Contiki JN516x start-up guide

Contiki TSCH for JN516x

Revision 1.0 — January 2016 **User Manual**

Purpose:

In the EIT Digital sponsored “Reliable IP for Channel Hopping networks” (RICH) activity, Contiki OS was ported to the NXP JN516x family of wireless microcontrollers and support for the IEEE 802.15.4e Time Slotted Channel Hopping (TSCH) MAC mode of operation was added together with elements of the emerging IETF 6TiSCH specifications.

This document is intended to get quickly started with Contiki and/or TSCH/6TiSCH minimal on the JN516x family of wireless microcontrollers.

1. Introduction

1.1 Introduction

Contiki OS is one of the leading open source operating systems for low power wireless networking solutions. Contiki networking is based on IEEE 802.15.4 for the MAC and PHY layer, the IETF 6LoWPAN adaptation layer, IPv4 and/or IPv6 with RPL routing for the network layer, and various other IETF standards for the higher layers such as UDP, TCP, and CoAP.

In the collaborative Reliable IP for Channel Hopping networks (RICH) activity (co-funded by EIT Digital) Contiki OS was ported to the JN516x family of wireless microcontrollers and support for the Time Slotted Channel Hopping (TSCH) mode of operation for the MAC, as specified in the IEEE 802.15.4e 2012 addendum, was added. Furthermore elements of the emerging IETF 6TiSCH specifications have been included.

This document aims to help getting started with Contiki OS, with or without TSCH, on the JN516x hardware. The document consists of the following parts:

- Architecture overview
- Installation of necessary software development tools
- Contiki examples on how to use the installed tools
- Q&A of the minimal TSCH with 6TiSCH stack
- Installation and configuration of a Border Router for 6TiSCH networks.

The reader is assumed to be familiar with IEEE 802.15.4e, TSCH, 6LoWPAN, IPv6 as well as the basics of Contiki OS. As the CSMA/CA MAC mode is well known and TSCH is relatively new, the examples will focus on using TSCH.

Development of Contiki is ongoing, which may lead to some discrepancies between this document and what you experience, these should be minor, however.

Support for the JN516x has been incorporated in the mainline Contiki (3.x) tree (see contiki-os/contiki/tree/master/platform/jn516x for port details). Support for TSCH has also been added to the mainline, see https://github.com/contiki-os/contiki/tree/master/core/net/mac/tsch for the implementation. Examples with TSCH can be found at https://github.com/contiki-os/contiki/tree/master/examples/jn516x .

The RICH development version contains some additional code, especially for CoAP TSCH scheduling resources (close to 6TOP); it is freely available from https://github.com/EIT-ICT-RICH/contiki.

1.2 Used abbreviations

6LoWPAN IPv6 over Low power Wireless Personal Area Networks

6Top 6TSCH Operation Sublayer

6TiSCH IPv6 over the TSCH mode of IEEE 802.15.4e

ASN Absolute Slot Number, a concept defined by TSCH

CoAP Constrained Application protocol

DAG Directed Acyclic Graph, part of RPL

DAO	Destination Advertisement Object, part of RPL
DIO	DAG Information Object, part of RPL
DIS	DAG Information Solicitation, part of RPL
DODAG	Destination Oriented Directed Acyclic Graph, part of RPL
EB	Enhanced Beacon, part of IEEE 802.15.4e
IE	Information Element, part of IEEE 802.15.4e
IETF	Internet Engineering Task Force
IPv6	Internet Protocol version 6
ND	Neighbor Discovery, part of 6LoWPAN (rfc6775)
PAN	Personal Area Network
RDC	Radio Duty Cycling
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks (rfc6550)
TSCH	Time Slotted Channel hopping, A MAC mode specified in IEEE 802.15.4e
UDP	User Datagram Protocol (rfc768)
VM	Virtual Machine

1.3 Useful links

1.3.1 JN516x

- NXP website with general references to JN516x related documentation.
- A description of the NXP JN516x Micro MAC API used by Contiki OS
- A description of the NXP on-chip peripherals API used by Contiki OS
- Information on JN516x Evaluation kit and the manual
- Hardware Description of base board DR1174 and shields (part of the Evaluation Kit)
- Reference manual for API of functions used to control functions on shields
- Manual for JN516x USB Dongle

1.3.2 Contiki

- Official Contiki site
- Contiki repository on GitHub.
- Contiki Wiki, provides valuable info on Contiki
- Contiki RICH repository on GitHub. Check here for introductions on Contiki RICH.

1.4 Acknowledgements

The Contiki port and TSCH implementation were largely done by a team of SICS Swedish ICT, a leading research institute for applied information and communication technology. Contributors include Atis Elsts, Beshr Al Nahas, and Simon Duquennoy.

Chapter 4, TSCH / 6TiSCH minimal Q&A, was originally written by Deepak Sudhakar from the Technical University of Eindhoven. Chapter 5, IoT Gateway Device as Border Router, is derived from a document by Pouria Zand from the university of Twente.

1.5 Trademarks

Beyond trademarks, service marks and logos of Beyond are property of Beyond Semiconductor Inc.

Linux is a trademark of the Linux Foundation

Firefox is a trademark of The Mozilla Foundation

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

OpenWrt is a trademark of Software in the Public Interest, Inc.

Ubuntu is a trademark of Canonical Ltd.

VMware Player is a trademark VMware, Inc.

VirtualBox is a trademark of Oracle

2. Overview

The Contiki OS platform implementation for the JN516x family of wireless micro-controllers is built on top of the base NXP libraries for accessing the MAC, on-chip peripherals, and AES coprocessor.

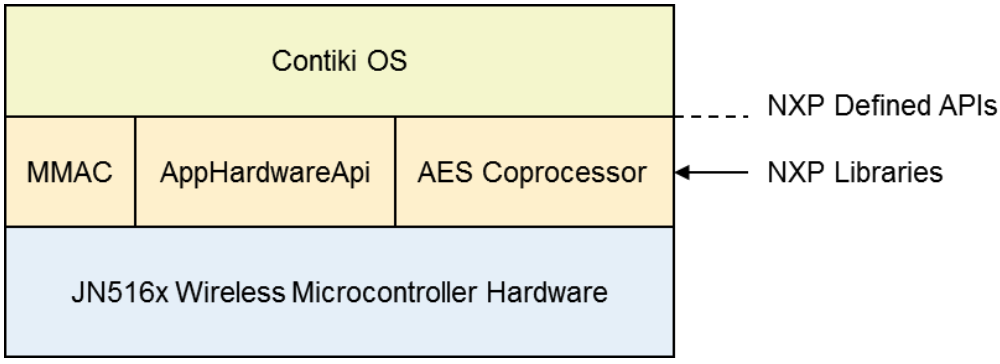


Fig 1. Contiki on JN516x architecture

The MAC is accessed through the MicroMAC (MMAC) API that abstracts from hardware register addresses and specific bit masks to ease porting to future hardware. The MMAC API is described in the ZigBee Green Power User Guide (JN-UG-3095.pdf).

The on-board peripherals such as Timers, UARTs, and Digital I/O are accessed through the AppHardwareApi. This API is described in the JN516x Integrated Peripherals API User Guide (JN-UG-3087.pdf).

Finally, if link layer security is enabled, Contiki uses the AES Coprocessor through the AES Coprocessor API. This API is described in JN-RM-2013.pdf

In order to build Contiki based software for the JN516x, the BeyondStudio for NXP Software toolchain (JN-SW-4141) is needed. All APIs are implemented in separate libraries that are provided with the NXP 802.15.4 Software Development Kit (JN-SW-4163) for JN516x. Both the toolchain and the SDK can be obtained free of charge from NXP as explained in detail below.

3. Installation of Tools and Libraries

In this chapter the following tool installations are described:

- **NXP JN516x Studio and SDK** - needed to perform builds for JN516x targets.
- **Git** - provides access to the RICH Contiki repository in Windows and Linux environment
- **Clone a Contiki repository** – the open source Contiki stack to work with.
- **Copper** - a CoAP client in the Mozilla Firefox web-browser.
- **Instant Contiki** – an out of the box, fully integrated Contiki development environment in a virtual machine.

3.1 JN516x Software Development Environment

The software development environment for Contiki for JN516x targets uses the Beyond Studio for NXP cross compilers running in a Windows environment and requires some base libraries as described in section 2.

3.1.1 Request access to the Beyond Studio for NXP and SDKs

To get the BeyondStudio for NXP Software toolchain (JN-SW-4141) and the JN516x IEEE 802.15.4 SDK (JN-SW-4163), permission is needed[1] from NXP Semiconductors, which can be requested in the following way:

- Go to: http://www.nxp.com/techzones/wireless-connectivity/ieee802-15-4
- Under **Software**, click **JN-SW-4141.**

Part Number	Title	Description	Version
Software			
JN-SW-4141	BeyondStudio for NXP	Toolchain for developing JN51xx applications.	1308
JN-SW-4163	JN516x IEEE802.15.4 SDK	Software for the development of IEEE802.15.4 networks based on JN516x devices. Must be installed on top of the BeyondStudio for NXP toolchain.	1307

Fig 2. JN516x SDE for Contiki

- A Request Software form is shown, that has to be filled out and submitted to NXP. The request is not just for JN-SW-4141, but for all relevant JN516x software development packages and only has to be requested once.

1. Note: For Export Control reasons NXP is legally obliged to validate all users of this software.

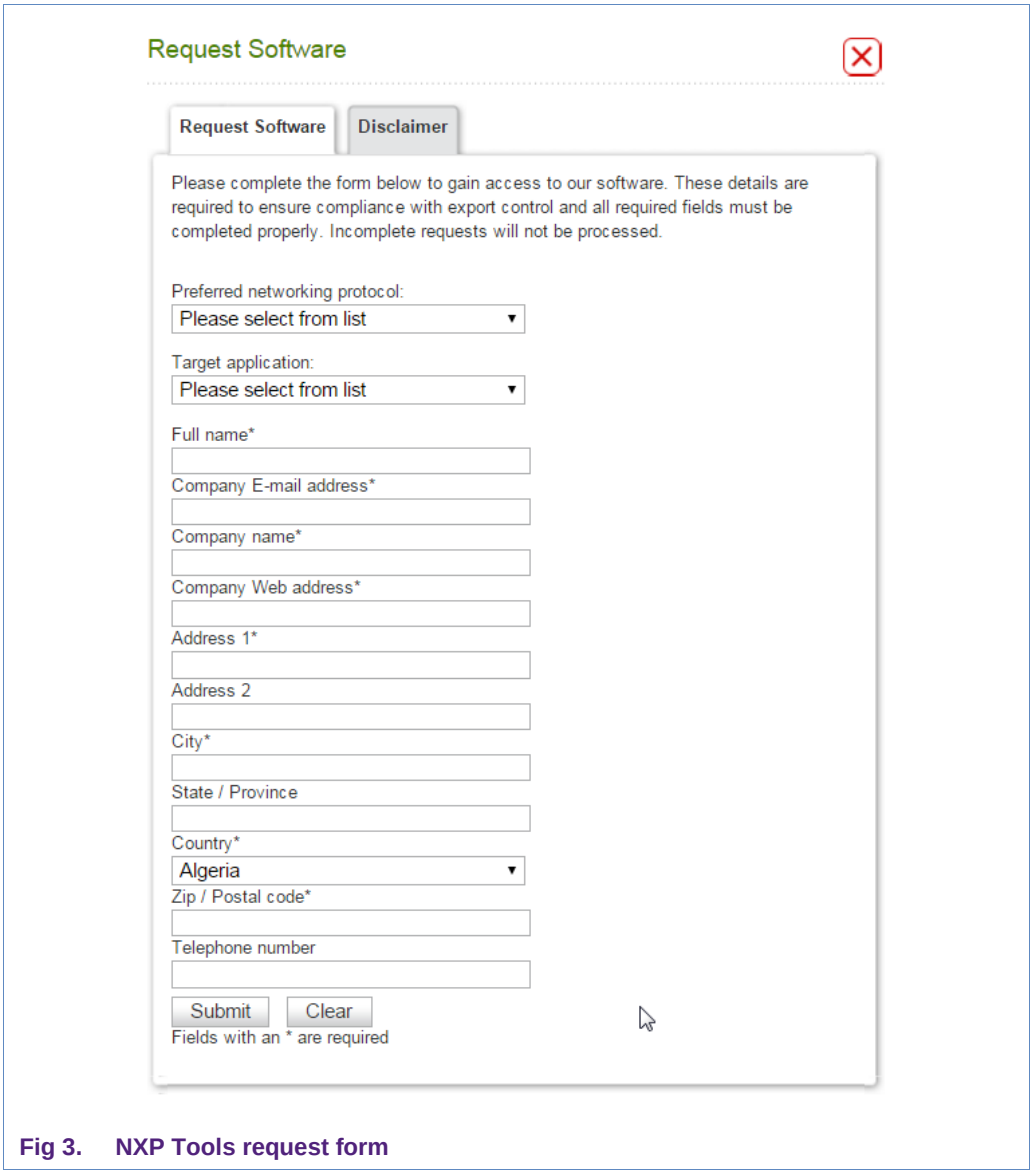


Fig 3. NXP Tools request form

- When the request has been granted[2], you will receive an email containing download links for the all packages. Fig 4 below shows an excerpt of such an email (note the exact content may change over time when packages get added or removed).

2. Typically within one working day

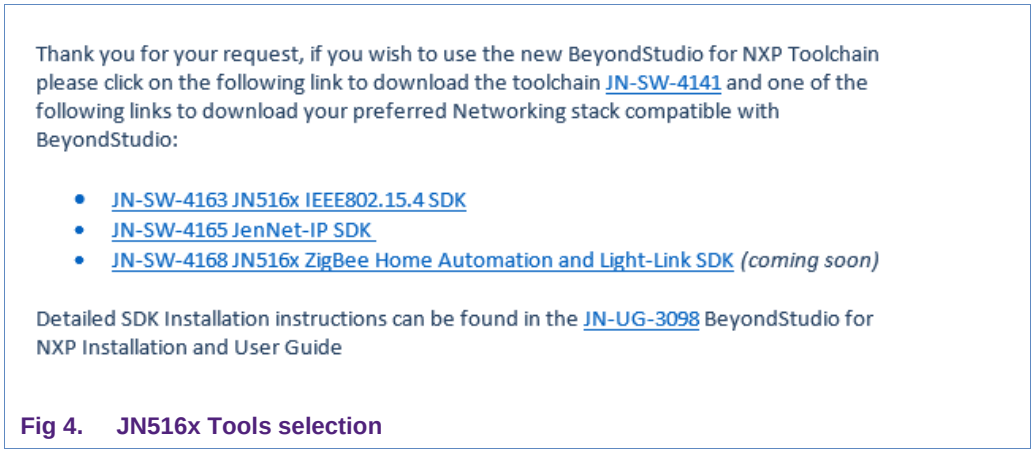


Fig 4. JN516x Tools selection

3.1.2 Install the Beyond Studio for NXP

The Beyond Studio for NXP contains all necessary tools, such as compilers, to build executable images for the JN516x family. It is based on GCC and uses Eclipse as software development environment (SDE).

The default installation directory is **C:\NXP\bstudio_nxp**. Details on the installation and usage of the Studio are described in the NXP User Guide JN-UG-3098, which can be retrieved from http://www.nxp.com/documents/user_manual/JN-UG-3098.pdf.

The Beyond Studio for NXP runs natively on Windows and uses the “Minimalist GNU for Windows” environment (MinGW) and the “Minimal SYStem” (MSYS) Bourne Shell command interpreter packaged with MinGW. MinGW provides, amongst others, the GNU Compiler Collections (GCC) C compiler for the JN516x wireless microcontrollers.

Use the link provided in the email from 3.1.1 to download the JN-SW-4141 toolchain installer. Both MinGW and MSYS get installed as part of the installation process.

Fig 5. JN516x Tools shortcuts

3.1.3 Install the JN516x IEEE 802.15.4 SDK.

The 15.4 SDK contains all the base libraries needed to be linked with the compiled Contiki code to form a complete image. The SDK is described in the JN-UG-3042 user guide which can be obtained from http://www.nxp.com/documents/user_manual/JN-UG-3024.pdf. Note that chapter 2 above provides the references for the documentation of all libraries used by Contiki.

Use the link for the JN-SW-4163 SDK from the email received in 3.1.1 to download and the required libraries. Use the same installation directory as for the Beyond Studio for NXP from the previous step.

3.2 Install Git

The Contiki open source software is maintained in a git repository on GitHub. To clone Contiki (i.e. make a private copy to work with and perhaps contribute additions or improvements) a git client is needed. Any of the following free clients for Windows will do:

- Git for Windows, the original command line client ported to Windows https://git-scm.com/downloads
- Git Extensions - http://gitextensions.github.io/
- git-cola - http://git-cola.github.io/
- SourceTree - https://www.sourcetreeapp.com/
- GitHub Desktop, for GitHub hosted repositories - https://desktop.github.com/

Note there are other options, some paid, some free for personal use.

For the examples in this Guide we use the GitHub Desktop which has a GUI client and a Git Shell. Please note that either of these or any other git client can achieve the same results.

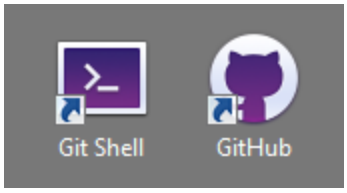


Fig 6. Git and GitHub for Windows

3.3 Install Contiki repositories

The Contiki stack is continuously under development. In order to get access to the latest sources, the GitHub cloud service is used.

At the time of writing this guide, there are two repositories that can be used, both hosted on GitHub:

- The mainline Contiki development at https://github.com/contiki-os/contiki
- The TSCH/6TiSCH Contiki development at https://github.com/EIT-ICT-RICH/contiki

The second is a clone from the first with some additions that have not (yet) made it back into the mainline.

If you just want to use Contiki OS on the JN516x, we advise you to use the mainline Contiki repository.

This guide describes the current public release of Contiki, but is equally valid for the Contiki RICH repository.

