Guide for measurements of Mobile Communications and Quantum Technologies Laboratory

GeoNetworking protocol

Place of measurement:
Department of Networked Systems and Services, Mobile Communications and Quantum Technologies Laboratory (MCL) I.B.113.

Made by:
Ádám Knapp

Last modification:
24. October 2017
Intelligent Transport System:

![ITS protocol stack](image1)

Figure 1. ITS protocol stack

![Operation of different message types](image2)

Figure 2. Operation of different message types

Measurement setup
Computer with Windows 7 OS:

- User: MCL
- Password: mcl
- IP address: 192.168.37.xxx

VirtualBox application and Linux image with cross-compiler for ITRI:
- User: user
- Password: password
- IP address: 192.168.37.xxx

ITRI devices:
- User: admin
- Password: iwcu2009
- IP address: 192.168.37.3x

**Task 1.**
Establish connection between devices and OS-es!

**Help:**
Check connection:

$ping 192.168.37.xxx

Set static IP address on Linux:

$ifconfig eth0 down

$ifconfig eth0 192.168.37.xx netmask 255.255.255.0 up

Set date on Linux:

$sudo date –s “2017-10-24 13:55:00”

**Task 2.**
Log in to the ITRI device using PuTTY client!

**Help:**
Steps for setup an SSH session:

1. Set connection type to SSH! (Default port for SSH is 22.)
2. Fill the Host Name field with the destination IP address!
3. Click to open!
4. Use the username and password to access the device!
Task 3.
Configure the ITRI device!

Set bandwidth and carrier frequency:
Check current value:

$cat /proc/sys/wave/channel
$wave0 10@5920

Set new value:

$echo wave0 10@5890 > /proc/sys/wave/channel

Possible values:
Set transmission power

Check current value:

$cat /proc/sys/wave/txpower

Set new value:

$echo "wave0 18" >/proc/sys/wave/txpower

Set modulation and coding scheme

Check current value:

$cat /proc/sys/wave/txrate

Set new value:

$echo "wave0 4" >/proc/sys/wave/txrate

Possible modulation and coding schemes:

<table>
<thead>
<tr>
<th>Value</th>
<th>Modulation</th>
<th>Data Rate at 10MHz Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BPSK 1/2</td>
<td>3Mbps</td>
</tr>
<tr>
<td>1</td>
<td>BPSK 3/4</td>
<td>4.5Mbps</td>
</tr>
<tr>
<td>2</td>
<td>QPSK 1/2</td>
<td>6Mbps</td>
</tr>
<tr>
<td>3</td>
<td>QPSK 3/4</td>
<td>9Mbps</td>
</tr>
<tr>
<td>4</td>
<td>16QAM 1/2</td>
<td>12Mbps</td>
</tr>
<tr>
<td>5</td>
<td>16QAM 3/4</td>
<td>18Mbps</td>
</tr>
<tr>
<td>6</td>
<td>64QAM 2/3</td>
<td>24Mbps</td>
</tr>
<tr>
<td>7</td>
<td>64QAM 3/4</td>
<td>27.5Mbps</td>
</tr>
</tbody>
</table>

Get statistics

$cat /proc/sys/wave/stats

Device information (MAC)
$cat /proc/net/gn/lpv

Neighbor device information (MAC)

$cat /proc/net/gn/loc

Set location information

Check current value:

$cat /proc/sys/net/gn/local_longitude

Set new value:

$echo 511234567 > /proc/sys/net/gn/local_latitude
$echo 51234567 > /proc/sys/net/gn/local_longitude

Note: The above format follows the WGS-84 format meaning that the latitude is set to 51.1234567 and the longitude is set to 5.1234567.

Task 4.
Develop a GeoNetworking sender and/or receiver application!

Using cross-compiler of the Linux VM

1. Copy the source codes in /user/home/example!
2. Go to the following directory:
   
   $cd /home/user/powerpc-e300c3-linux-gnu/bin

3. Compile the code with the following command:
   
   $./powerpc-e300c3-linux-gnu-gcc /home/user/example/proba.c /home/user/example/gn.h -o /home/user/example/proba

After successful compilation, the executable binary will be available under /home/user/example.

Copy the binary file to the ITRI using scp:

scp <files to copy> username@<ip address >:<destination directory>

Example:

scp proba user@192.168.37.30:/home/user/

Run the application on the ITRI device!

Help:

Socket initialization:

int socket(int domain, int type, int protocol)

Example:

sd = socket(PF_BTP, SOCK_RAW, 0);
**Binding:**

```c
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

**Example:**

```c
err = bind(sd, (struct sockaddr*) &sbs, sizeof(sbs));
```

**GeoNetworking header fields in structure:**

```c
struct sbs {
    unsigned short btp_family;
    unsigned char btp_type;
    unsigned short sport;
    unsigned short dport;
    unsigned short dport_info;
    unsigned char packet_type;
    union gn_dest dest;
    struct long_pv src_pv;
    unsigned char commun_profile;
    unsigned char sec_profile;
    struct life_time packet_lifetime;
    unsigned short max_repeat_time;
    unsigned short repeat_interval;
    unsigned char hop_limit;
    struct traffic tc;
};
```

```c
union gn_dest {
    struct gn_addr addr;
    struct gn_area area;
};
```

```c
struct gn_addr {
    unsigned short manual:1,
                   type:5,
                   country_code:10;
    unsigned char mid[MAC_SIZE];
};
```

```c
struct gn_area{
    unsigned char area_type;
    unsigned int pos_latitude;
    unsigned int pos_longitude;
    unsigned short distance_a;
    unsigned short distance_b;
    unsigned short angle;
};
```

```c
struct long_pv {
    struct gn_addr addr;
    __u32 timestamp;
    __s32 latitude;
    __s32 longitude;
    __u8 pai : 1;
```
__s16  speed : 15;
__u16  heading;
}

Example for accessing an element:
printf("heading= %d\n", sbs.src_pv.heading);

Data transmission:
int sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr *dest_addr, socklen_t addrlen);

Example:
sendto(sd, msg, 2048, 0, (struct sockaddr *) &sbs, sizeof(sbs));

Data reception:
int recvfrom(int sockfd, void *buf, size_t len, int flags,
struct sockaddr *src_addr, socklen_t *addrlen);

Example:
int recvfrom(sd, msg, 2048, 0, (struct sockaddr *) &sbs, &len);

Closing socket:
close(sd);