ule6lo getting started

Contents

1 Introduction 2

[1.1 Scope 2](#_Toc439680495)

[1.2 History 2](#_Toc439680496)

[1.3 References 2](#_Toc439680497)

[1.4 Terms & Abbreviations 2](#_Toc439680498)

[2 Requirements 3](#_Toc439680499)

[2.1 Operating system 3](#_Toc439680500)

[2.1.1 Virtual box installation 3](#_Toc439680501)

[2.1.2 Installation of Ubuntu inside virtual box. 3](#_Toc439680502)

[2.1.3 Required packages 4](#_Toc439680503)

[2.2 IPv6 capable router 4](#_Toc439680504)

[2.2.1 Router configuration 4](#_Toc439680505)

[2.2.2 Verifying router setup 4](#_Toc439680506)

[2.2.3 Bridge the network to allow access for the library. 5](#_Toc439680507)

[2.3 DNS server 5](#_Toc439680508)

[2.3.1 DNS server listing in router advertisements 5](#_Toc439680509)

[2.3.2 DNS server 7](#_Toc439680510)

[3 Software structure 8](#_Toc439680511)

[3.1 Apps 8](#_Toc439680512)

[3.2 Src 9](#_Toc439680513)

[3.2.1 6LBR 9](#_Toc439680514)

[3.2.2 6LN 9](#_Toc439680515)

[3.2.3 Contiki 9](#_Toc439680516)

[3.3 Tools 9](#_Toc439680517)

[3.3.1 DebugFunctions 9](#_Toc439680518)

[3.3.2 FileLogger 9](#_Toc439680519)

[3.3.3 LinkLayerAbs 9](#_Toc439680520)

[4 Compiling the code 10](#_Toc439680521)

[4.1 Enter folder 10](#_Toc439680522)

[4.2 Creating/updating dependencies 10](#_Toc439680523)

[4.3 Compile the code 10](#_Toc439680524)

[5 Running the applications 10](#_Toc439680525)

[5.1 6LBR 10](#_Toc439680526)

[5.2 6LN 11](#_Toc439680527)

[5.3 External tools 11](#_Toc439680528)

[5.3.1 Netcat 11](#_Toc439680529)

[5.3.2 udp\_server.c 12](#_Toc439680530)

# Introduction

## Scope

This document is a short guide for getting started using the current development release of the ule6lo library and test applications. The document will not describe in details how the software is structured, but only give a short introduction to the general structure of the code. In addition the requirements for installing and running the software is provided. The state of the code is considered a development release and is only sunshine tested.

## History

| **Revision** | **Author** | **Issue Date** | **Comments** |
| --- | --- | --- | --- |
| 0.1 | JJO | 5-NOV-15 | Initial Revision |
| 0.2 | JJO | 04-DEC-15 | Added section 2.2 to include additional requirements for the border router feature.  Added section 5.3 to include some external tools useful for testing the border router feature. |
| 0.3 | JJO | 01-JAN-16 | Added information about DNS resolvement, chapter 2.3.  Added short section on how to resolv in the apps. (5.1 / 5.2)  Updated 5.2 6LN section with a new way to launch nodes. |

## References

|  |  |  |
| --- | --- | --- |
| [1] | **Title:** | Interface Specification - Open source 6LoWPAN implementation for ULE FP and PP |
|  | Author: | Thomas Kjeldsen |
|  | Location: | ??? |
|  |  |  |

## Terms & Abbreviations

|  |  |
| --- | --- |
| **6LoWPAN**  **6LBR**  **6LN**  **API** | IPv6 over Low power Wireless Personal Area Networks  6LoWPAN Board Router (In DECT known as FP)  6LoWPAN Node (In DECT known as PP)  Application Interface |
| **DECT** | Digital Enhanced Cordless Telephone |
| **FP** | DECT Fixed Part (Base) |
| **HS** | DECT Handset |
| **MMI** | Man Machine Interface / User interface |
| **PP** | DECT Portable Part (Handset) |
| **DNS** | Domain Name System |
| **ULE** | DECT Ultra Low Energy |
| **RTX** | RTX A/S |
| **SW** | Software |
| **HW** | Hardware |
| **FP** | Fixed Part |
| **PP** | Portable Part |
| **ME** | Management Entity |
| **MM** | Mobility Management |
| **HSO** | Hochschule Offenburg (Offenburg University of Applied Sciences) |
| **UA** | ULE Alliance |
| **LL** | Link Layer, in this case the DECT ULE layer. (In some documents the DECT ULE layer is referred as Transport Layer, TL, which contradicts both the OSI and Internet model where the ULE layer would be Physical/Data Link layer and network layer respectively ) |