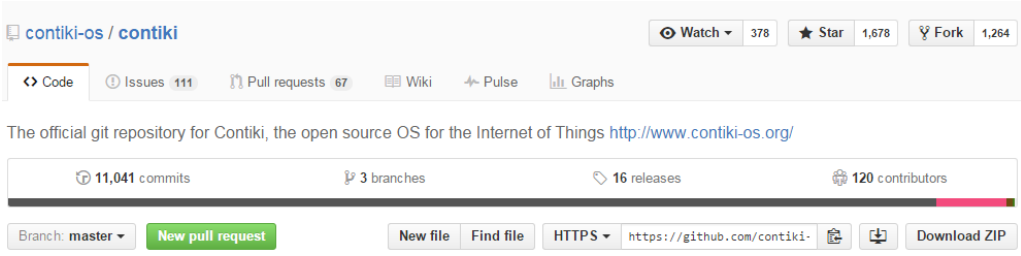


Fig 7. Contiki TSCH / 6TiSCH minimal on GitHub

- On your local Windows drive, create a directory to store the copies of the RICH repository. In this example: **C:/GitHubRepo**
- To get a local copy of the Contiki repository, first activate the **Git Shell** icon.
- Go to the directory of the repositories. In this example: **C:/GitRepo**
- The example uses a working directory **C:/GitRepo/contiki-os**.
  Create this directory.
- To download the master branch enter:
  git clone https://github.com/contiki-os/contiki contiki-os

```
C:\GitRepo> git clone https://github.com/contiki-os/contiki contiki-os
Cloning into 'contiki-os'...
remote: Counting objects: 90886, done.
remote: Total 90886 (delta 0), reused 0 (delta 0), pack-reused 90886
Receiving objects: 100% (90886/90886), 64.94 MiB | 8.89 MiB/s, done.
Resolving deltas: 100% (65200/65200), done.
Checking connectivity... done.
Checking out files: 100% (3965/3965), done.
C:\GitRepo> cd contiki-os
C:\GitRepo\contiki-os [master]>
```

Fig 8. Download contiki codebase

- The working directory will show the branch. If needed, the version can be verified with the **git branch** command.
- Because Contiki is still under development, regularly update the working area with:

  cd C:/GitRepo/contiki-os
  git pull

3.4 Verify Installation

Optionally you can now verify the installation of the tools and Contiki by building an example application:

- Open the **MSYS** shell
- Go to the Contiki examples directory, e.g.: **contiki-os\examples\jn516x\rpl\node**

- Enter: **make clean**
- Enter: **make TARGET=jn516x**

When the installation is correct, this should build without errors.

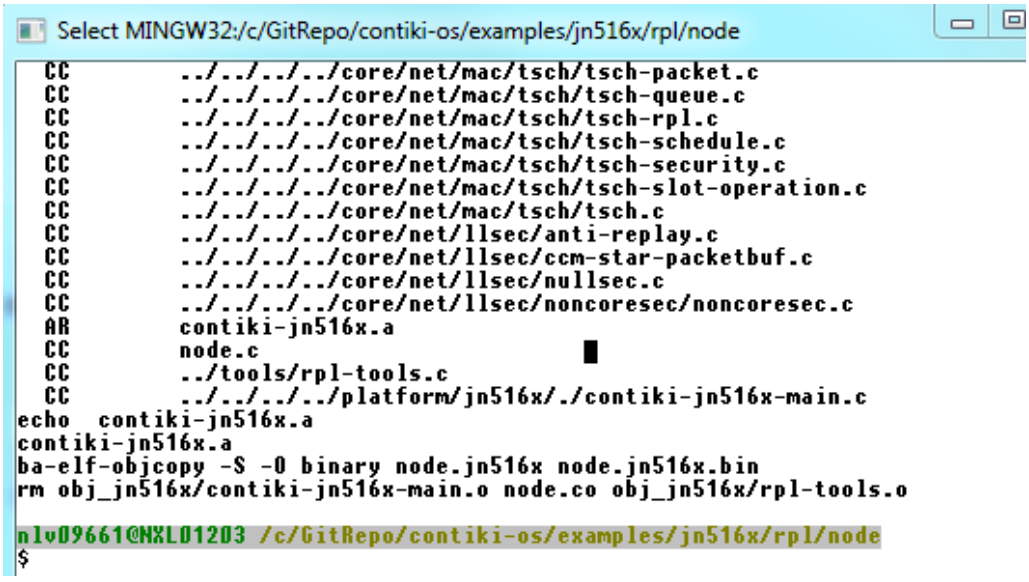


Fig 9. Contiki verification build

For making images for JN5169:

- Enter: **make TARGET=jn516x CHIP=JN5169**

If “CHIP=JN5169” is omitted, the image is built for JN5168 by default.

3.5 Install CoAP plug-in in web browser

End nodes may have application layers running on CoAP. On the IP side of the network, it may be useful to have a CoAP plug-in installed to monitor status and data at application level.

- If necessary download and install the Mozilla Firefox browser.
- Open the **Mozilla Firefox** web browser

Install the Copper CoAP user agent plug-in from https://addons.mozilla.org/en-US/firefox/addon/copper-270430/.

Fig 10. Installation of CoAP in web browser

3.6 Install Instant Contiki

Instant Contiki is a convenient development environment to simulate and build Contiki based targets. It runs under Linux Ubuntu in a VMware virtual machine environment.

The Contiki developers supply Instant Contiki pre-packaged in the form of a ready to run VMware virtual machine image, complete with all compilers and a cloned Contiki repository. To run this virtual machine image, VMware must be installed on the host machine (e.g. Windows).

Instant Contiki is primarily used for simulation purposes and to run a tunslip6 handler in case no hardware border router is available.

Follow the next installation procedure:

- Go to: http://contiki-os.org/start.html. This link provides download links for Instant Contiki and for the VMware Workstation Player virtual machine engine. At the time of writing, the latest version is Instant Contiki 3.0.

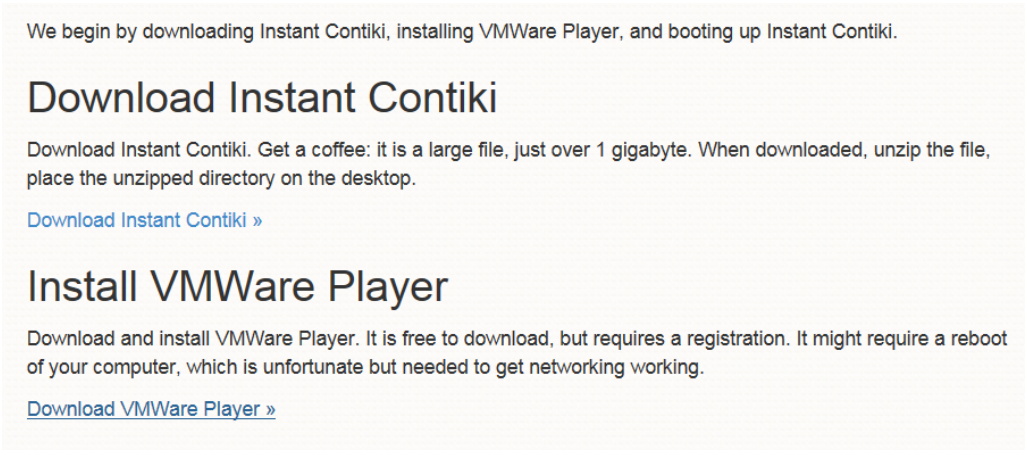


Fig 11. Instant Contiki

- Download and save **VMware Workstation player** from the link.
- Execute the **VMware installer**

  This version was VMware Workstation 12 Player version 12.1.0 build-3272444). Use default settings.
- Download and save the zip file with the latest **Instant Contiki** version. Extract the content of the zip file to the required directory.

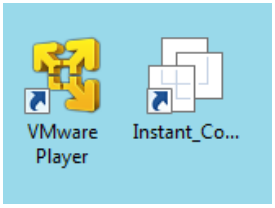


Fig 12. VMware Player and Instant Contiki shortcuts

- Start Instant Contiki by running **Instant_Contiki-Ubuntu_12.04_32_bit.vmx**.

  Note that the exact name depends on the Contiki version.
- After start-up, you may be prompted to download “VMware Tools for Linux”. Accept by pressing “Download and Install” button.
- Now the Ubuntu welcome screen opens; the password that has to be entered is “**user**”.

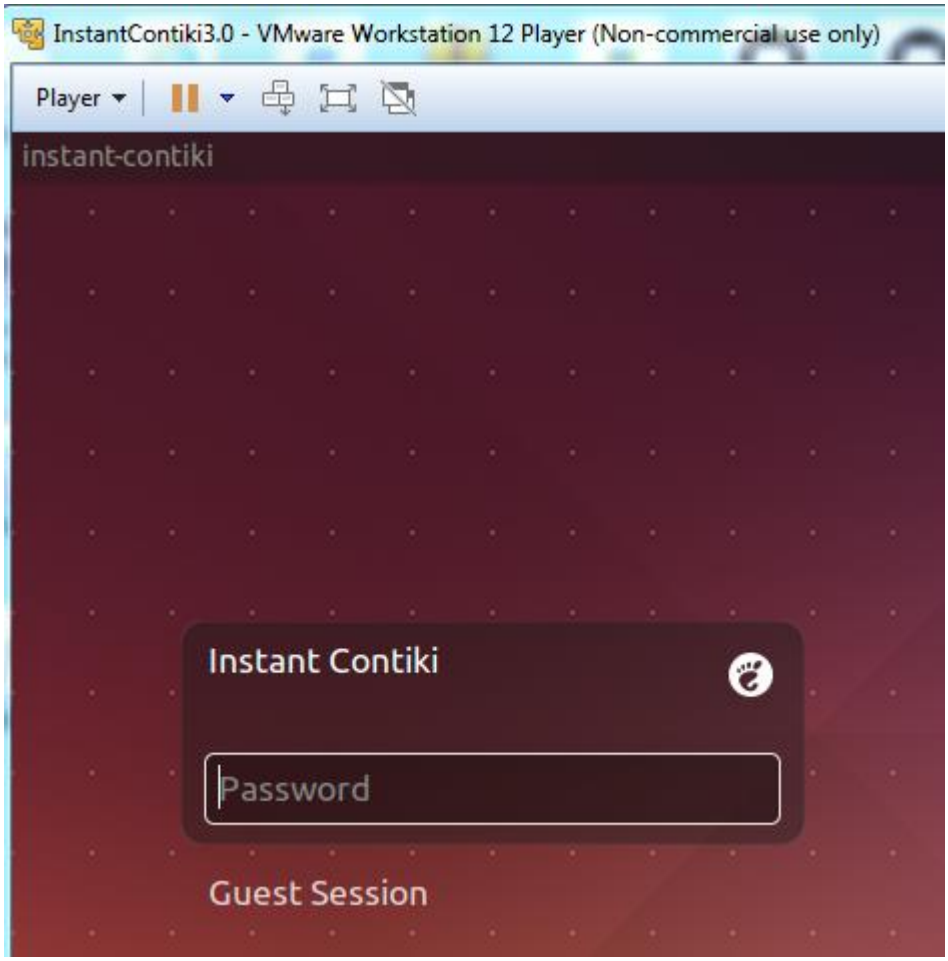


Fig 13. Instant Contiki logon screen

- Open a terminal. At the user prompt **user@instant-contiki** enter **ls.** One of the directories is **Contiki-3.0.** This is the current released version of the Contiki stack. Looking inside this directory the following set-up is shown:

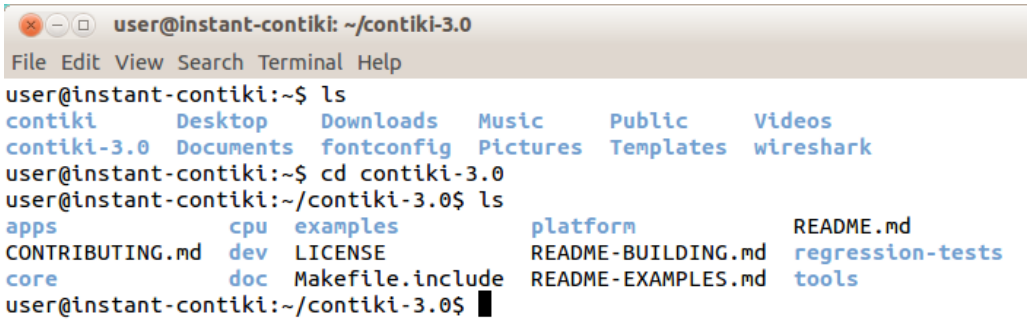


Fig 14. Instant Contiki directory content

- Instant Contiki is installed.
- Instant Contiki contains a network simulator called Cooja. The simulator can be started from the opening screen. Operation of Instant Contiki and Cooja can be verified by running the example that is shown in http://contiki-os.org/start.html.
- Note that the Instant Contiki resource does not yet support JN516x applications. The dedicated repository will be loaded later.

3.6.1 Contiki JN516x on Instant Contiki environment

This repository will be used for COOJA simulation and to provide a tunslip6 connection for the border router on a PC.

- Open Instant Contiki on VMware player
- Open a terminal
- To download the master branch enter:

  git clone https://github.com/contiki-os/contiki contiki-os
- Verify branch with: **git branch**

```
user@instant-contiki:~$ git clone https://github.com/contiki-os/contiki contiki-os
Cloning into 'contiki-os'...
remote: Counting objects: 90886, done.
remote: Total 90886 (delta 0), reused 0 (delta 0), pack-reused 90886
Receiving objects: 100% (90886/90886), 64.94 MiB | 673.00 KiB/s, done.
Resolving deltas: 100% (65200/65200), done.
Checking connectivity... done.
user@instant-contiki:~$ cd contiki-os
user@instant-contiki:~/contiki-os$ git branch
* master
user@instant-contiki:~/contiki-os$
```

Fig 15. Contiki with JN516x under Instant Contiki

To get updated for new commits, regularly call **git pull**

3.6.2 JN516x tools under Instant Contiki

For a number of builds under Instant Contiki (i.e. making tunslip6 for PC and COOJA), the JN516x tools are assumed to be present under Instant Contiki as well. The installation can be a simple copy of the Windows installation.

Note that the installation steps as described in ch.3.1 should be completed first, before doing this installation.

- In the Windows environment: make a zip file of the content of **C:\NXP\bstudio_nxp\sdk\JN-SW-4163** (Uninstall.exe may be excluded**).**

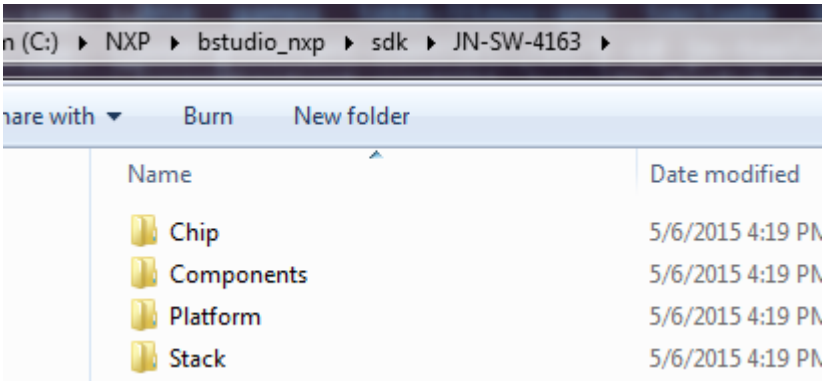


Fig 16. JN516x tools for Instant Contiki

- Copy this zip file (e.g. USB stick) to the Instant Contiki directory: **/usr/jn516x-sdk/JN-SW-4163**

3.7 Flash JN516x images

After a successful build, the resulting JN516x binary needs to be flashed in the hardware device.

- Naming of the target image: **<target-name>.jn516x.bin**
- Connect the serial interface of the JN516x target to a COM port
- Open Beyond Studio for NXP on the desk top
- When prompted for the Workspace, just click OK
- Wait while workbench is loading. When finished see the screen below.

