Wrong nwid/essid */
      __u32       code;       /* Rx : Unable to code/decode (WEP) */
      __u32       fragment;   /* Rx : Can't perform MAC reassembly */
      __u32       retries;    /* Tx : Max MAC retries num reached */
      __u32       misc;       /* Others cases */
};

/*
 *    Packet/Time period missed in the wireless adapter due to
 *    "wireless" specific problems...
 */
struct      iw_missed
{
      __u32       beacon;         /* Missed beacons/superframe */
};

/*
 *    Quality range (for spy threshold)
 */
struct      iw_thrspy
{
      struct sockaddr         addr;       /* Source address (hw/mac) */
      struct iw_quality qual;       /* Quality of the link */
      struct iw_quality low;        /* Low threshold */
      struct iw_quality high;       /* High threshold */
};

/* ------------------------ WIRELESS STATS ------------------------ */
/*
 * Wireless statistics (used for /proc/net/wireless)
 */
struct      iw_statistics
{
      __u16       status;         /* Status
                                   * - device dependent for now */

      struct iw_quality qual;       /* Quality of the link
                                     * (instant/mean/max) */
```

```
      struct iw_discarded     discard;    /* Packet discarded counts */
      struct iw_missed  miss;       /* Packet missed counts */
};

/* ------------------------ IOCTL REQUEST ------------------------ */
/*
 * This structure defines the payload of an ioctl, and is used
 * below.
 *
 * Note that this structure should fit on the memory footprint
 * of iwreq (which is the same as ifreq), which mean a max size of
 * 16 octets = 128 bits. Warning, pointers might be 64 bits wide...
 * You should check this when increasing the structures defined
 * above in this file...
 */
union iwreq_data
{
      /* Config - generic */
      char        name[IFNAMSIZ];
      /* Name : used to verify the presence of  wireless extensions.
       * Name of the protocol/provider... */

      struct iw_point   essid;            /* Extended network name */
      struct iw_param   nwid;       /* network id (or domain - the
cell) */
      struct iw_freq    freq;       /* frequency or channel :
                              * 0-1000 = channel
                              * > 1000 = frequency in Hz */

      struct iw_param   sens;       /* signal level threshold */
      struct iw_param   bitrate;    /* default bit rate */
      struct iw_param   txpower;    /* default transmit power */
      struct iw_param   rts;        /* RTS threshold threshold */
      struct iw_param   frag;       /* Fragmentation threshold */
      __u32       mode;       /* Operation mode */
      struct iw_param   retry;            /* Retry limits & lifetime */

      struct iw_point   encoding;   /* Encoding stuff : tokens */
      struct iw_param   power;            /* PM duration/timeout */
      struct iw_quality qual;       /* Quality part of statistics */

      struct sockaddr   ap_addr;    /* Access point address */
      struct sockaddr   addr;       /* Destination address (hw/mac) */

      struct iw_param   param;            /* Other small parameters */
      struct iw_point   data;       /* Other large parameters */
};

/*
 * The structure to exchange data for ioctl.
 * This structure is the same as 'struct ifreq', but (re)defined for
 * convenience...
 * Do I need to remind you about structure size (32 octets) ?
 */
struct      iwreq
{
      union
```

```
      {
            char  ifrn_name[IFNAMSIZ];    /* if name, e.g. "eth0" */
      } ifr_ifrn;

      /* Data part (defined just above) */
      union iwreq_data  u;
};

/* ------------------------- IOCTL DATA ------------------------- */
/*
 *    For those ioctl which want to exchange mode data that what could
 *    fit in the above structure...
 */

/*
 *    Range of parameters
 */

struct      iw_range
{
      /* Informative stuff (to choose between different interface) */
      __u32       throughput; /* To give an idea... */
      /* In theory this value should be the maximum benchmarked
       * TCP/IP throughput, because with most of these devices the
       * bit rate is meaningless (overhead an co) to estimate how
       * fast the connection will go and pick the fastest one.
       * I suggest people to play with Netperf or any benchmark...
       */

      /* NWID (or domain id) */
      __u32       min_nwid;   /* Minimal NWID we are able to set */
      __u32       max_nwid;   /* Maximal NWID we are able to set */

      /* Old Frequency (backward compat - moved lower ) */
      __u16       old_num_channels;
      __u8        old_num_frequency;
      /* Filler to keep "version" at the same offset */
      __s32       old_freq[6];

      /* signal level threshold range */
      __s32 sensitivity;

      /* Quality of link & SNR stuff */
      /* Quality range (link, level, noise)
       * If the quality is absolute, it will be in the range [0 ;
max_qual],
       * if the quality is dBm, it will be in the range [max_qual ; 0].
       * Don't forget that we use 8 bit arithmetics... */
      struct iw_quality max_qual;   /* Quality of the link */
      /* This should contain the average/typical values of the quality
       * indicator. This should be the threshold between a "good" and
       * a "bad" link (example : monitor going from green to orange).
       * Currently, user space apps like quality monitors don't have
any
       * way to calibrate the measurement. With this, they can split
       * the range between 0 and max_qual in different quality level
       * (using a geometric subdivision centered on the average).
```