# Requirements

## Operating system

In order to run the code, a computer running Linux is required. This computer can either run Linux natively or via virtualization. In this description, Ubuntu 14.04.3 LTS 64 bit has been installed via VirtualBox virtualization. The software listed can be located here:

* VirtualBox  
  <https://www.virtualbox.org/wiki/Downloads>
* Ubuntu  
  <http://www.ubuntu.com/download/desktop>

### Virtual box installation

Installing VirtualBox is just as any other program. It is however recommended that the extension pack is added to virtual box after installation. The extension pack is located on the same download page as for VirtualBox itself. They are installed by choosing Preferences from the file menu, and choosing the extensions tab. Then simply add the downloaded extension.

Another key item is that in order to install 64 bit operating systems, Virtualization has to be enabled in the computers BIOS. If help is required for this step, please consult the computers manual, as it is different for every computer/BIOS.

### Installation of Ubuntu inside virtual box.

Create a new virtual machine inside virtual box and follow the instructions. The following parameters should be chosen along the process:

* Memory:  
  In order to provide a good performance, at least 2G is recommended. However it is memory which will not be available for the host system, so don’t overdo it. Its recommended to leave at least 4G of memory for the host system. On a 8Gb memory host computer, 3G is recommended.
* Create a virtual harddrive now and use variable size. Size should be around 30G.

Once this is done, enter properties and change the following settings on the VM:

* Screen
  + Graphics memory to 32M if possible.
  + Allow 2D and 3D acceleration
  + Storage units. Click the CD icon and choose the Ubuntu installation file.

Now the Virtual machine can be started and installation of Linux will begin. Once Linux is installed, from the VM window chose the Devices and insert the guest additions CD (virtual). Allow it to run, this will let it use the full size of the monitor, not just a tiny area.

### Required packages

A number of packages are required to be installed on the system in order to be able to compile the software. The names of these may differ between Linux distributions. The names listed here are for Ubuntu. A package is installed by executing this command:

*sudo apt-get install <package\_name>*

* make
* gcc

### IPv6 capable router

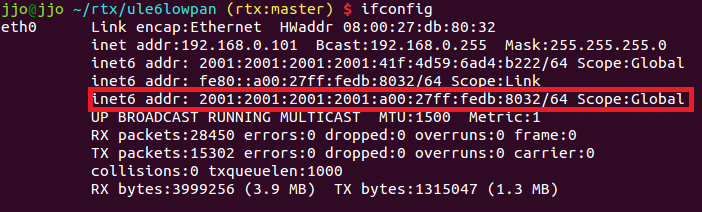
To be able to obtain a globally routable IP address via the border router functionality inside the library, a router is required.

### Router configuration

The router must be configured to use SLAAC (stateless auto configuration), but otherwise there are no special requirements. It is however recommended that the router is not connected on the wan side, and that it has only the testing computer connected on the LAN side. Due to the library not being fully stable, coupling it the the office network may have undesired side effects. Note that if using a virtual box, the network interface must be set to bridge mode to work.

### Verifying router setup

To verify the router is configured and working properly, check inside the Linux machine issuing the “ifconfig” command from a terminal. It must report a globally unique IPv6 address on the interface ( most likely eth0 ). IP addresses starting with FE80: can be ignored. Those are link local, and are not obtained via the router. See example below, valid IP is marked.