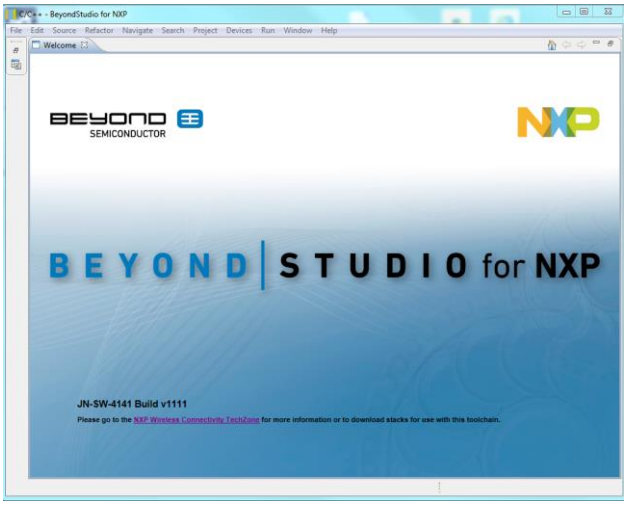


Fig 17. Beyond-Studio for NXP

- Enter: Devices -> Program Device (or shortcut: Ctrl+6)
- The Program serial device menu becomes visible.
- When properly connected, the MAC address of the device becomes visible at the bottom. Unless you have your own private range of MAC addresses and fully understand the consequences **do not modify the MAC address**
- Select the correct COM port.
- Insure “Preserve EEPROM” is selected if not explicitly wanting to modify the EEPROM contents.
- With **File System…** select <target-name.jn516x.bin

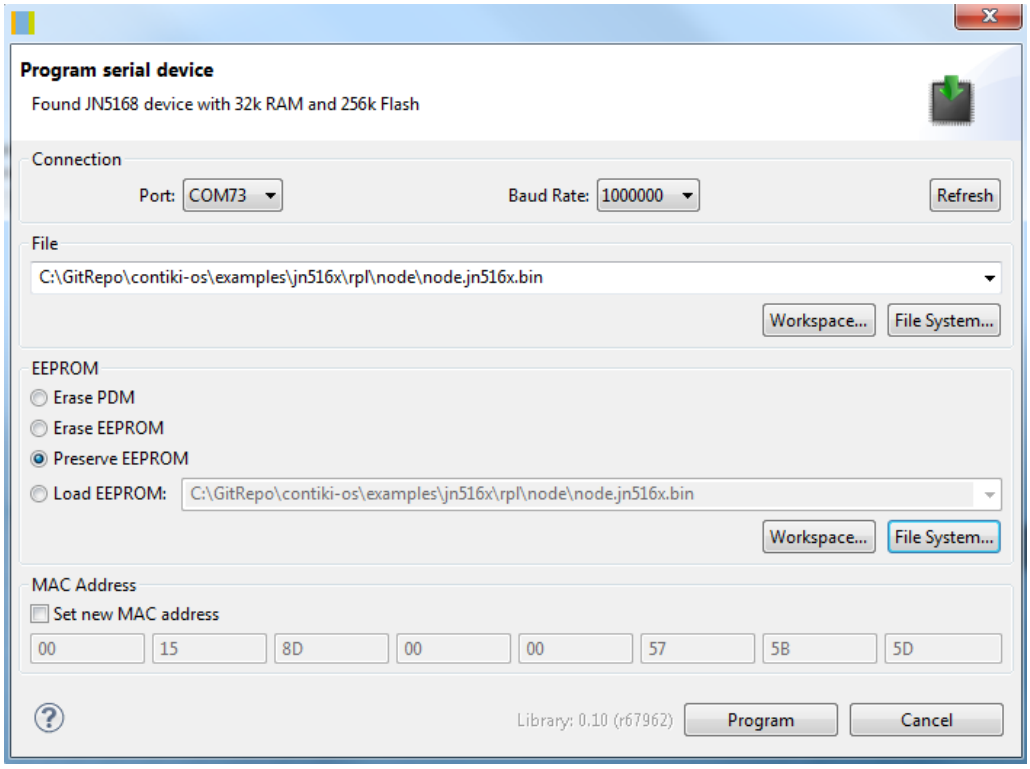
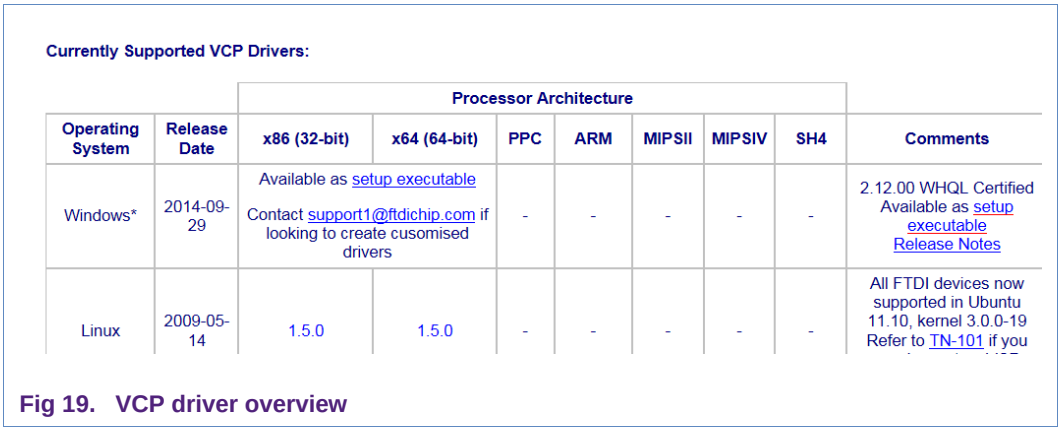


Fig 18. Menu for flashing JN516x images

- Press **Program** button to start flashing
- When programming has successfully finished, the Beyond Studio for NXP becomes visible again.
- If no COM port is shown and the target is properly connected to the host, then the right driver may have to be installed.

  Check in the **Device Manager** if the device gets detected. Plugging and unplugging the device should correspond with the presence of the **USB Serial Port** in **Ports (COM & LPT)**. If that is not the case, install the VCP device driver.

  - Go to: http://www.ftdichip.com/Drivers/VCP.htm.
  - Save the **setup executable** for Windows.



Currently Supported VCP Drivers:

Operating System	Release Date	Processor Architecture							Comments
		x86 (32-bit)	x64 (64-bit)	PPC	ARM	MIPSII	MIPSIV	SH4	
Windows*	2014-09-29	Available as setup executable Contact support1@ftdichip.com if looking to create cusomised drivers		-	-	-	-	-	2.12.00 WHQL Certified Available as setup executable Release Notes
Linux	2009-05-14	1.5.0	1.5.0	-	-	-	-	-	All FTDI devices now supported in Ubuntu 11.10, kernel 3.0.0-19 Refer to TN-101 if you

Fig 19. VCP driver overview

- Then run the downloaded file as Administrator; use default settings.
- When the COM port is OK, press the Program button to start flashing

3.8 JN516x example

This example shows the building of the firmware for the components to build a small network. It consists of an RPL border router and nodes operating as a small UDP node. The RPL border router image will have a SLIP interface to the PC. For the JN516x target, a UART/USB realizes the SLIP communication. In order to have an IPv6 network interface, a network adapter is installed to tunnel the SLIP traffic (tunslip6).

An example is shown to run on JN516x hardware targets; another examples shows how to run a COOJA simulation.

3.8.1 JN516x target

For this test JN516x dongles or DR1174 evaluation boards with a JN516x module may be used. These hardware targets are part of the NXP JN516x Evaluation Kit (see http://www.nxp.com/documents/leaflet/75017368.pdf).

3.8.1.1 Building an end node

- Open the MSYS shell and enter: **cd C:/**
- In these examples, the local copies are stored in **C:/GitRepo**
- **cd GitRepo/contiki-os/examples/jn516x/rpl/node**

  This is the directory to build a simple node. The source file for the node is: **node.c**
- For debug purposes it is good to have the UART configured for a baud rate that is recognized by terminals.

  Modify project_conf.h in GitRepo/contiki-os/examples/jn516x/rpl/node

```
#include "../common-conf.h"
//modified for debugging
#undef UART_BAUD_RATE
#define UART_BAUD_RATE UART_RATE_19200

#endif /* __PROJECT_CONF_H__ */
```

Fig 20. Node configuration

- Build:
  make clean
  make
- After a successful build, the output should finish as below:

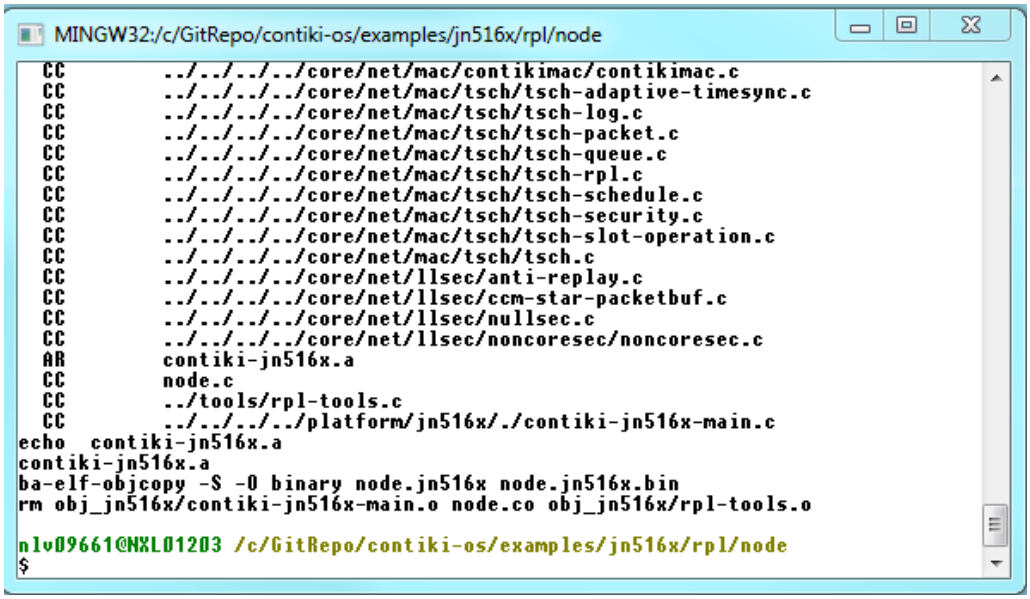


Fig 21. Building JN516x node

- A binary **node.jn516x.bin** is generated. Flash the binary in a target device as described in ch.3.6.2.

3.8.1.2 Building the Border Router

- Open the MSYS shell and enter: **cd C:/**
- In this examples, the local copies are stored in **C:/GitRepo**
- **cd GitRepo/contiki-os/examples/jn516x/rpl/border-router**

- This is the directory to build the rpl-border-router. The source file for the border-router is: **border-router.c.**
- Open **project-conf.h** of border-router and check if UART_XONOFF_FLOW_CTRL is enabled.

```
43
44  /* Disabling HW_FLOW_CTRL because of lack of support
45  #undef UART_HW_FLOW_CTRL
46  #define UART_HW_FLOW_CTRL  0
47
48  /* Enabling XON/XOFF */
49  #undef UART_XONXOFF_FLOW_CTRL
50  #define UART_XONXOFF_FLOW_CTRL 1
51
52  /* Baudrate for the IoT Gateway */
53  #undef UART_BAUD_RATE
54  #define UART_BAUD_RATE UART_RATE_1000000
55
56  #endif /* PROJECT_ROUTER_CONF_H_ */
```

Fig 22. Border router configuration

- Build:
  make clean
  make

- A binary **border-router.jn516x.bin** is generated. Flash the binary in a target device as described in ch.3.6.2.

3.8.1.3 Network set-up

The next step is to set-up the tunslip6 tunneling via a network adapter on a PC. We achieve by connecting a USB dongle that contains a JN516x running border router firmware via the network adapter on the host PC. The alternative of using a dedicated hardware border-router is described in ch.5 (IoT Gateway)

- Disconnect all devices (border-router, node(s))
- Open **Instant Contiki** and open a terminal window.
- Ensure a Contiki repository is available in this environment that supports JN516x. If not, then clone it (see ch.3.6.1).

  cd GitRepo/contiki-os/examples/jn516x/rpl/border-router
- The **Makefile** needs to be modified in order for tunslip6 to recognize the USB ports. See code snippet below for changes.

```
include $(CONTIKI)/Makefile.include

#using XON/XOFF flow control
connect-router-sw:      $(CONTIKI)/tools/tunslip6
#       sudo $(CONTIKI)/tools/tunslip6 -v1 -X -B 1000000 $(PREFIX)
        sudo $(CONTIKI)/tools/tunslip6 -v1 -X -B 1000000 -s /dev/ttyUSB0 $(PREFIX)

#using hw flow control
connect-router-hw:      $(CONTIKI)/tools/tunslip6
#       sudo $(CONTIKI)/tools/tunslip6 -v1 -H -B 1000000 $(PREFIX)
        sudo $(CONTIKI)/tools/tunslip6 -v1 -H -B 1000000 -s /dev/ttyUSB0 $(PREFIX)

#using no flow control
connect-router-no:      $(CONTIKI)/tools/tunslip6
#       sudo $(CONTIKI)/tools/tunslip6 -v1 -B 1000000 $(PREFIX)
        sudo $(CONTIKI)/tools/tunslip6 -v1 -B 1000000 -s /dev/ttyUSB0 $(PREFIX)

connect-router-cooja:   $(CONTIKI)/tools/tunslip6
        sudo $(CONTIKI)/tools/tunslip6 -a 127.0.0.1 $(PREFIX)
                                                                    Makefile
```

Fig 23. Makefile adaption

- Connect the border router (on JN516x target) to a USB port on the PC host.
- To verify presence of SLIP over USB enter:
  dmesg | grep USB
  This indicates a device connected on **ttyUSB0**

```
[ 9843.507265] ftdi_sio ttyUSB0: FTDI USB Serial Device converter now disconnected from
 ttyUSB0
[ 9847.705338] usb 2-2.1: new full-speed USB device number 5 using uhci_hcd
[ 9848.254613] usb 2-2.1: New USB device found, idVendor=0403, idProduct=6001
[ 9848.254618] usb 2-2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 9848.254620] usb 2-2.1: Product: JN5168 USB Dongle
[ 9848.317102] ftdi_sio 2-2.1:1.0: FTDI USB Serial Device converter detected
[ 9848.348578] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
user@instant-contiki:~/contiki-os$
```

Fig 24. Dongle detection

- Start the wireless network by entering:
  make connect-router-sw.
  Output see below. Notice IPv6 address of rpl-border-router in this set-up is aaaa::215:8d00:57:5b5d

```
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Waiting for prefix
*** Address:aaaa::1 => aaaa:0000:0000:0000
Waiting for prefix
Obtained prefix: aaaa::
Server IPv6 addresses:
 aaaa::215:8d00:57:5b5d
 fe80::215:8d00:57:5b5d
```

Fig 25. Border router at network setup

- Now connect a node. It may take some time to connect.

  In this example IP address of node is: aaaa::215:8d00:32:daf0. This can be derived from the node's MAC address or the network status that is regularly printed on the InstantContiki terminal screen (see Fig 26).

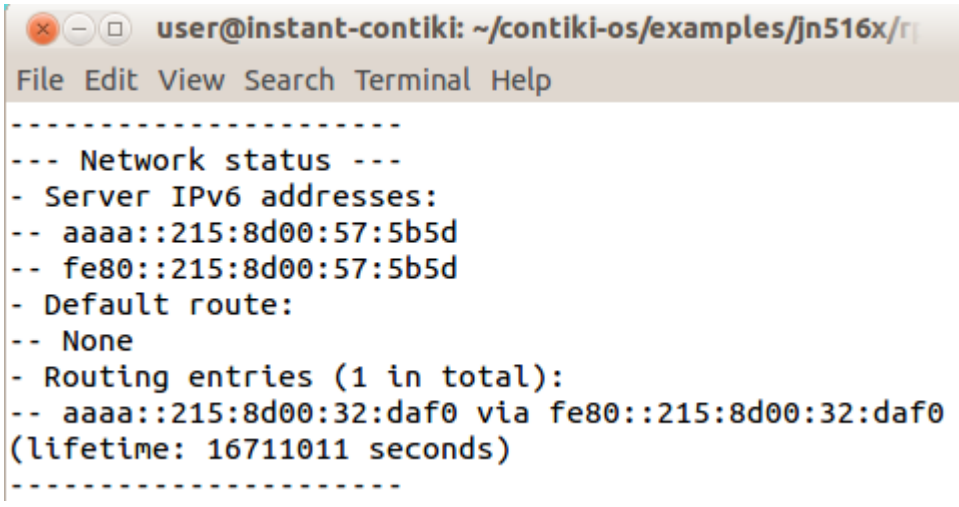


Fig 26. Network status from border router

- A route has been established between border-router and node.

3.8.1.4 Modifying example to TSCH

- The previous example did not use TSCH. In order to have the example use TSCH, copy **GitRepo/contiki-os/examples/jn516x/rpl/border-router** and **GitRepo/contiki-os/examples/jn516x/rpl/node** to **GitRepo/contiki-os/examples/jn516x/tsch/.** Make following addition (highlighted in Fig 27) in the Makefile for both node **and** border-router in the **GitRepo/contiki-os/examples/jn516x/tsch/** directory.

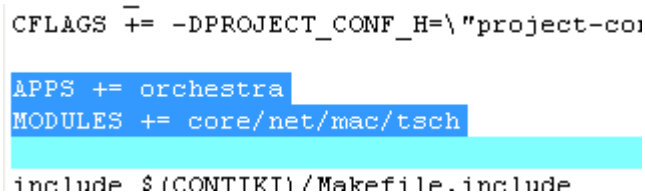


Fig 27. Makefile for TSCH example

- Build and flash the new the node example in **GitRepo/contiki-os/examples/jn516x/tsch/** as described in ch.3.8.1.1.
- Build and flash the new the border-router example in **GitRepo/contiki-os/examples/jn516x/tsch/** as described in ch.3.8.1.2.
- Execute the network setup as described in ch.3.8.1.3. Note that connecting the node to the border router will take longer with the current TSCH default settings. The output on the InstantContiki terminal screen is shown below. Note the dedicated TSCH logging (time slots, channel hopping …).

```
TSCH: starting as coordinator
Server IPv6 addresses:
 aaaa::215:8d00:57:5b5d
 fe80::215:8d00:57:5b5d
--- Network status ---
- Server IPv6 addresses:
-- aaaa::215:8d00:57:5b5d
-- fe80::215:8d00:57:5b5d
- Default route:
-- None
- Routing entries (0 in total):
----------------------
TSCH: enqueue EB packet 36 17
TSCH: {asn-0.b link-0-41-11-0 ch-25} bc-0-0 36 tx 0, st 0-1
TSCH: send packet to 255 with seqno 2, queue 0 1, len 15 95
TSCH: {asn-0.15e link-2-7-0-2 ch-17} bc-1-0 95 tx 0, st 0-1
TSCH: enqueue EB packet 36 17
TSCH: {asn-0.17c link-0-41-11-0 ch-19} bc-0-0 36 tx 0, st 0-1
TSCH: enqueue EB packet 36 17
```

Fig 28. Logging output of border router with TSCH at start-up

- When the serial port of the node is connected to a COM port of the host, TSCH logging of the node can be monitored as well.

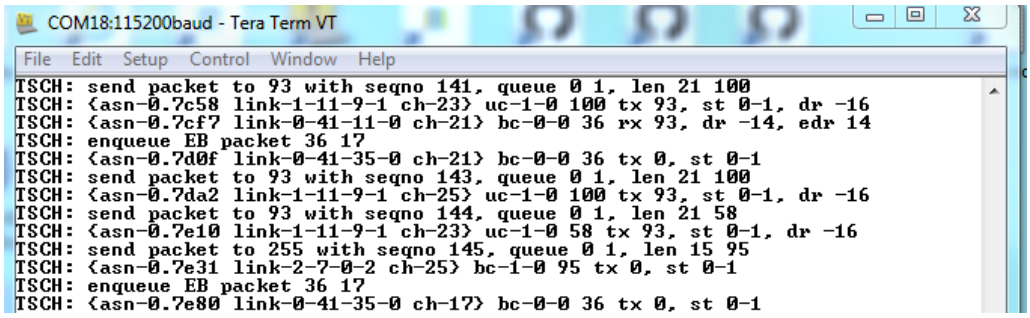


Fig 29. Logging output of node with TSCH

3.8.2 Cooja simulation

Simulation of network behavior on COOJA is at the time of writing only possible for **TARGET=z1**. Note that the Z1 has limited memory size, limiting the size of the applications that can be simulated. The example will use the RPL example.

