How to get info for the wireless interface using nl80211 in C

09 Oct 2016 · **Coding** · **linux** **wifi** **c**

Contents

nl80211

The nl80211 is the 802.11 netlink-based userspace interface for the new cfg80211 configuration system for wireless hardware. Together they are intended to replace the old

Wireless-Extensions. Current users of nl80211 include:

- iw
- crda
- hostapd
- wpa_supplicant (-Dnl80211)

Includes

First of all, let's add the necessary includes:

```
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netlink/netlink.h>    //lots of netlink functions
#include <netlink/genl/genl.h>  //genl_connect, genlmsg_put
#include <netlink/genl/family.h>
#include <netlink/genl/ctrl.h>  //genl_ctrl_resolve
#include <linux/nl80211.h>      //NL80211 definitions
```

Structs

Let's define some data structures we will need later on:

```
typedef struct {
  int id;
  struct nl_sock* socket;
  struct nl_cb* cb1,* cb2;
  int result1, result2;
} Netlink;

typedef struct {
  char ifname[30];
  int ifindex;
  int signal;
  int txrate;
} Wifi;

static struct nla_policy stats_policy[NL80211_STA_INFO_MAX + 1] = {
  [NL80211_STA_INFO_INACTIVE_TIME] = { .type = NLA_U32 },
  [NL80211_STA_INFO_RX_BYTES] = { .type = NLA_U32 },
  [NL80211_STA_INFO_TX_BYTES] = { .type = NLA_U32 },
  [NL80211_STA_INFO_RX_PACKETS] = { .type = NLA_U32 },
  [NL80211_STA_INFO_TX_PACKETS] = { .type = NLA_U32 },
  [NL80211_STA_INFO_SIGNAL] = { .type = NLA_U8 },
  [NL80211_STA_INFO_TX_BITRATE] = { .type = NLA_NESTED },
  [NL80211_STA_INFO_LLID] = { .type = NLA_U16 },
  [NL80211_STA_INFO_PLID] = { .type = NLA_U16 },
  [NL80211_STA_INFO_PLINK_STATE] = { .type = NLA_U8 },
};

static struct nla_policy rate_policy[NL80211_RATE_INFO_MAX + 1] = {
  [NL80211_RATE_INFO_BITRATE] = { .type = NLA_U16 },
  [NL80211_RATE_INFO_MCS] = { .type = NLA_U8 },
  [NL80211_RATE_INFO_40_MHZ_WIDTH] = { .type = NLA_FLAG },
  [NL80211_RATE_INFO_SHORT_GI] = { .type = NLA_FLAG },
};
```

Functions

We define our functions:

```
static int initNl80211(Netlink* nl, Wifi* w);
static int finish_handler(struct nl_msg *msg, void *arg);
static int getWifiName_callback(struct nl_msg *msg, void *arg);
static int getWifiInfo_callback(struct nl_msg *msg, void *arg);
static int getWifiStatus(Netlink* nl, Wifi* w);
```

initNl80211()

In our function initNl80211, we initialize our communication with the Kernel. We follow these steps:

1. We allocate a netlink socket (using nl_socket_alloc)

2. Optionally, we can set the socket buffer size (using nl_socket_set_buffer_size)

3. We connect to the generic netlink socket (using genl_connect)

4. We ask the Kernel to resolve family name "nl80211" to family id (using genl_ctrl_resolve)

5. We allocate two new callback handles (using nl_cb_alloc)

6. We set some callbacks (using nl_cb_set).

```
static int initNl80211(Netlink* nl, Wifi* w) {
  nl->socket = nl_socket_alloc();
  if (!nl->socket) {
    fprintf(stderr, "Failed to allocate netlink socket.\n");
    return -ENOMEM;
  }

  nl_socket_set_buffer_size(nl->socket, 8192, 8192);

  if (genl_connect(nl->socket)) {
    fprintf(stderr, "Failed to connect to netlink socket.\n");
    nl_close(nl->socket);
    nl_socket_free(nl->socket);
    return -ENOLINK;
  }

  nl->id = genl_ctrl_resolve(nl->socket, "nl80211");
  if (nl->id< 0) {
    fprintf(stderr, "Nl80211 interface not found.\n");
    nl_close(nl->socket);
    nl_socket_free(nl->socket);
    return -ENOENT;
  }

  nl->cb1 = nl_cb_alloc(NL_CB_DEFAULT);
  nl->cb2 = nl_cb_alloc(NL_CB_DEFAULT);
  if ((!nl->cb1) || (!nl->cb2)) {
     fprintf(stderr, "Failed to allocate netlink callback.\n");
     nl_close(nl->socket);
     nl_socket_free(nl->socket);
     return ENOMEM;
  }

  nl_cb_set(nl->cb1, NL_CB_VALID , NL_CB_CUSTOM, getWifiName_callback, w);
  nl_cb_set(nl->cb1, NL_CB_FINISH, NL_CB_CUSTOM, finish_handler, &(nl->result1));
  nl_cb_set(nl->cb2, NL_CB_VALID , NL_CB_CUSTOM, getWifiInfo_callback, w);
  nl_cb_set(nl->cb2, NL_CB_FINISH, NL_CB_CUSTOM, finish_handler, &(nl->result2));

  return nl->id;
}
```

finish_handler()

This is our finish_handler.

```
static int finish_handler(struct nl_msg *msg, void *arg) {
  int *ret = arg;
  *ret = 0;
  return NL_SKIP;
}
```

The finish_handler will allow us -later on- to receive our messages from Kernel like that:

```
while (nl->result1 > 0) { nl_recvmsgs(nlsocket, nl->cb1); }
while (nl->result2 > 0) { nl_recvmsgs(nlsocket, nl->cb2); }
```

getWifiName_callback()

This is our getWifiName_callback. Here we parse the message from the Kernel and we get the interface name (wifi_iface) and its index (wifi_index). If you would like, you can use the nl_msg_dump(msg, stdout) to see the raw message.

```
static int getWifiName_callback(struct nl_msg *msg, void *arg) {

  struct genlmsghdr *gnlh = nlmsg_data(nlmsg_hdr(msg));

  struct nlattr *tb_msg[NL80211_ATTR_MAX + 1];

  //nl_msg_dump(msg, stdout);

  nla_parse(tb_msg,
            NL80211_ATTR_MAX,
            genlmsg_attrdata(gnlh, 0),
            genlmsg_attrlen(gnlh, 0),
            NULL);

  if (tb_msg[NL80211_ATTR_IFNAME]) {
    strcpy(((Wifi*)arg)->ifname, nla_get_string(tb_msg[NL80211_ATTR_IFNAME]));
  }

  if (tb_msg[NL80211_ATTR_IFINDEX]) {
    ((Wifi*)arg)->ifindex = nla_get_u32(tb_msg[NL80211_ATTR_IFINDEX]);
  }

  return NL_SKIP;
}
```

getWifiInfo_callback()

This is our getWifiInfo_callback. Here we parse the message from the Kernel and we get the wifi signal (wifi_signal) and txrate (wifi_bitrate). If you would like, you can use the nl_msg_dump(msg, stdout) to see the raw message.

```
static int getWifiInfo_callback(struct nl_msg *msg, void *arg) {
  struct nlattr *tb[NL80211_ATTR_MAX + 1];
  struct genlmsghdr *gnlh = nlmsg_data(nlmsg_hdr(msg));
  struct nlattr *sinfo[NL80211_STA_INFO_MAX + 1];
  struct nlattr *rinfo[NL80211_RATE_INFO_MAX + 1];

  //nl_msg_dump(msg, stdout);

  nla_parse(tb,
            NL80211_ATTR_MAX,
            genlmsg_attrdata(gnlh, 0),
            genlmsg_attrlen(gnlh, 0),
            NULL);

  if (!tb[NL80211_ATTR_STA_INFO]) {
    fprintf(stderr, "sta stats missing!\n"); return NL_SKIP;
  }

  if (nla_parse_nested(sinfo, NL80211_STA_INFO_MAX,
                       tb[NL80211_ATTR_STA_INFO], stats_policy)) {
    fprintf(stderr, "failed to parse nested attributes!\n"); return NL_SKIP;
  }

  if (sinfo[NL80211_STA_INFO_SIGNAL]) {
    ((Wifi*)arg)->signal = 100+(int8_t)nla_get_u8(sinfo[NL80211_STA_INFO_SIGNAL]);
  }

  if (sinfo[NL80211_STA_INFO_TX_BITRATE]) {
    if (nla_parse_nested(rinfo, NL80211_RATE_INFO_MAX,
                         sinfo[NL80211_STA_INFO_TX_BITRATE], rate_policy)) {
      fprintf(stderr, "failed to parse nested rate attributes!\n"); }
    else {
      if (rinfo[NL80211_RATE_INFO_BITRATE]) {
        ((Wifi*)arg)->txrate = nla_get_u16(rinfo[NL80211_RATE_INFO_BITRATE]);
      }
    }
  }
  return NL_SKIP;
}
```