### Bridge the network to allow access for the library.

In order to enable the 6LBR access to the host’s network, a TAP interface must be created and it must be bridged with the Ethernet interface. A script has been made to accomplish this, it’s located in the app/6LBR folder and called setup\_bridge.sh. First the script must be modified to fit the environment.

* Change the Ethernet interface name from “eth0” to whatever the interface name is on the computer in question. Typically this already be correct.

Execute the script with sudo rights as this:

*sudo ./setup-bridge.sh <username>*

This will create the TAP interface, bridge it with the Ethernet interface and allow the 6LBR application to access it.

The username specifies who will be owner of the created TAP interface. This is to allow connection to the TAP interface without having to run the application as root.

The bridge can be taken down again using a tear down script:

*sudo ./tear\_down\_bridge.sh*

## MLD

Multicast is made available through MLDv1. It requires a router that has support for MLD to communicate out to the network. But from the LBR to the nodes multicast is possible without support in the router.

MLD should be disabled if not used, since it cost RAM and extra communication.

Out comment this in the makefile for both LN and LBR to disable it:

*CFLAGS += -DUIP\_CONF\_MLD=1*

## DNS server

In order to utilize the DNS features to resolve names to IP addresses, a DNS server must be provided by the router, and that DNS server must be available. Normally this will be already setup in the router and should not require any setup. However in test scenarios with isolated networks, this may not be the case. So here is an example of what has been done during development to use the feature.

NOTE: When using software based Router and DNS features as here, the network plug has to be removed from the PC, or the external router and the software router (radvd) may conflict with each other.

### DNS server listing in router advertisements

The router I had available (TP-Link N750) did not want to provide the DNS server(s) in the Router Advertisement(RA) messages. As this could be because I have no connection on the WAN side, I chose to my host computer (Linux) as router. This was done using the radvd software router feature. This can typically be found in the package “radvd”.

#### Installation of radvd

Install the package “radvd”:

*sudo apt-get install radvd*

#### Configuration of radvd

The required configuration changes are done in the file “/etc/radvd.conf” file. A copy of the configuration file used in this test is provided in the “/doc/radvd/” folder. That can either be used directly as long as RDNS server address is updated(in red), or used as inspiration. Besides that, the changes made compared to default radvd settings are shortly explained below.

* Setting the interface name

The outermost configuration is the interface name, which most likely defaults to eth0. For this purpose we shall use our bridge interface, which is named “br0”.

* Setting a prefix

A standard /64 prefix should be announced for the interface (br0), using these settings:

*prefix 2001:2001:2001:2002::/64*

*{*

*AdvOnLink on;*

*AdvAutonomous on;*

*AdvRouterAddr off;*

*};*

The actual prefix can of course be any valid global prefix. Do be careful to use either an official given prefix, or having the network isolated, to avoid address conflicts.

* Setting the RDNS server address

The IP address of the DNS server must be supplied. This will then be included in the RA messages. In this example the IP address of my host is used, so it must be modified to fit the IP address of the actual host. The Life time is in seconds, and is set aggressively to allow testing for timeouts.

*RDNSS 2001:2001:2001:2002:a00:27ff:fedb:8032*

*{*

*AdvRDNSSLifetime 30;*

*};*

* Inclusion of RDNS address in the RA messages.

To include the RDNS server address in the Router Advertisement messages, the following flag must be enabled:

*AdvOtherConfigFlag on;*

#### Starting radvd daemon

To start the daemon simply issue the following command:

*sudo radvd start*

#### Stopping radvd daemon

Stop it using this command:

*sudo killall radvd*

### DNS server

When running in an isolated network, it is not possible to use the public DNS servers typically used, such as google or an internet providers DNS. During development, the dns server daemon “dnsmasq” has been used. This is available as a daemon on Linux and is very easy to configure.

#### Installation

The dnsmasq is available in the package of the same name. So on Ubuntu, its installed by this command:

*sudo apt-get install dnsmasq*

#### Configuration

The required configuration changes are done in the file “/etc/dnsmasq.conf” file. A copy of the configuration file used in this test is provided in the “/doc/dnsmasq/” folder. That can either be used directly, or used as inspiration. Besides that, the changes made compared to default dnsmasq settings are shortly explained below.