- Open a terminal in your **Instant Contiki**
- Build node:
  cd**~/contiki-os/examples/jn516x/rpl/node**
  make clean
  make TARGET=z1

  After a successful Z1 build:

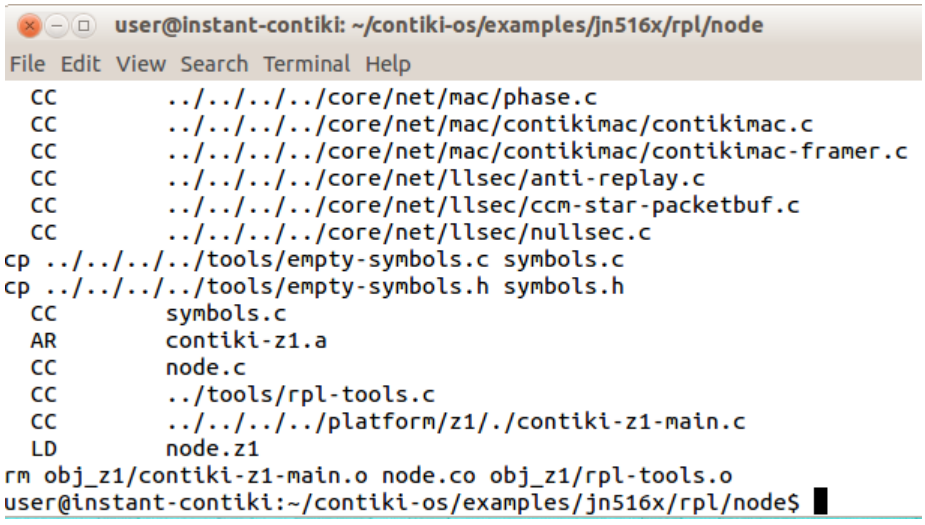


Fig 30. Z1 build

- Build border-router:
  cd**~/contiki-os/examples/jn516x/rpl/border-router**
  make clean
  make TARGET=z1
- Open COOJA:
  Open a new terminal window and enter commands below:

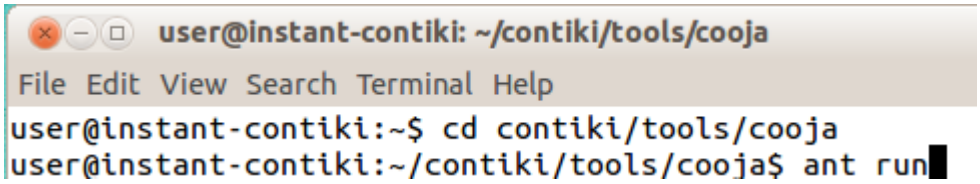


Fig 31. Starting COOJA

- This will result in the COOJA opening screen.
  The first time you invoke COOJA you may be requested to call:
  git submodule update --init
  Follow instructions.
- **File->New Simulation->Create**
- Create the border-router:

  Motes->Add motes->Create new mote type->Z1 mote->Browse.

  Brows to and open: **contiki-os /examples/jn516x/rpl/border-router/border-router.z1**
- **Create->a new mote->Add motes**
- The **Network** window shows the new border-router. Next step is to make IP connection to the host (tunslip6).
- Right-click on the border-router in the **Network** window.
- **Mote tools for z1->Serial Socket(SERVER)->Start**

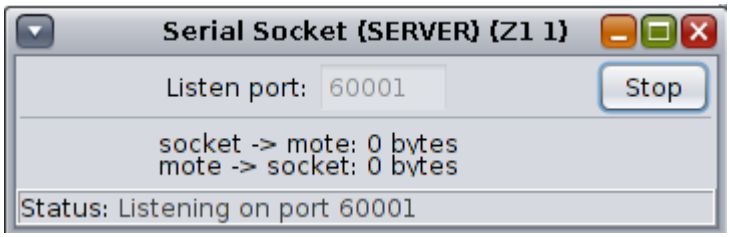


Fig 32. TSCH - COOJA simulation - 1

- Press **Start** in the **Simulation control** window. Border requests for prefix; we have to connect the border-router to an 'IPv6 network'.

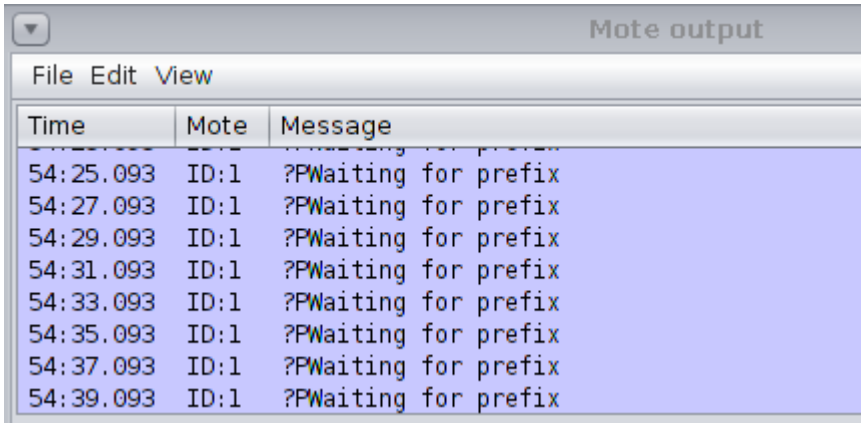


Fig 33. TSCH - COOJA simulation - 2

- In the **Network** window:

  View->Address: IP or Rime
  View->Radio traffic

- Open a new terminal window and open to the border router with:
  cd ~/contiki-os/examples/jn516x/rpl/border-router
  make connect-router-cooja

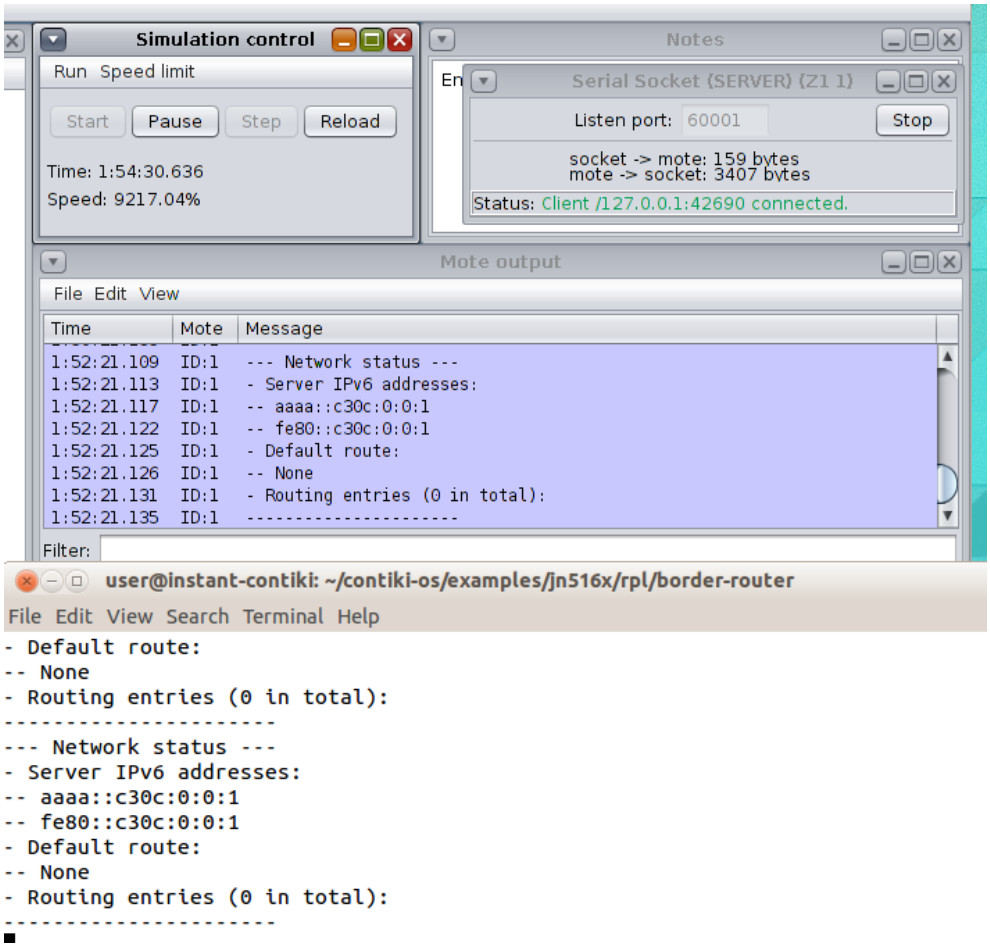


Fig 34. TSCH - COOJA simulation - 3

- Create e.g. two additional nodes:

  Motes->Add motes->Create new mote type->Z1 mote->Browse

  Browse to and open: **contiki-RICH-3.x /examples/rich/node/node.z1**
- **Create->2 new motes->Add motes**
- Press **Start** in **Simulation control** window
  Wait till 2 nodes connect. The network status is regularly output by the devices and visible in the Mote output window of COOJA.
- This example shows mainly RPL traffic

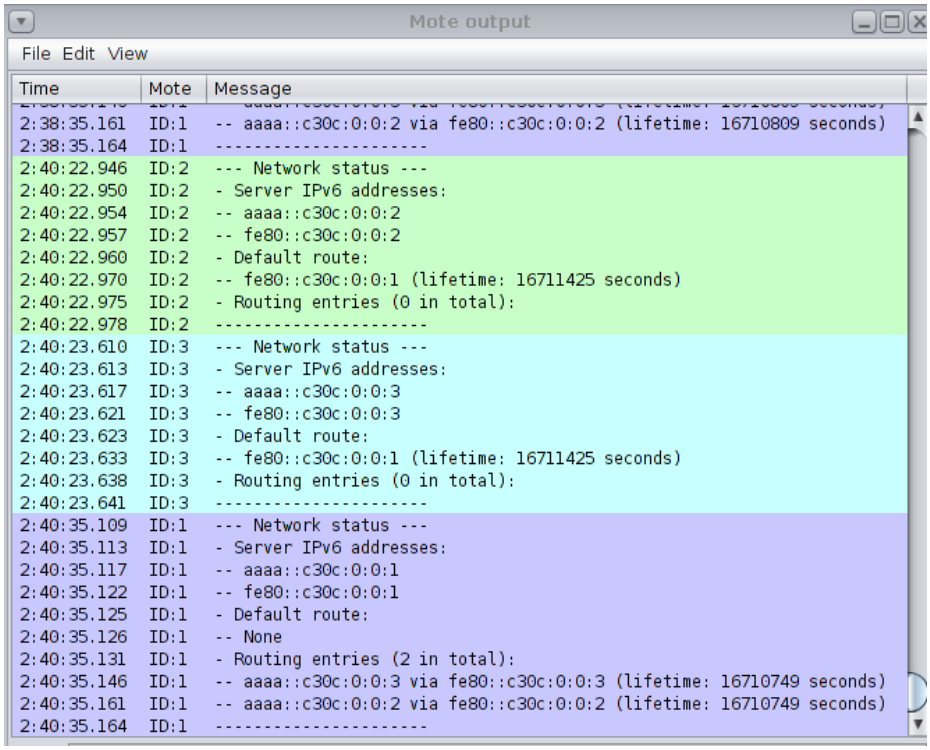


Fig 35. Output generated by the motes in COOJA

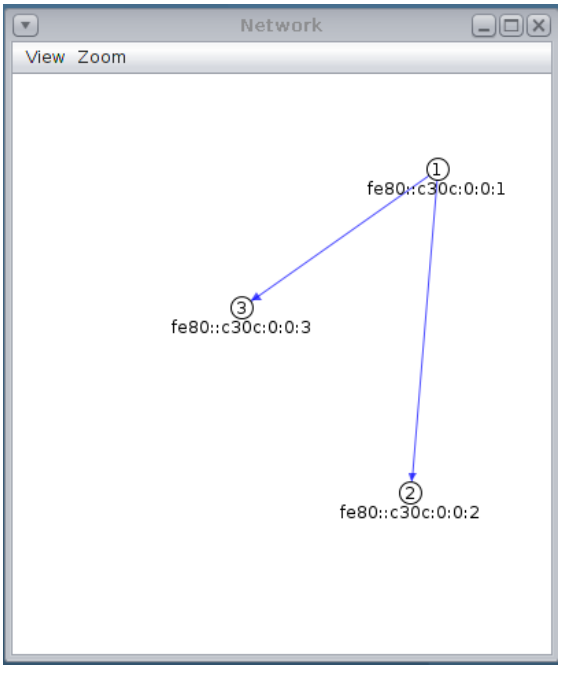


Fig 36. Radio traffic of the motes in COOJA

4. TSCH / 6TiSCH minimal Q&A

While the previous examples were derived from the Contiki stack to check proper functioning, this section will explain the TSCH and minimal 6TiSCH network configuration. Also some more explanation will be given of how packets pass through the stack, and how the network is formed.

The structure of this chapter is organized as a Q&A. Some of the shown output is the result of COOJA simulations.

In this chapter a number of references are made to CoAP based TSCH scheduling. For Contiki v3.x, the external scheduler and the related CoAP resources are under construction. Alternatively, the 6top like interface (core/net/mac/tsch/tsch-schedule.h) may be used to locally schedule links for the application.

4.1 What is minimal 6TiSCH Configuration

Contiki's current TSCH implementation is based on Minimal 6TiSCH Configuration defined by IETF working group (IPv6 over the TSCH mode of IEEE 802.15.4e). It describes the minimal set of rules to operate an IEEE802154e Time Slotted Channel Hopping (TSCH) network. This minimal mode of operation can be used during network bootstrap, as a fallback mode of operation when no dynamic scheduling solution is available or functioning, or during early interoperability testing and development.

Note that the 6TiSCH working group is still actively developing their (draft) specifications, so the reality of the specification may not (anymore) correspond to the description in this document. Still it should give you a head-start.

4.2 What is the 6TiSCH stack configuration?

The typical stack configuration of a 6TiSCH-like network and looks as Fig 37. This figure also shows the standardization bodies that create the specifications.

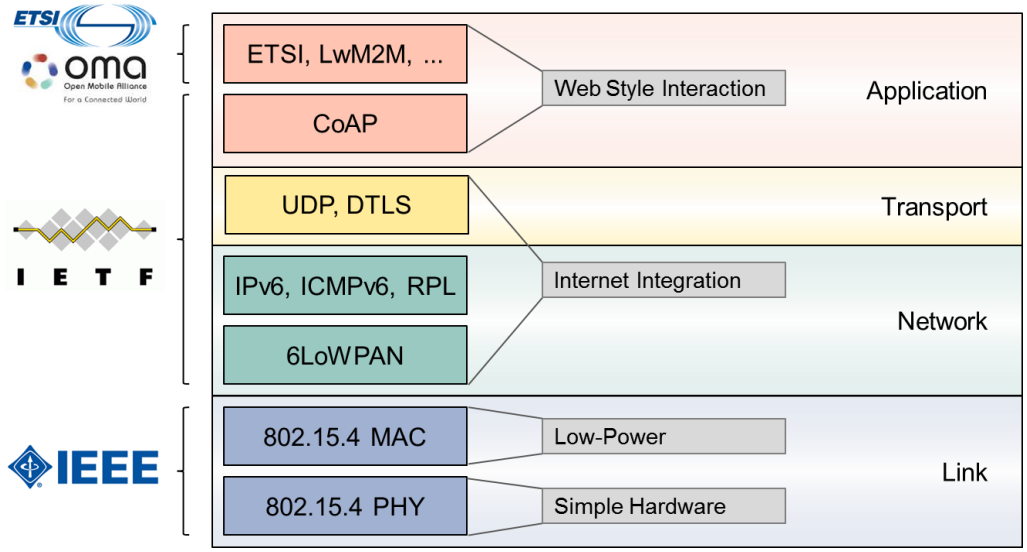


Fig 37. 6TiSCH network stack

The following table shows the various Contiki #defines that can be used to configure Contiki stack layers. The table shows the settings for a JN516x based IPv6 stack with TSCH.

STACK	Contiki CONFIGURATION
UDP	**#define** UIP_CONF_UDP 1
RPL uIP6 6LoWPAN	**#define** UIP_CONF_IPV6_RPL 1 **#define** UIP_CONF_IPV6 1 **#define** NETSTACK_CONF_NETWORK sicslowpan_driver
llsec[3] 6Top MAC RDC Radio	**#define** NETSTACK_CONF_LLSEC nullsec_driver **#define** NETSTACK_CONF_MAC tschmac_driver **#define** NETSTACK_CONF_FRAMER framer_802154 **#define** NETSTACK_CONF_RDC nordc_driver **#define** NETSTACK_CONF_RADIO micromac_radio_driver

4.3 How does a received packet propagate through the stack?

The following table describes the processing of an incoming packet.