```
       * I expect that people doing the user space apps will feedback
       * us on which value we need to put in each driver... */
      struct iw_quality avg_qual;   /* Quality of the link */

      /* Rates */
      __u8        num_bitrates;     /* Number of entries in the list */
      __s32       bitrate[IW_MAX_BITRATES];     /* list, in bps */

      /* RTS threshold */
      __s32       min_rts;    /* Minimal RTS threshold */
      __s32       max_rts;    /* Maximal RTS threshold */

      /* Frag threshold */
      __s32       min_frag;   /* Minimal frag threshold */
      __s32       max_frag;   /* Maximal frag threshold */

      /* Power Management duration & timeout */
      __s32       min_pmp;    /* Minimal PM period */
      __s32       max_pmp;    /* Maximal PM period */
      __s32       min_pmt;    /* Minimal PM timeout */
      __s32       max_pmt;    /* Maximal PM timeout */
      __u16       pmp_flags;  /* How to decode max/min PM period */
      __u16       pmt_flags;  /* How to decode max/min PM timeout */
      __u16       pm_capa;    /* What PM options are supported */

      /* Encoder stuff */
      __u16 encoding_size[IW_MAX_ENCODING_SIZES];     /* Different
token sizes */
      __u8  num_encoding_sizes;     /* Number of entry in the list */
      __u8  max_encoding_tokens;    /* Max number of tokens */
      /* For drivers that need a "login/passwd" form */
      __u8  encoding_login_index;   /* token index for login token */

      /* Transmit power */
      __u16       txpower_capa;     /* What options are supported */
      __u8        num_txpower;      /* Number of entries in the list */
      __s32       txpower[IW_MAX_TXPOWER];      /* list, in bps */

      /* Wireless Extension version info */
      __u8        we_version_compiled;    /* Must be WIRELESS_EXT */
      __u8        we_version_source;      /* Last update of source */

      /* Retry limits and lifetime */
      __u16       retry_capa; /* What retry options are supported */
      __u16       retry_flags;      /* How to decode max/min retry
limit */
      __u16       r_time_flags;     /* How to decode max/min retry life
*/
      __s32       min_retry;  /* Minimal number of retries */
      __s32       max_retry;  /* Maximal number of retries */
      __s32       min_r_time; /* Minimal retry lifetime */
      __s32       max_r_time; /* Maximal retry lifetime */

      /* Frequency */
      __u16       num_channels;     /* Number of channels [0; num - 1]
*/
      __u8        num_frequency;    /* Number of entry in the list */
```

```
      struct iw_freq    freq[IW_MAX_FREQUENCIES];     /* list */
      /* Note : this frequency list doesn't need to fit channel
numbers,
       * because each entry contain its channel index */
};

/*
 * Private ioctl interface information
 */

struct      iw_priv_args
{
      __u32       cmd;        /* Number of the ioctl to issue */
      __u16       set_args;   /* Type and number of args */
      __u16       get_args;   /* Type and number of args */
      char        name[IFNAMSIZ];   /* Name of the extension */
};

/* ------------------------ WIRELESS EVENTS ----------------------- */
/*
 * Wireless events are carried through the rtnetlink socket to user
 * space. They are encapsulated in the IFLA_WIRELESS field of
 * a RTM_NEWLINK message.
 */

/*
 * A Wireless Event. Contains basically the same data as the ioctl...
 */
struct iw_event
{
      __u16       len;              /* Real lenght of this stuff */
      __u16       cmd;              /* Wireless IOCTL */
      union iwreq_data  u;          /* IOCTL fixed payload */
};

/* Size of the Event prefix (including padding and alignement junk) */
#define IW_EV_LCP_LEN   (sizeof(struct iw_event) - sizeof(union
iwreq_data))
/* Size of the various events */
#define IW_EV_CHAR_LEN  (IW_EV_LCP_LEN + IFNAMSIZ)
#define IW_EV_UINT_LEN  (IW_EV_LCP_LEN + sizeof(__u32))
#define IW_EV_FREQ_LEN  (IW_EV_LCP_LEN + sizeof(struct iw_freq))
#define IW_EV_POINT_LEN (IW_EV_LCP_LEN + sizeof(struct iw_point))
#define IW_EV_PARAM_LEN (IW_EV_LCP_LEN + sizeof(struct iw_param))
#define IW_EV_ADDR_LEN  (IW_EV_LCP_LEN + sizeof(struct sockaddr))
#define IW_EV_QUAL_LEN  (IW_EV_LCP_LEN + sizeof(struct iw_quality))

/* Note : in the case of iw_point, the extra data will come at the
 * end of the event */

#endif      /* _LINUX_WIRELESS_H */
```