* TTL  
  The time to live setting may be set to 0 (default if remarked), which means don’t cache the results obtained from this DNS server. This is not usable with Contiki’s interface, as it can only return a valid cached entry to the user application. Therefore set the TTL setting to a short time. In this example 30 seconds is used. Find the setting as shown below:  
    
  *local-ttl=30*
* Interface  
  The interface(s) to operate on must be specified. In this case only the br0 network bridge interface is used. Find the following section in the configuration file and enable the specified interface, and disable the DHCP feature on them.

*# If you want dnsmasq to listen for DHCP and DNS requests only on*

*# specified interfaces (and the loopback) give the name of the*

*# interface (eg eth0) here.*

*# Repeat the line for more than one interface.*

*#interface= eth0*

*interface= br0*

*#interface= tap0*

*# Or you can specify which interface \_not\_ to listen on*

*#except-interface=*

*# Or which to listen on by address (remember to include 127.0.0.1 if*

*# you use this.)*

*#listen-address=*

*# If you want dnsmasq to provide only DNS service on an interface,*

*# configure it as shown above, and then use the following line to*

*# disable DHCP and TFTP on it.*

*#no-dhcp-interface= eth0*

*no-dhcp-interface= br0*

*#no-dhcp-interface= tap0*

* Test translations

To be able to use the DNS server’s functionality, add one or more fixed translations to it. The example below lists the host pc (IP address will be different for you) having the name test.jjo. The www.thekelleys.org.uk is a default entry in the list, included here to help searching for the location in the configuration file.

*address=/www.thekelleys.org.uk/fe80::20d:60ff:fe36:f83*

*address=/test.jjo/2001:2001:2001:2002:a00:27ff:fedb:8032*

#### Start/stop/restart of DNS server

The DNS server uses the standard Linux service system, so start/stop/restart is done via these commands:

*sudo service dnsmasq start*

*sudo service dnsmasq stop*

*sudo service dnsmasq restart*

# Software structure

The software is placed in this folder structure:

* Apps
  + 6LBR
  + 6LN
* src
  + 6LBR
  + 6LN
  + Contiki
* Tools
  + DebugFunctions
  + FileLogger
  + LinkLayerAbs
    - 6LBR
    - 6LN

## Apps

The apps folder contains test applications, using the ule6lo library. The applications is command line based, and provides a UDP communication to other nodes. See later in the document for the usage of the application. There is one application for the 6LBR device and one for 6LN devices. This is also where the make files are located.

## Src

This contains the ule6lo library source code. The src is divided into 3 main blocks:

### 6LBR

This contains the public interfaces for accessing the ule6lo library for the 6LBR device. See [1] for a closer description of the interface. Most of the functions in this folder is simple wrapper access for the contiki source code.

### 6LN

This contains the public interfaces for accessing the ule6lo library for the 6LN device. See [1] for a closer description of the interface. Most of the functions in this folder is simple wrapper access for the contiki source code.

### Contiki

This folder contains a limited about of source code of the contiki open source project. This is the network stack (uip) and a limited amount of the general functions. At this time, there is quite a few more files in there, than is actually being used. These will be removed at a later time.

## Tools

This folder contains various tools, which is not meant to be a permanent part of the library, but which are very convenient during development.

### DebugFunctions

This folder contains a set of utility functions to obtain information about the status of ule6lo library. Most simply print to the command line. At this present time, they are required, as calls to these methods have been made inside contiki.

### FileLogger

A simple filer logger, which can be used to log information to a file. Used for assistance with debugging during development.

### LinkLayerAbs

Contains an abstraction layer, to emulate a link layer. In a final product this would be replaced by the Dect/ULE stack. At this time in the project there is no DECT layer, so it is simulated using IPv4 UDP messages. The link layer currently uses a hardcoded neighbor list, mapping between UDP destinations and IPEI numbers. This is located in the 6LBR link layer. The link layer uses the proper interface as much as is possible, but naturally requires its own initialization code.

# Compiling the code

There are 2 programs included in this delivery, placed in the apps/6LBR and apps/6LN folders. Compiling is done identical for both applications.