getWifiStatus()

Let's see now what happens in our next function, getWifiStatus:

1. We allocate a netlink message structure (using nlmsg_alloc)
2. We add generic netlink headers to the netlink message (using genlmsg_put)
3. We finalize and transmit the netlink message (using nl_send_auto)
4. We receive a set of messages from the netlink socket (using nl_recvmsgs)
5. We release the netlink message reference (using nlmsg_free)

We execute those steps twice:

a) The first time we use the NL80211_CMD_GET_INTERFACE command identifier to get the wireless interface name and index.

b) The second time we use the NL80211_CMD_GET_STATION command identifier to get signal strength and transmit bitrate. The reason for it is that the signal and bitrate values have meaning only relative to a station. Note that we have to put in the message the interface index using the nla_put_u32(msg2, NL80211_ATTR_IFINDEX, w->ifindex).

```
static int getWifiStatus(Netlink* nl, Wifi* w) {
  nl->result1 = 1;
  nl->result2 = 1;

  struct nl_msg* msg1 = nlmsg_alloc();
  if (!msg1) {
    fprintf(stderr, "Failed to allocate netlink message.\n");
    return -2;
  }

  genlmsg_put(msg1,
              NL_AUTO_PORT,
              NL_AUTO_SEQ,
              nl->id,
              0,
              NLM_F_DUMP,
              NL80211_CMD_GET_INTERFACE,
              0);

  nl_send_auto(nl->socket, msg1);

  while (nl->result1 > 0) { nl_recvmsgs(nl->socket, nl->cb1); }
  nlmsg_free(msg1);

  if (w->ifindex < 0) { return -1; }

  struct nl_msg* msg2 = nlmsg_alloc();

  if (!msg2) {
    fprintf(stderr, "Failed to allocate netlink message.\n");
    return -2;
  }

  genlmsg_put(msg2,
              NL_AUTO_PORT,
              NL_AUTO_SEQ,
              nl->id,
              0,
              NLM_F_DUMP,
              NL80211_CMD_GET_STATION,
              0);

  nla_put_u32(msg2, NL80211_ATTR_IFINDEX, w->ifindex);
  nl_send_auto(nl->socket, msg2);
  while (nl->result2 > 0) { nl_recvmsgs(nl->socket, nl->cb2); }
  nlmsg_free(msg2);

  return 0;
}
```

main()

Here is our main() function. First, we initialize the communication using the initNl80211 and after that, we continuously call the getWifiStatus() inside a loop, every 1 sec, until the user presses ctrl + c:

```
int main(int argc, char **argv) {
  Netlink nl;
  Wifi w;

  signal(SIGINT, ctrl_c_handler);

  nl.id = initNl80211(&nl, &w);
  if (nl.id < 0) {
    fprintf(stderr, "Error initializing netlink 802.11\n");
    return -1;
  }
  do {
    getWifiStatus(&nl, &w);
    printf("Interface: %s | signal: %d dB | txrate: %.1f MBit/s\n",
           w.ifname, w.signal, (float)w.txrate/10);
    sleep(1);
  } while(keepRunning);

  printf("\nExiting gracefully... ");
  nl_cb_put(nl.cb1);
  nl_cb_put(nl.cb2);
  nl_close(nl.socket);
  nl_socket_free(nl.socket);
  printf("OK\n");
  return 0;
}
```

Source code

Here is the full code listing:

- nl80211_info.c
- makefile