OSI Model	Contiki Net Stack	Functions
Link Layer	RADIO: MICROMAC-RADIO /platform/jn5168/dev/micromac-radio.c	In TSCH, radio interrupts are disabled. The node which intends to receive a packet wakes up in the time slot with the help of timer interrupts and copies the packet from the radio Rx buffer to input ring buffer and polls `tsch_rx_process_pending()` in `tsch_pending_events_process()`.
	MAC: TSCH core/net/mac/tsch/tsch.c	`tsch_pending_events_process()` On received packet, `packet_input()` is called. It parses the input packet by calling `framer_802154.parse` and obtains the packet attributes to check whether the packet is intended for it and also performs duplicate packet detection with the help of sequence numbers. Calls `sicslowpan_driver.input()` via `nullsec_driver.input()`
Network Layer	Adaptation: 6LoWPAN core/net/ipv6/sicslowpan.c	`sicslowpan_driver.input()` The 6lowpan packet is put in `packetbuf` by the MAC. If it's a frag1 (first fragment) or a non-fragmented packet, the IP header is uncompressed. The 6lowpan payload and possibly the uncompressed IP header are then copied in `siclowpan_buf`. If the IP packet is complete, it is copied to `uip_buf` and the IP layer is called: `tcpip_input()`.
	Network: IPv6	`tcpip_input()`

[3] Note: the JN516x supports hardware based security and this can be enabled in Contiki

OSI Model	Contiki Net Stack	Functions
	core/net/ip/tcpip.c	A `PACKET_INPUT` event is generated for the tcpip_process. If a packet is present in the uip buffer with a `uip_length` > 0, the input handler function, `uip_input`, is invoked.
Transport Layer	Transport: UDP /core/net/ip/tcpip.c and /core/net/ipv6/uip6.c	`uip_input()` The input handler function parses the packet in the uip buffer to check the IP header and checksum. It checks if the packet is destined for its IP address and if the protocol field in the header matches (UIP_PROTO_UDP = 0x11), then jumps to `udp_input`: `udp_input`: Performs a UDP header checksum check and checks if the UDP destination port number in the udp buffer is non zero. It de-multiplexes the UDP packet between the multiple active UDP ports based on the destination port, source port and source IP address. After identifying the matching UDP connection, it forwards it to the application associated with the matching connection UIP_UDP_APPCALL. For a CoAP ports this is defined to be the function `tcpip_uipcall()` in net/ip/tcpip.c. The function will generate a `tcpip-event` to the registered CoAP process 'coap_engine' (part of `uip_conn` structure that was set up with `udp_bind()` when defining the CoAP socket)
Application Layer	Application: CoAP /apps/er-coap/er-coap-engine.c and /apps/er-coap/er-coap.c	`Coap_engine`: `coap_parse_message()` is called on a `tcpip_event` when new data is available. Parses the CoAP requests to identify the type of transaction, type of resource and method to validate it acts as a CoAP server.

4.4 How does a transmit packet propagate through the stack?

The following table describes the processing of an outgoing CoAP message.

OSI Model	Contiki Net Stack	Functions
Application Layer	Application: CoAP /apps/er-coap/er-coap.c	A CoAP based application creates a packet with CoAP header and payload and calls `coap_send_message()` which creates a UDP connection with the destination port and address and calls `uip_udp_packet_send()`
Transport Layer	Transport: UDP core/net/ip/uip-udp-packet.c	`uip_udp_packet_send()` Copies the packet into the uip buffer and adds UDP and IP header (`uip_process()` in net/ipv6/uip6.c). Once the above steps are done it calls the `tcpip_ipv6_output()` function.

OSI Model	Contiki Net Stack	Functions
Network Layer	Network :IPv6 core/net/ip/tcpip.c	`tcpip_ipv6_output()` Checks if route to the destination address exists and calls `sicslowpan_driver.output`() which was installed with `sicslowpan_init()`.
	Adaptation: 6LoWPAN core/net/ipv6/sicslowpan.c	`sicslowpan_driver.output()` The IP packet is initially in uip_buf. Its header is compressed and if necessary it is fragmented. The resulting packet/fragments are put in `packetbuf` and delivered to the 802.15.4 MAC by calling `tschmac_driver.send()` via `nullsec_driver.send()`.
Link Layer	MAC: TSCH core/net/mac/tsch/tsch.c	`tschmac_driver.send()` Calls `NETSTACK_FRAMER.create()` function to add an 802.15.4e specific MAC header to the IP packet and copies the packet in `packetbuf` to the neighbor queue list. Later, when a suitable TSCH slot is executed (in interrupt context with the CPU awoken from timer), the packet will be actually sent by calling `micromac-radio.prepare()` followed by `micromac-radio.transmit()`.
	RADIO: MICROMAC-RADIO /platform/jn516x/dev/micromac-radio.c	`micromac-radio.prepare()` Pushes the packet into the radio Tx buffer from the queue. `micromac-radio.transmit()` sends the packet

4.5 How to configure a node as PAN coordinator?

In PAN network there should be one and only one device configured as coordinator. There are 2 option to do this:

- Call `NETSTACK_RDC.off(1)` in your initialization (see **examples/jn516x**), or
- `#include "net/mac/tsch/tsch-private.h"`

  Set the global variable `tsch_is_coordinator = 1;`

If you want to use the minimal 6TiSCH schedule, ensure that the `TSCH_CONF_WITH_MINIMAL_SCHEDULE` define is set to 1.

4.6 How does RDC work in minimal 6TiSCH?

In the minimal 6TiSCH implementation all nodes wake up during the first time slot of the slot frame (active time slot), and sleep during the rest of the slot frame duration as shown in Fig 38. Pending transmissions are queued and transmitted in the first time slot of the subsequent slot frames. The slot frame length (expressed in number of time slots) is configurable. The length used determines the duty cycle of the network. For example, a network with a 0.99% duty cycle is composed of a slot frame of 101 slots, which includes 1 active slot.

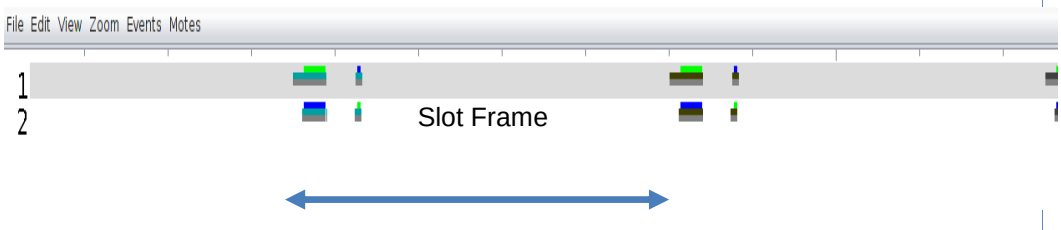


Fig 38. Slot frame definition

Note: in the Contiki 6TiSCH implementation, one can create links in the unscheduled time slots in a slot frame with the help of the API described in ch.4.18.

4.7 Which information elements are present in an Enhanced Beacon?

The coordinator and router nodes periodically sent an enhanced beacon (EB) frame to allow other nodes to join the TSCH network. It is the first step in network formation. A node that likes to join the network listens for EBs. Upon receiving the EB from the coordinator or another router, it parses the information present in the EB packet to learn the slot frame and active time slot information as well as join priority from the **Synchronization IE.**

It obtains the time slot template information (TsDuration=10ms default, offset etc.) from the **Timeslot IE.**

It obtains the hopping sequence list and the number of channels from the **Channel hopping IE.**

It obtains the number of slot frames and the respective links associated from the **TSCH slotframe & link IE.**

See `struct ieee802154_ies` in **core/net/mac/tsch/frame802154e-ie.h** which IE's are supported.

4.8 How frequently are Enhanced Beacon sent?

There is no fixed period for EB transmission. In the minimal configuration, EBs are transmitted in the first time slot (active time slot) of the slot frame.

At initialization, the EB period is set to TSCH_EB_PERIOD. If a frame with an EB has been transmitted, the next frame that will be assigned an EB, will be after a random period in the interval between 0.75*EB period and 1*EB period.

This randomness tries to avoid that two nodes send EB frames in the same active time slot.

At run-time, the function `tsch_set_eb_period` (in **core/net/mac/tsch/tsch.h**) can be used to change the EB period.

4.9 What are the implications of minimal TSCH?

Minimal TSCH means only 1 active time slot is used in the slot frame. As a consequence, both EBs and data share the same time slot. If the coordinator sends an EB, and a child

node sends data to the coordinator, it will occur in the same time slot. The result will be data collision and packet loss.

Fig 39. Packet Collison in Active time slot

Both radios of coordinator and child node are transmitting during the same time slot (blue transmission). When the node does not receive an ACK in a shared slot, it performs a CSMA exponential back-off.

When the child node wants to send packets to the coordinator at very short intervals which are small in comparison to the slot frame duration, the queue will store these packets. The oldest packet in the queue will be sent in the next active time slot of the slot frame. This will result in latency. If the packet transmission rate is very high it might result in overflow of the queue and the packet cannot be added in the queue anymore.

4.10 What is minimal 6TiSCH slot frame and time slot configuration?

In the minimal configuration, there is only a single active slot in the slot frame, used to transmit data and EBs, and to receive information. The trade-off between bandwidth, latency and energy consumption can be controlled by choosing a different slot frame length. The active slot **may** be scheduled at the slot Offset 0x00 and channel Offset 0x00 and **must** be announced in the EBs. EBs are sent using this active slot to the link-layer broadcast address (and are therefore not acknowledged).

Minimal configuration

Property	Value
Number of time slots per Slotframe	Variable
Number of available frequencies	16
Number of scheduled cells	1 (slotOffset 0) (macLinkType NORMAL)
Number of unscheduled cells	The remainder of the slotframe
Number of MAC retransmissions (max)	3 (4 attempts to tx)

Fig 40. Minimal 6TiSCH Slot frame configuration

The present minimal TSCH document recommends the use of a default slot duration set to 10ms and its corresponding default timeslot timings defined by the [IEEE802154e]

macTimeslotTemplate. The use of the default macTimeslotTemplate must be announced in the EB by using the Timeslot IE containing only the default macTimeslotTemplateId. Other time slot durations may be supported and must be announced in the EBs. If one uses a timeslot duration different than 10ms, it is recommended to use a power-of-two of 10ms (i.e. 20ms, 40ms, 80ms, etc.).

4.11 How is the slot frame configured?

- The Active slot in Contiki 6TiSCH implementation is scheduled at the slot Offset 0x00 and channel Offset 0x00 i.e. first time slot of the slot frame.
- The default time slot duration in Contiki 6TiSCH implementation is 10ms. Defined in **/rich/core/net/mac/tsch/tsch-conf.h**

```
#define TSCH_CONF_DEFAULT_TIMESLOT_LENGTH 10000
```

- The parameters that define the timing of the time slot are defined in **tsch-conf.h** as well.

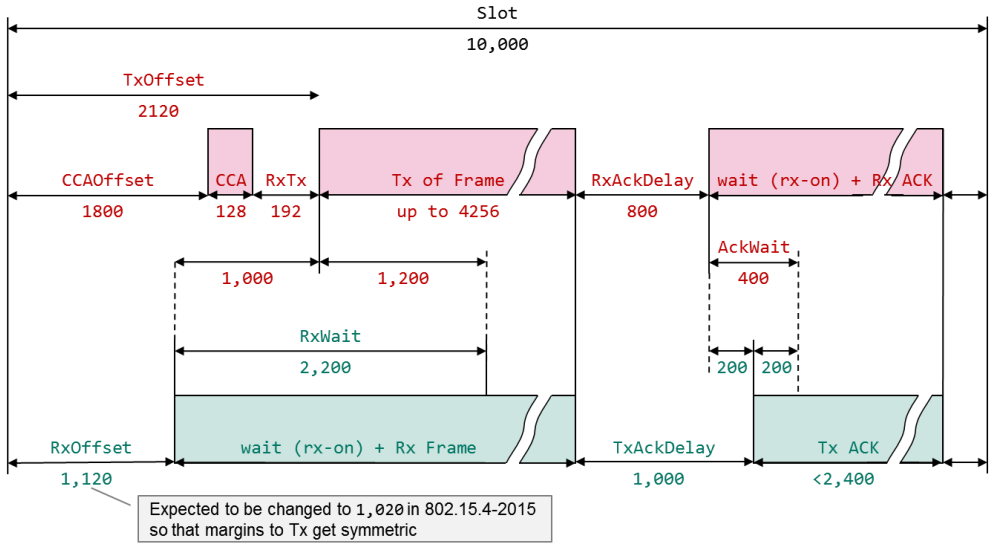


Fig 41. Minimal 6TiSCH time slot configuration

4.12 How to configure the TSCH Hopping Sequence?

The TSCH configuration is done in **/core/net/mac/tsch/tsch-conf.h**. A number of default hopping sequences are defined as convenience and these can be set as default hopping sequence to be used by the coordinator and for the scan lists for joining nodes.

Note, the default hopping sequence list is defined as:

```
{ 15, 25, 26, 20 }
```

4.13 How does 6TiSCH create networks?

The TSCH network is formed by the coordinator. The coordinator defines the essential parameters such as slotframe structure, hopping list and the slot timing parameters, and advertises these in Enhanced Beacon (EB) broadcast frames.

Non coordinator nodes listen for EBs and, once they have received one or more EBs use the information from the EB to synchronize to the TSCH network.

Once a node is synchronized to the network, it will use RPL to place itself in the routing tree. Once the node has a RPL rank, it too can start to broadcast EB frames, so nodes further away from the coordinator can also join the network.

This is depicted in a COOJA simulation in Fig 42 and Fig 43 where

Node 1 is the TSCH coordinator and RPL root,

Node 2 is a one hop neighbor,

Node 3 is a two hop neighbor.

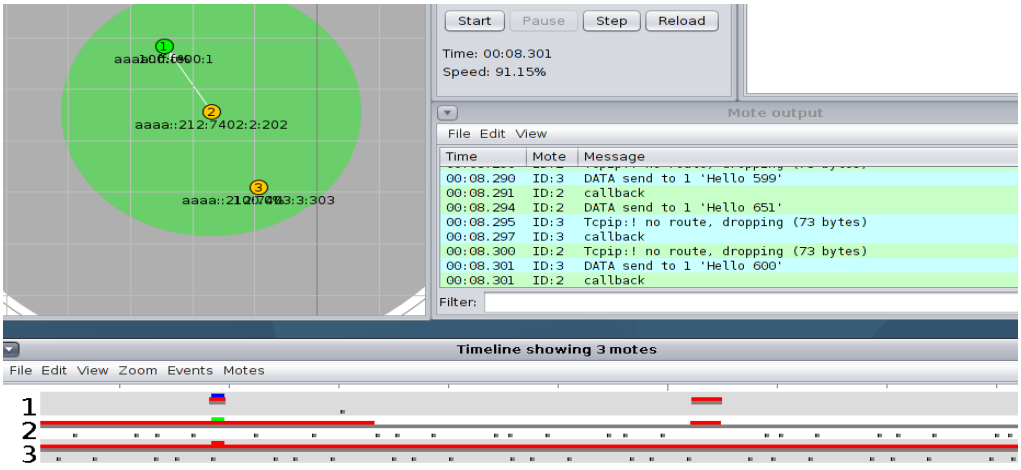


Fig 42. EB advertisement by TSCH coordinator

Node 1 (TSCH coordinator) advertises the EB in the active time slot of the slot frame and sends DIO for route formation.

Node 2 listens for EBs, to form a TSCH association. Once it receives the enhanced beacon from node 1, it obtains the slot frame, time slot and channel hopping information. It forms TSCH association with node 1 and selects the node 1 as its time source neighbor based on the join priority present in the Synchronization IE.

Node 2 broadcasts DIS packet to join the network and waits for the DIO response. Node 1 responds with a DIO message and is selected as parent by node 2.

Node 2 sends DAO response to form the upward route. Once node 2 has joined the RPL graph (and thus has a RPL rank), it starts advertising EB's periodically.

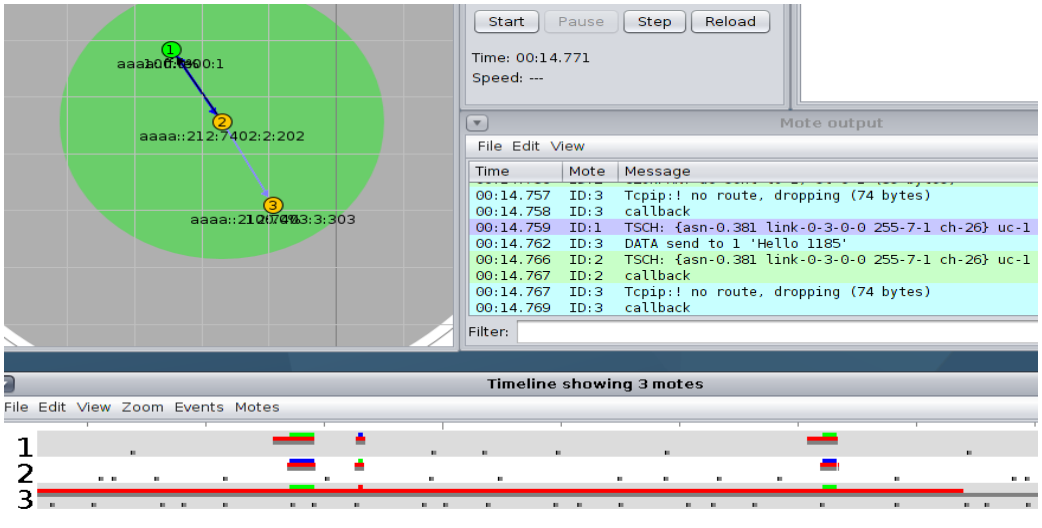


Fig 43. Node 3 joining network

Node 3 listens for EBs to join the network and receives the enhanced beacon from node 2 as shown in Fig 43.

Node 3 obtains the slot frame, timeslot, channel hopping sequence list, number of channels and joint priority information from the EB sent by node 2. Node 3 selects node 2 as its time source neighbor and forms a TSCH association. After TSCH association, node 3 broadcasts a DIS to join the network. Node 2 replies with a DIO. Node 3 selects node 2 as its parent and responds with DAO to form upward routes.

Once nodes join the network, they can exchange data packets.

4.14 How does TSCH handle shared links?

Shared slots mean that multiple links are assigned to the same time slot. The Contiki 6TiSCH minimal implementation offers shared links in the active time slots. Both EB's, RPL route formation messages and data are transmitted in the active time slot, if the other cells are unscheduled.