## Enter folder

From a command line change into the relevant application folder.

## Creating/updating dependencies

This step will eventually be handled automatically, but for now it requires a little help. This step must be done before compiling for the first time, and every time the make file is changed. To create/update the dependencies, execute this command:

*make deps*

## Compile the code

To actually compile the code, simply execute a standard make command:

*make*

# Running the applications

The currently supported scenario consists of 1 6LBR base station and 2 6LN nodes, all run on the same computer. It is recommended to use a separate terminal for each application.

## 6LBR

To start the 6LBR application, simply execute the application like this:

*./6LBR*

Usage instructions are printed upon starting, but are also explained here.

To send a message to one of the 2 6LN nodes, write the following command:

*<global\_ip\_address\_of\_6LN> <UDP port> <Message>*

This will send the message to the specified 6LN node. At the current time, port 3001 is hardcoded into the application, so this is the only allowed IP. Support for fixing typing errors are not yet supported. You can copy paste (middle mouse) though.

Example:

*fc00:5254:5800:0000:3e97:0eff:fe00:0001 3001 Hello world*

Multicast Example(all nodes address)

*FF02:0000:0000:0000:0000:0000:0000:0001 3001 Hello multicast*

Received messages will be printed out to the terminal. For convenience the 6LBR’s IP addresses as well as Multicast addresses and its known neighbors are printed to the console every 10 seconds. Hint: Copy the IP address from here. Messages can also be transmitted to the host PC or any other PC connected to the same border router.

Another feature has been added, which is to resolv a currently hardcoded entry “test.jjo”. To try it, type the command “resolv”. If the entry is known in the cache, it will return the matching IP address, if not it will attempt to resolve it. Replies on the command line.

## 6LN

To start the 6LN application, it is recommended to use the helper scripts created, as it requires a number of command line options. To start up a number of nodes, run the node.sh command with the number of nodes requested. Each node will launch in its own gnome-terminal. The example launches 5 nodes.

*./node.sh 5*

Usage instructions are printed upon starting, but are also explained here.

To send a message to one of the 2 6LN nodes, write the following command:

*<global\_ip\_address\_of\_destination> <UDP port> <Message>*

This will send the message to the specified destination, which can be either another node or the 6LBR. At the current time, port 3001 is hardcoded into the application, so this is the only allowed IP. Support for fixing typing errors are not yet supported. You can copy paste (middle mouse) though.

Example:

*fc00:5254:5800:0000:3e97:0eff:fe00:0001 3001 Hello world*

Received messages will be printed out to the terminal. For convenience the 6LN’s IP addresses and its known neighbors are printed to the console every 10 seconds. Hint: Copy the IP address from here. Note that it can only see the 6LBR as its neighbor, this is intentional.

When sending to another 6LN node, the data is first send to the 6LBR, which then forwards the information on to the request 6LN node. Messages can also be transmitted to the host PC or any other PC connected to the same border router.

Another feature has been added, which is to resolv a currently hardcoded entry “test.jjo”. To try it, type the command “resolv”. If the entry is known in the cache, it will return the matching IP address, if not it will attempt to resolve it. Replies on the command line.

## External tools

To test the UDP communication between the library and the host network, a couple of tools are suggested.

### Netcat

This tool (nc on the command line) can be used to send and receive UDP messages very easily. Although due to its internal way of handling connections, I do not recommend it for receiving. It is a little too restrictive, which could make it look like the library is not working, even if it is. So use it only for sending. To set it up to send messages to a destination, use this command:

*nc –u -6 <ip> <port>*

where IP is the globally unique IP address and port is 3001 (The one the test applications are setup to use). Messages can then be written and they will be transferred when pressing the enter key.

### udp\_server.c

This is a quick application I shamelessly stole on the internet. It listens on ports 3001 and 3002 and just prints what is received. It works perfect for the purpose of listening for the messages from the 6LBR and 6LN. To compile it simply execute this command:

*gcc udp\_server.c*

This creates the program as a.out, which can then be executed like this:

*./a.out*