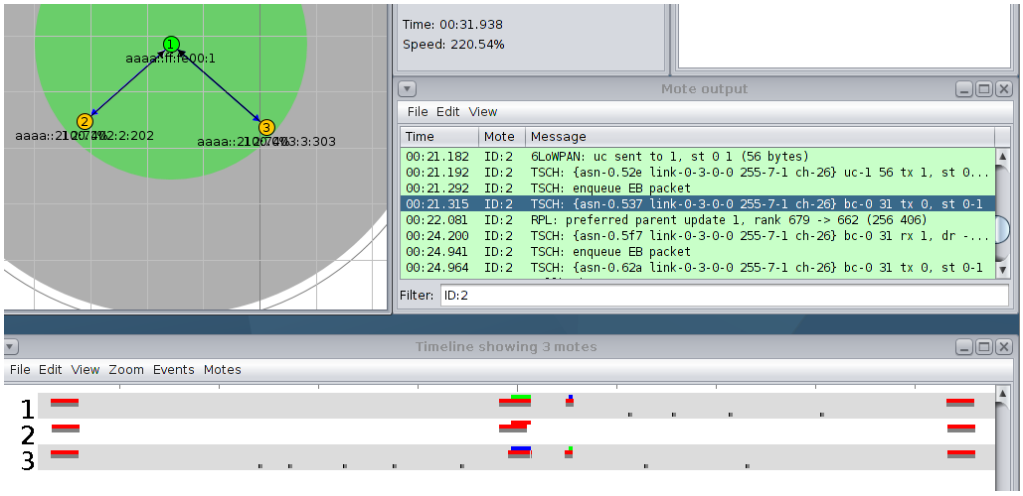


Fig 44. EB advertisement by node 2

Link 1-2 and Link 1-3 share the same active timeslot to transmit messages as shown in Fig 44. CCA is disabled for INTRA-PAN transmissions as nodes are tightly synchronized, both nodes 2 and 3 start transmitting in their active time slot, resulting in collision at node 1 and consequently a non-received ACK at the transmit side. A retransmission back-off TSCH-CSMA CA algorithm is implemented, that results in the retransmission of the packet on another scheduled frame/slot. The algorithm reduces the probability of further collisions.

4.15 How is synchronization between nodes done?

Initial synchronization is performed by the use of EB's. Synchronization is maintained by slotted communication with other devices in the PAN (Frame-based and ACK-Based synchronization)

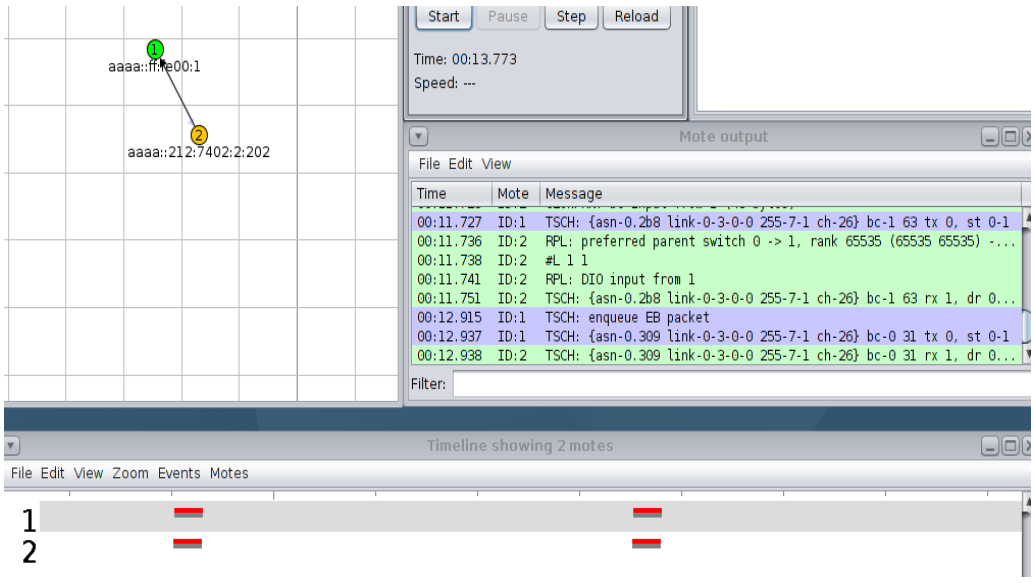


Fig 45. Node synchronization

This is depicted in a COOJA simulation in Fig 45 where

Node 1 is the TSCH coordinator and RPL root,

Node 2 is a one hop neighbor

Node 2 initially synchronizes with node 1 when it receives an EB from node 1 and makes node 1 its time source neighbor based on the join priority present in the Synchronization IE of EB. The synchronization Information Element (IE) in the EB frame contains the Absolute Slot Number (ASN) of the timeslot in which node 1 sent the EB.

Once node 2 forms a TSCH association with node 1, it continues its time synchronization with the help of data packets exchanged between the two nodes and the enhanced beacons periodically sent by the node 1 if the nodes do not communicate for a long time.

When node 2 receives a data packet from node 1, it identifies it as coming from its time source neighbor and tries to correct the drift in start of timeslot. (Frame-based Synchronization).

When node 1 (time source) receives a data packet from node 2, it sends an enhanced ACK packet with drift information in the time slot to node 2 (Ack-based Synchronization).

In addition, Contiki also provides a keep-alive mechanism to keep nodes in sync. EBs are efficient in the case where a single packet resynchronizes all children. Keep-alives are used if a child didn't get the last EBs, is drifting away, and needs explicit resync (the keep-alive is an acknowledged unicast, it will be resent until ACK'ed).

4.16 What is the scheduler interface in Contiki?

Contiki 6TiSCH implementation provides a CoAP based external scheduler interface that makes use of the 6Top MAC Layer interfaces to provide dynamic scheduling solutions (examples see 4.17 and 4.18).

4.17 How to add a slot frame

The slot frame can be added with the TSCH schedule interface **(core/net/mac/tsch/tsch-schedule.h)**:

```
struct tsch_slotframe *tsch_schedule_add_slotframe(uint16_t handle,
                                                   uint16_t size);
```

To add a slotframe, the caller passes a handle (`uint16_t handle`), and the number of time slots in the slotframe (`uint16_t size`).

The new created slotframe (`tsch_slotframe *)` holds the slot frame parameters and links associated to this slotframe. The slot frames are stored as a list. Returning a `NULL` indicates failed slot frame creation.

4.18 How to schedule a link

A link can be scheduled in a time slot of a slot frame by using the TSCH schedule interface **(core/net/mac/tsch/tsch-schedule.h)**:

```
struct tsch_link *
tsch_schedule_add_link(struct tsch_slotframe *slotframe,
                       uint8_t link_options,
                       enum link_type link_type,
                       const rimeaddr_t *address,
                       uint16_t timeslot,
                       uint16_t channel_offset);
```

The user parameters are:

`*tsch_slotframe`: The slotframe to which the link will be added (see ch.4.17).

`link_options`: Link Options field indicates whether the link is a TX link, an RX link or a S**hared** TX link and whether the device to which it is being linked is to be used for clock synchronization.

`link_type`: Indicates whether the link type is normal or advertising. Default is NORMAL (=0). ADVERTISING (=1), indicates the link may be used to send an enhanced beacon.

`*address`: address of the node connected to this link or the broadcast address.

`timeslot`: Timeslot in the slot frame used for this link.

`channel_offset`: Channel_offset used for this link

The new created link is stored in (`tsch_link *`). Returning a `NULL` indicates failed link creation.

4.19 Orchestra

Orchestra is an autonomous scheduling solution for TSCH, where nodes maintain their own schedule solely based on their local RPL state. There is no centralized scheduler nor negotiation with neighbors, i.e. no traffic overhead. The default Orchestra rules can be used out-of-box in any RPL network, reducing contention to a low level. Orchestra is described and evaluated in

[*Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH*] (http://www.simonduquennoy.net/papers/duquennoy15orchestra.pdf), ACM SenSys'15.

The example applications in examples/jn516x/tsch are configured to use Orchestra.

5. IoT Gateway Device as Border Router

In the environment described so far, the IPv6 communication outside the WPAN was realized by SLIP over a serial channel between a Linux machine and a JN516x node acting as border router. On the Linux side, tunslip6 is used to emulate a serial network interface over which to tunnel the IPv6 packets. On the JN516x side, the border router software routes all packets destined outside the WPAN over the SLIP link to the outside world and vice versa. For the serial link either a UART or a USB port with Communications Device Class capabilities to emulate an RS-232 port can be used.

Fig 46. NXP IoT Gateway Device

With the NXP IoT Gateway hardware, a border router[4] can be realized between a without the need for an external Linux PC, as it is included in the IOT Gateway box. This chapter describes the process of configuring the IoT Gateway box for this purpose.

Note, this chapter is only relevant for detailed understanding and building non-standard border routers.

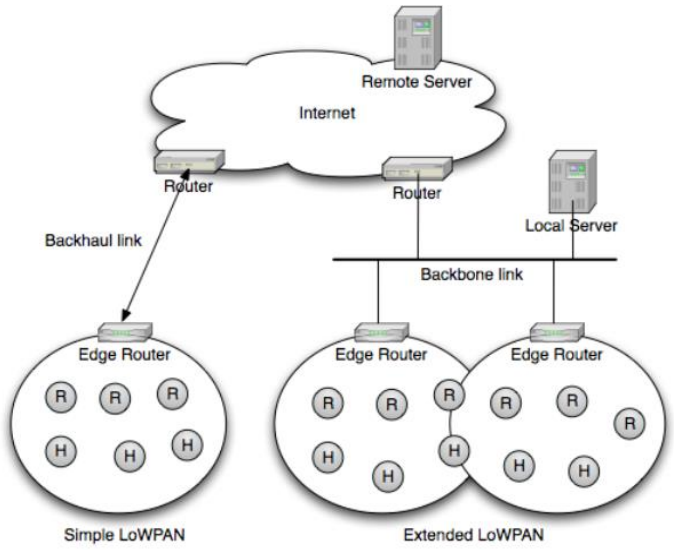
5.1 System architecture

Fig 47 shows an overview of the desired system. Several components are shown in this figure.

- Edge Router: In the 6TiSCH architecture, it is an LBR and also an IPv6 ND-efficiency-aware Router. It performs ND proxy operations between registered devices and classical ND devices that are located over the backbone.
- LLN: Low power and Lossy networks (LLNs) are typically composed of many embedded devices with limited power, memory, and processing resources

[4]Note that the border router functionality is mostly located on the JN5168 wireless microcontroller, but for convenience we also call the IoT Gateway box “border router”, as this is how it will be used and conceived.

interconnected by a variety of links, such as IEEE 802.15.4 or Low Power WiFi. There is a wide scope of application areas for LLNs, including industrial monitoring, building automation (HVAC, lighting, access control), connected home, healthcare, environmental monitoring, urban sensor networks, energy management, assets tracking and refrigeration.



Source: http://6lowpan.net/wp-content/uploads/2009/12/6lowpan-book-slides-full-20091206.pdf, also see http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470747994.html

Fig 47. System architecture

Fig 48 and Fig 49 show the hardware structure of a border router based on the NXP IoT Gateway hardware found in the JN5168 Evaluation Kit. The hardware is built around the ARM 9 based LPC3240 module and the JN5168. These microcontrollers are connected to each other using UARTS.

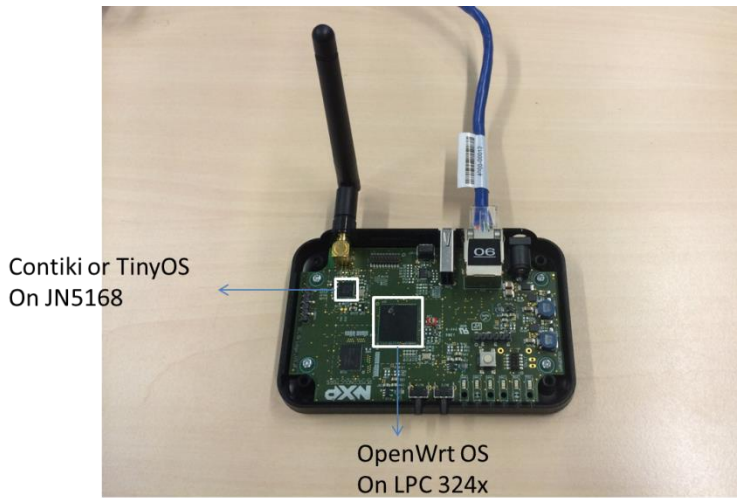


Fig 48. Border Router hardware used

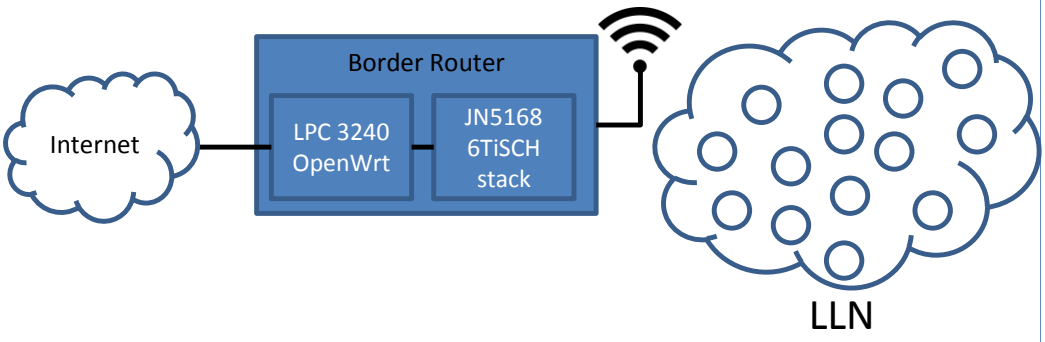


Fig 49. Border router in Context

The LAN nodes will use the UDP/IPv6 protocol to communicate with the border router's LPC3240. The border router uses the JN5168 to communicate with the WPAN devices. The IoT Gateway device uses SLIP over a UART port to transfer IPv6 packets between the LPC3240 (Linux) and JN516x microcontroller (Contiki).

The software layers of the desired system are shown in Fig 50.

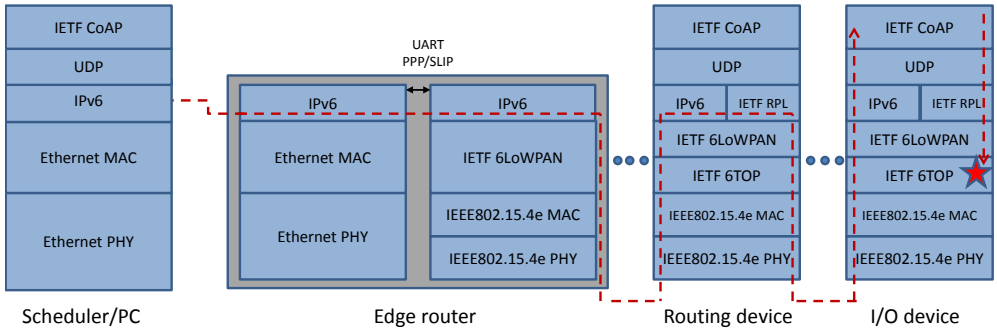


Fig 50. Network Layer view

In the border router, the host module (LPC3240) uses a serial connection via UART to communicate with the border router node (JN5168).

The NXP border router functionality can be added to any Linux system, using common system packages and a few NXP developed ones. The NXP Border-Router in the evaluation kit is based upon the OpenWrt Linux distribution. This distribution was chosen because it has support for many cheap commercial off the shelf WiFi routers. It is very configurable and has a wide variety of packages available for installation that provide a solution to just about any networking requirement.

5.2 Build and configure the IoT Gateway firmware

This installation is not for the faint of heart and has a number of phases:

- Installation of Ubuntu Linux on VirtualBox on Windows host. On this host PC, LPC32xx firmware will be developed under Ubuntu.
- Installation of IoT-Gateway build environment in the Ubuntu guest OS
- Building the OpenWrt firmware for the ARM 9 based LPC3240
- Flashing the OpenWrt firmware on the LPC3240
- Flashing the border router image in the JN5168
- Configuring and validating the IoT Gateway firmware

5.2.1 Installation of Ubuntu on VirtualBox

Note, this section describes the VirtualBox virtual machine, if you already have the VMware player with Instant Contiki (as described in 3.6) you can skip this section and go directly to section 5.2.2, Building OpenWrt for IoT Gateway.

For VirtualBox on Windows follow these steps:

- Download VirtualBox for Windows hosts from:

  https://www.virtualbox.org/wiki/downloads

Fig 51. VirtualBox download

- Go to http://www.ubuntu.com/download.
- Select **Ubuntu Desktop** and download the iso disc file of the latest version.
- Use the **latest** versions of both Ubuntu and VirtualBox. This example installation uses Ubuntu 14.04.2LTS and VirtualBox 4.3.24

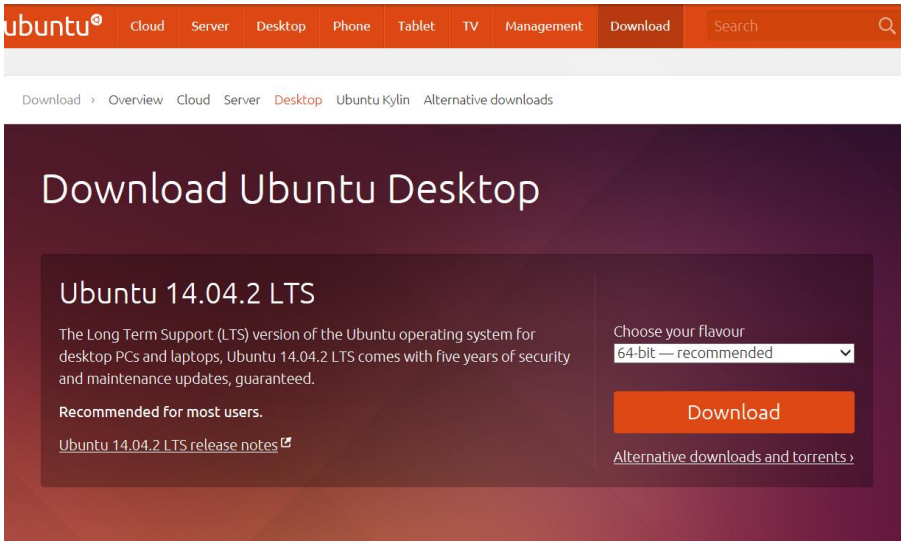


Fig 52. Ubuntu download in VirtualBox

- Open VirtualBox and select: **Machine->New**
- Name the Virtual Machine (**IoT Gateway** in this example)

- Enter OS settings and press **Next**

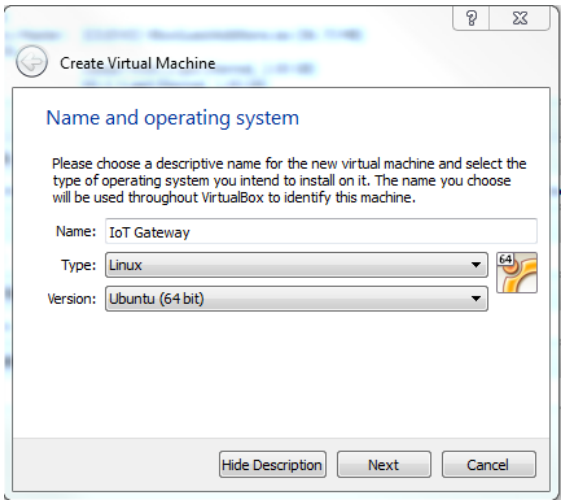


Fig 53. Installation of Ubuntu in VirtualBox

- Use the default Memory Size. Press **Next**

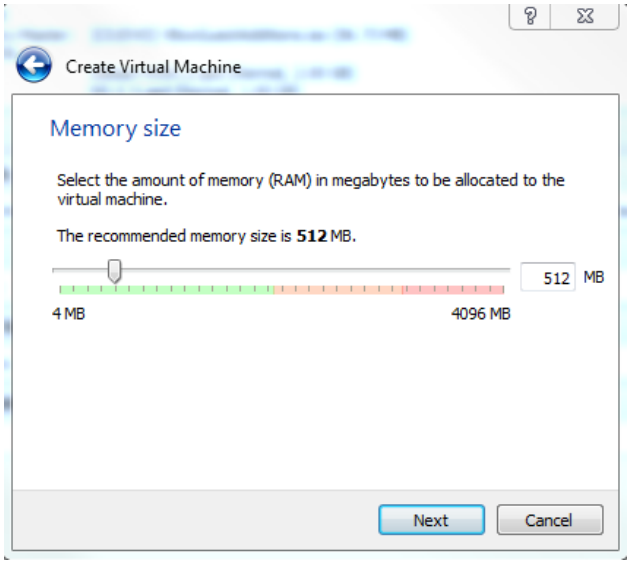


Fig 54. VirtualBox memory configuration

- Select **Create a virtual hard drive now**, then **Create**

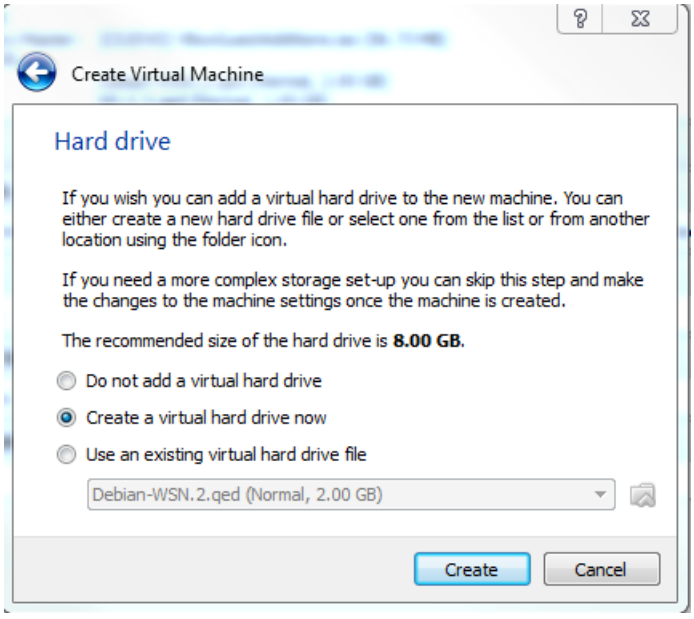


Fig 55. VirtualBox disk configuration

- Select **VDI**, then **Next**

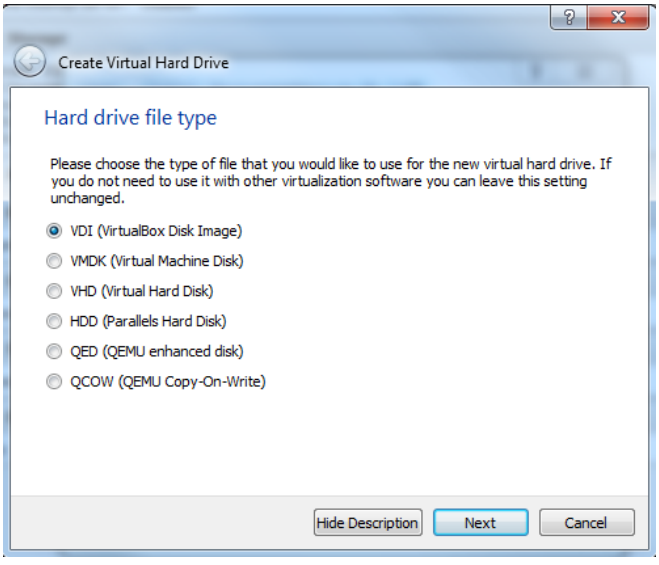


Fig 56. VirtualBox disk file type

- Select **Fixed size**, then **Next**

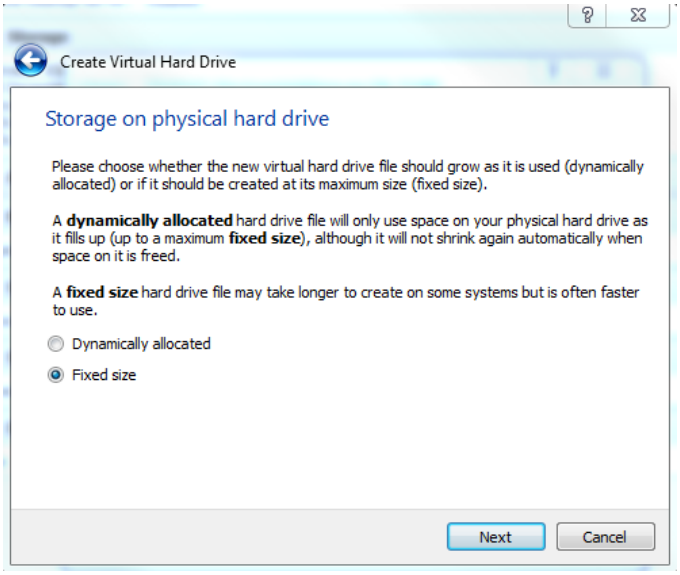


Fig 57. VirtualBox disk sizing

- Set drive size on **20G**. Then **Create**

  This avoids having to extend the drive size after the virtual machine is installed as it is quite a hassle (http://www.ehow.com/how_12146670_increase-virtualbox-hard-drive-size.html)

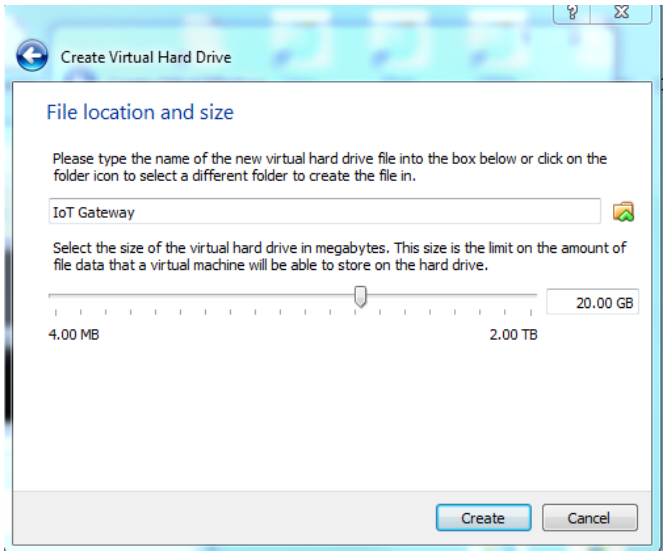


Fig 58. VirtualBox image name

Creating the drive may take a while.

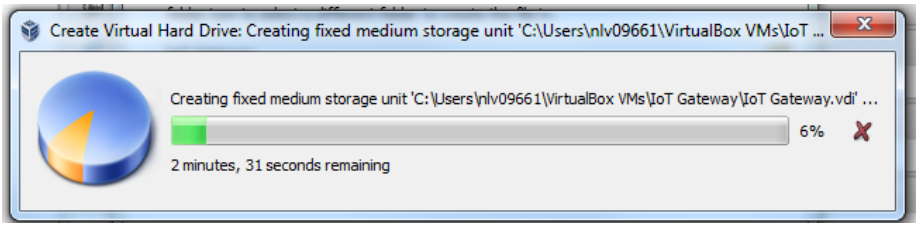


Fig 59. VirtualBox hard disk image file creation progress

- In the VirtualBox manager, the IoT Gateway Virtual Machine is now visible (among any other previously installed VM's).

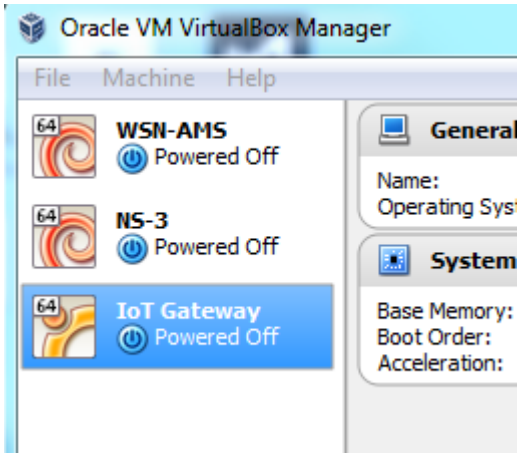


Fig 60. IoT Gateway Virtual Machine in VirtualBox Manager

- Right click on **IoT Gateway** and select: **Settings->Storage**

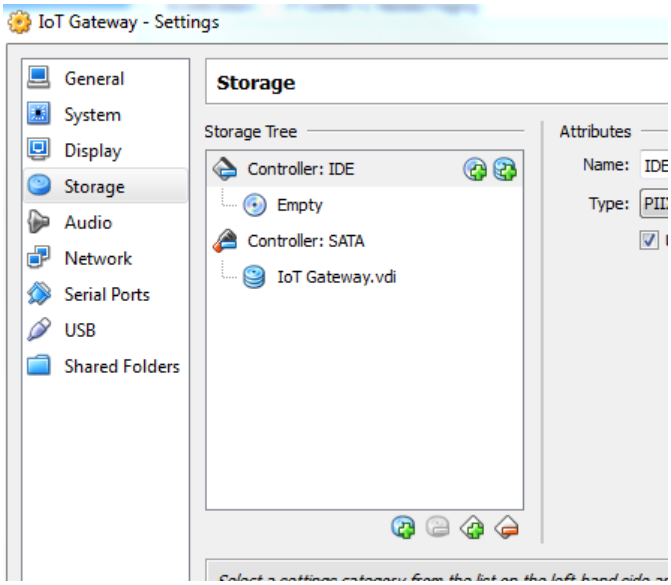


Fig 61. Prepare installation of Ubuntu on a Virtual Machine

- At **Controller: IDE**: Click **Add CD/DVD Device** (use the left side “+” icon on the bottom) and select **Choose disk**

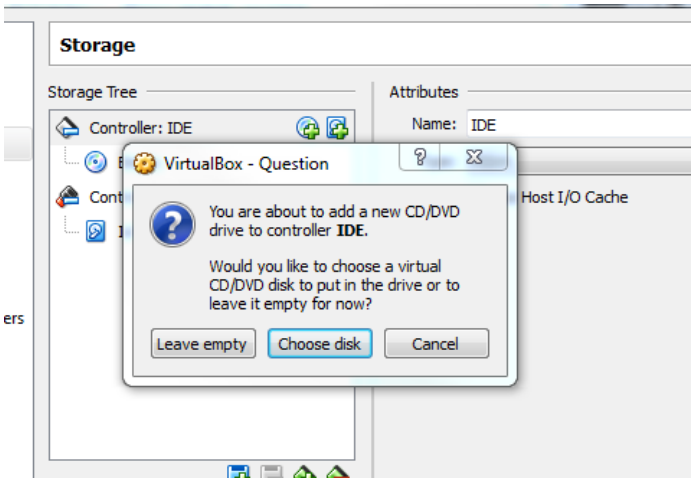


Fig 62. Select the Ubuntu boot disk in the virtual DVD Drive

- Select the downloaded Ubuntu .iso file.
- Right click on **empty drive** and select **remove attachment**.

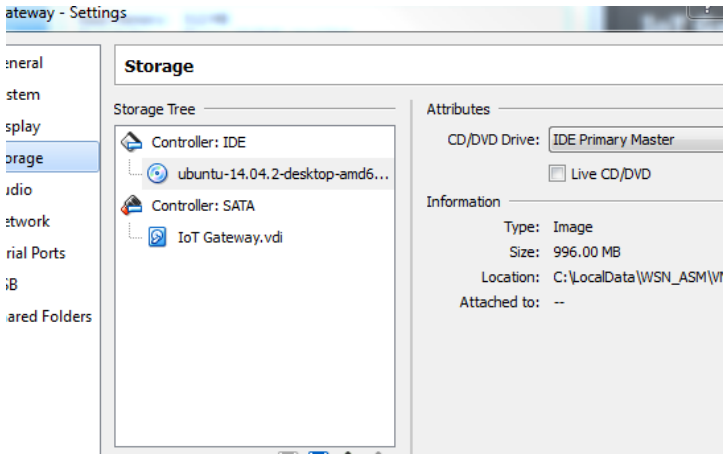


Fig 63. Virtual Machine configuration with Ubuntu boot DVD image file

- Back in VirtualBox manager, double click **IoT Gateway**. Ignore any warnings. The installation of Ubuntu will now take place in the new virtual machine image. Select your preferred language and press **Install Ubuntu**.

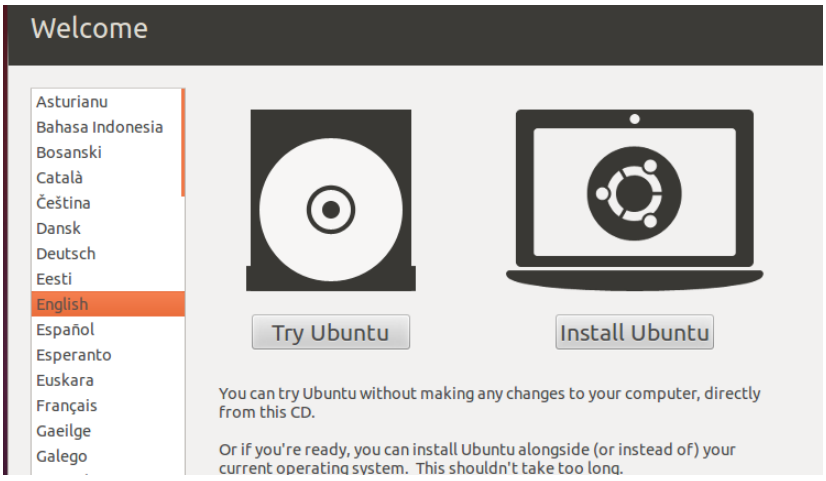


Fig 64. Starting installation of Ubuntu in the Virtual Machine

- Select **Download updates while installing** and **Continue**

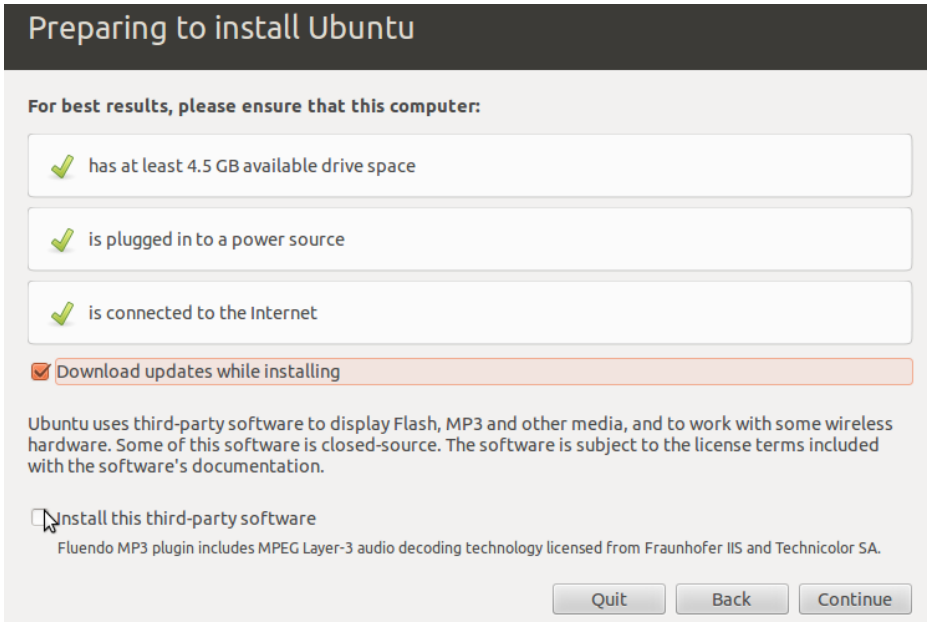


Fig 65. Ubuntu installation progress

- Select **Erase disk and install Ubuntu**
- then **Continue-> Install Now**

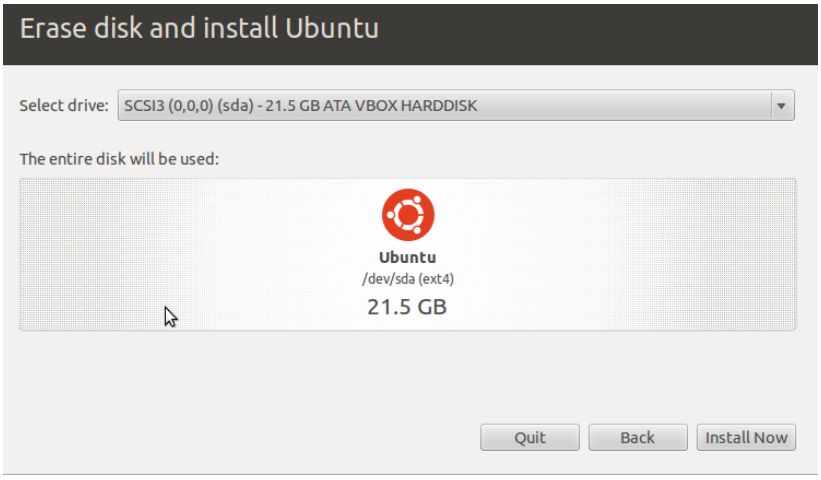


Fig 66. Installation of Ubuntu selection of target hard drive image file

- During installation you will be asked for location, keyboard layout
  Enter **your data** and press **Continue**

Fig 67. Ubuntu credential entry

- Installation continues with a progress bar showing progress. This will take a while.
- After installation, restart the virtual computer.

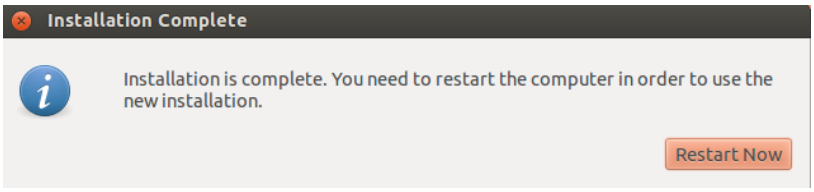


Fig 68. Installation of Ubuntu on VirtualBox complete

- Now the system may hang-up! Apparently this is a known issue: https://forums.virtualbox.org/viewtopic.php?f=6&t=52381. The following sequence will resolve this issue.

  Close the Virtual Machine and select **Power off the machine -> OK**

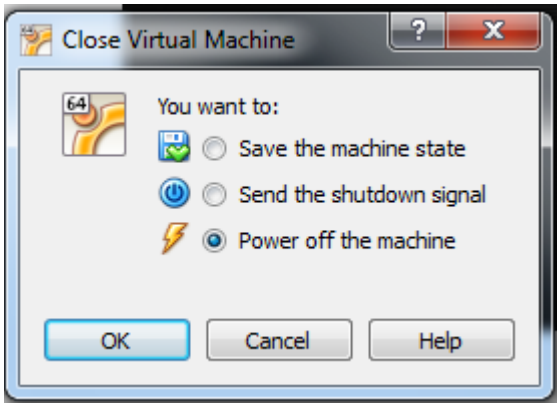


Fig 69. Powering off the Ubuntu virtual machine in VirtualBox

- Right click **IoT Gateway**, then select **Settings->Storage**
  Check if the Ubuntu iso drive is gone. If not, **r**ight click on Ubuntu entry and select **Remove Attachment**

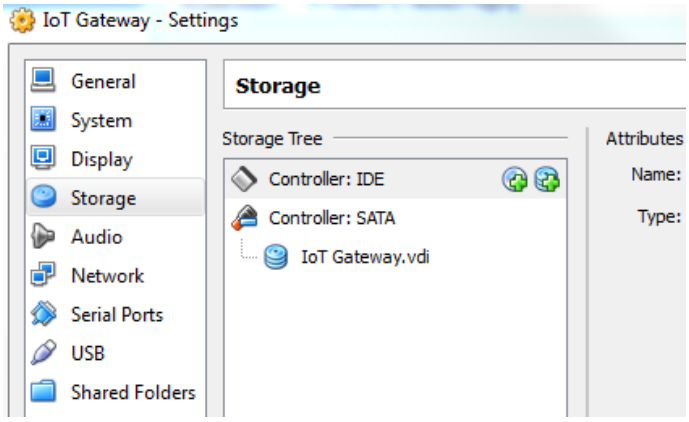


Fig 70. Ubuntu boot CD image removed from virtual DVD drive

- For better UI and screen size options execute the following additions:
  - At **Controller: IDE**: Click **Add CD/DVD Device** and select **Choose disk**.
  - Add drive: **C:\Program files\Oracle\VirtualBox\VBoxGuestAdditions.iso**.
  - Check attributes that CD/DVD drive is: **IDE Secondary Master**. If not modify it.
  - When finished, press **OK**.

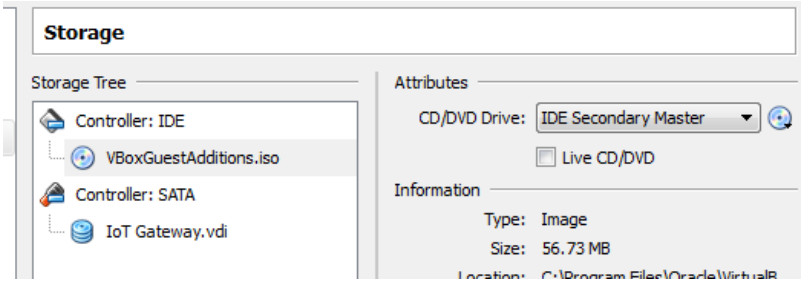


Fig 71. Installation of VBoxGuestAdditions in Ubuntu

- In the VirtualBox Manager start the **IoT Gateway.**

  After entering the password, the Ubuntu desktop will appear. Open a terminal (CTRL_ALT_T).
- The VBoxGuestAdditions must now be installed in the guest OS (Ubuntu).
- To fully update the guest system enter:
  - **sudo apt-get update**
  - **sudo apt-get upgrade**
- Then install DKMS using: **sudo apt-get install dkms**

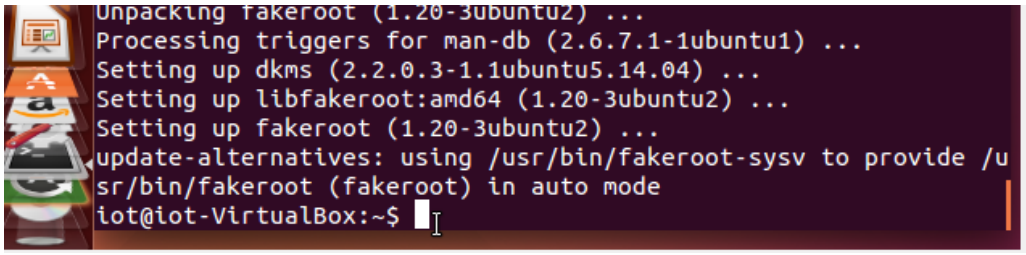


Fig 72. Getting Ubuntu packages for VBoxGuestAdditions

- Close the VM and select Power off the machine -> OK
- Restart the IoT Gateway
- Open a terminal and go to the VBOXADDITIONS drive.

  With the current installations, this is:

  cd /media/iot/VBOXADDITIONS_4.3.24_ 98716

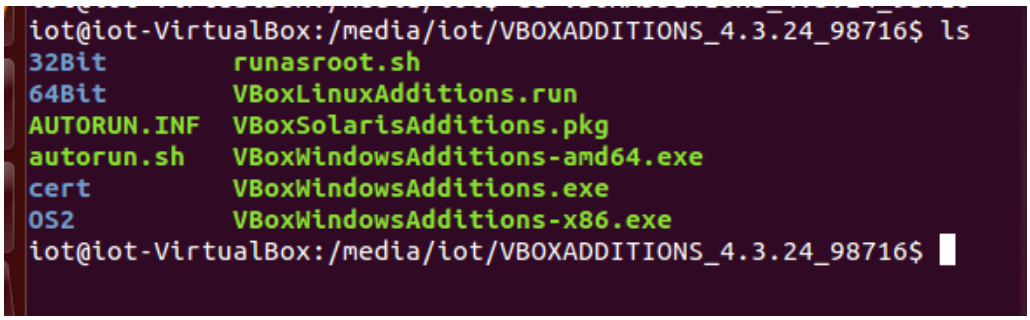


Fig 73. VBoxGuestAdditions files

- Then execute:

  sudo sh ./VBoxLinuxAdditions.run

  This will add the additions to the Linux kernel.

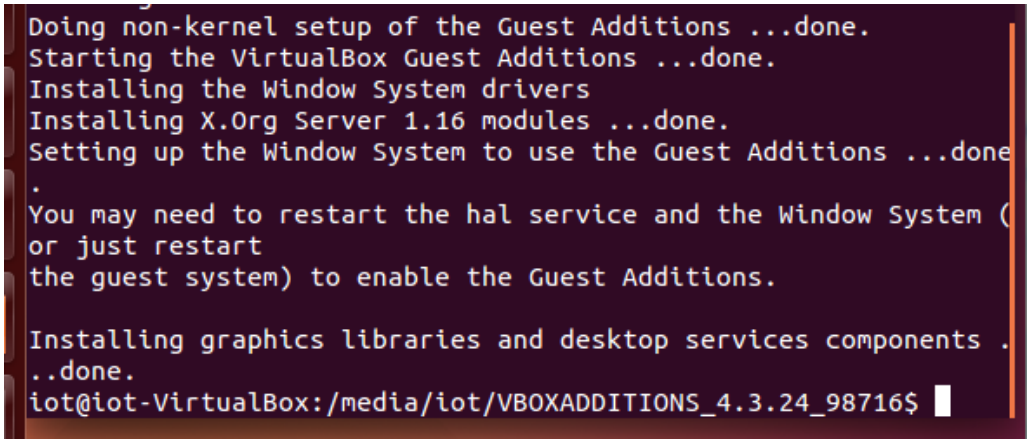


Fig 74. VBoxGuestAdditions installation execution

- It is useful to have a shared drive between the Ubuntu VM and the Windows Host. This is one of the features of the installed VBoxGuestAdditions. We will make a share on the Host named **UbuntuVMShare**.
- On the Windows host, create a directory UbuntuVMShare.
- Right click on the directory, select **Properties->Sharing->Share**

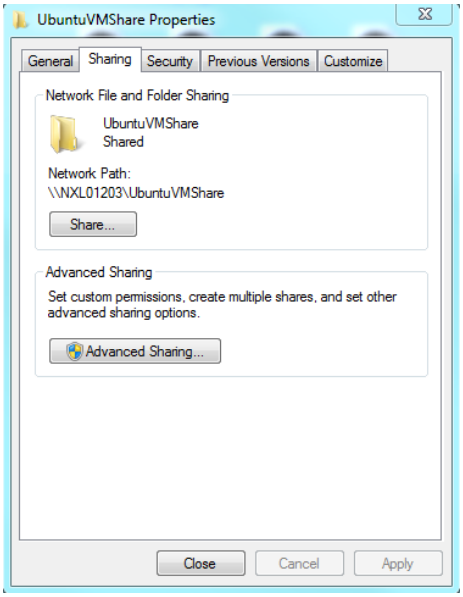


Fig 75. Creating a shared directory in the Windows file system

- In VirtualBox Manager, select the **IoT Gateway VM**
- right click **Settings->Shared Folders**

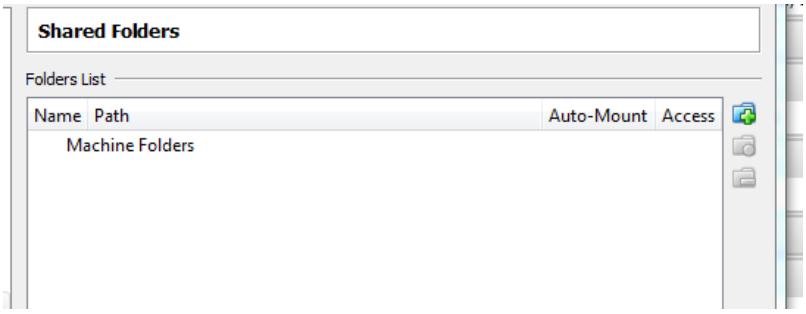


Fig 76. Shared Folders list in the VirtualBox Manager

- Select **Machine Folders**
- click on the **Add folder** icon (green +).
- Set the path to the shared folder.

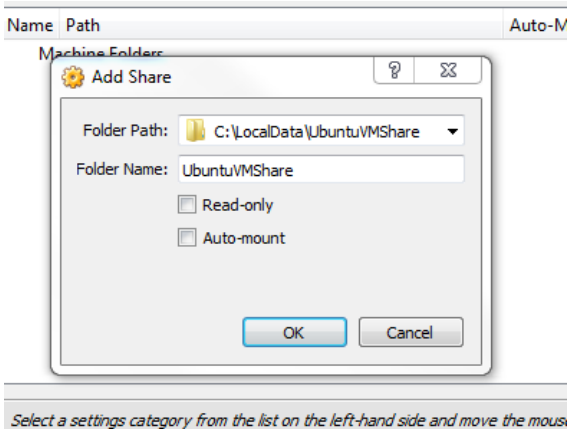


Fig 77. Linking to the shared directory in Ubuntu

- Press **OK** to go back into the VirtualBox Manager.

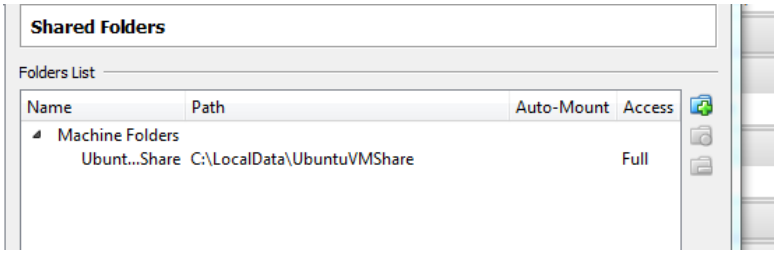


Fig 78. Shared Folders list in the VirtualBox Manager with linked directory

- Switch off VirtualBox again and restart.
- Double click on **IoT Gateway**. Ubuntu will boot and request for password.
- Log in into Ubuntu and open a terminal.
- Create a directory where you want your share to be visible.
  In this example: **mkdir Share**
- Mount the shared folder with: **sudo mount –t vboxsf UbuntuVMShare ~/Share**
- **ls** The share is visible with the green background:

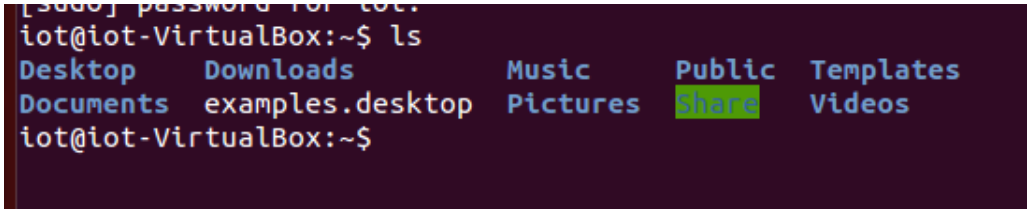


Fig 79. Shared folder in Ubuntu directory listing

- Go to the **Settings** menu of **IoT Gateway** VM in **VirtualBox**.
- Select: **Network**
- In **Attached to:** Select **Bridged Adapter**
- **The Name** of the adapter will depend on your host PC setup.
- Finalize with **OK.**

  With this setting, the VM will have access to the external LAN.

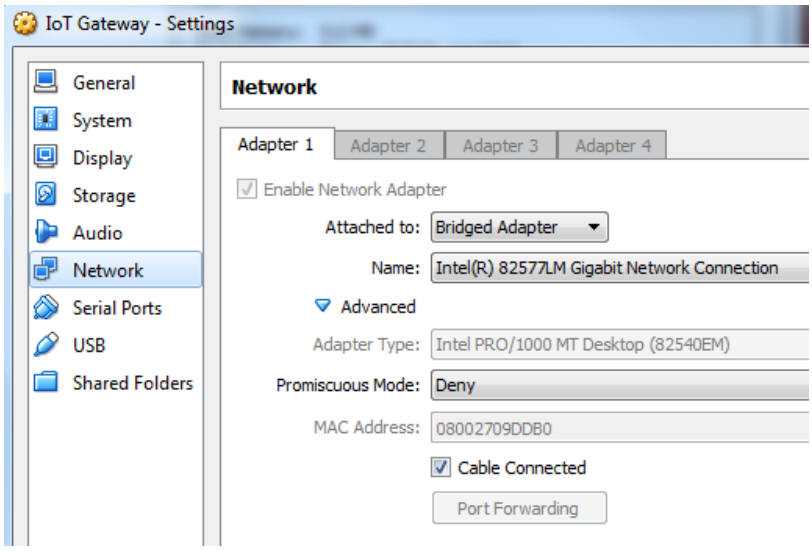


Fig 80. Selecting Network Adaptor mode in the Virtual Machine

5.2.2 Building OpenWrt for IoT Gateway

This section describes how to build the Linux image for the LPC3240 microcontroller in the IoT Gateway box. For this we use the OpenWrt distribution as it has great support for networking.

The build environment is a variation on the one described in Application note JN-AN-1194 (http://www.nxp.com/documents/application_note/JN-AN-1194.zip), which describes a ZigBee IoT Gateway. For the Border Router in the Contiki environment we use the same OpenWrt distribution but it is configured differently.

- Open **IoT Gateway** VM in **VirtualBox.**
- Open Firefox and enter the URL:
  http://www.nxp.com/documents/application_note/JN-AN-1194.zip
  File saved in: **~/Downloads**
  Most of the following installation instructions are derived from the JN-AN-1194-ZigBee-IoT-Gateway-Host_1v1.pdf in the downloaded zip file.
- Create a JN-AN-1194 directory, copy JN-AN-1194.zip to this directory from **~/Downloads** and extract files from zip file with: **unzip JN-AN-1194.zip**
- The following packages need to be installed:

  sudo apt-get install subversion

  sudo apt-get install zlib1g-dev

  sudo apt-get install libncurses5-dev

  sudo apt-get install build-essential

  sudo apt-get install git-core

  sudo apt-get install gawk

  sudo apt-get install libxml-libxml-perl

  sudo apt-get install gettext
- Create a work area to build OpenWrt (in this example: **wa_openWRT**)
- Copy tarballs of ~/JN-AN-1194/…/Host to work area:

  cp ~/JN-AN-1194/ZigbeeNodeControlBridge/Source/Host/*.tar.gz ~/wa_openWRT
- **cd wa_openWRT**

  The directory should look as follows:

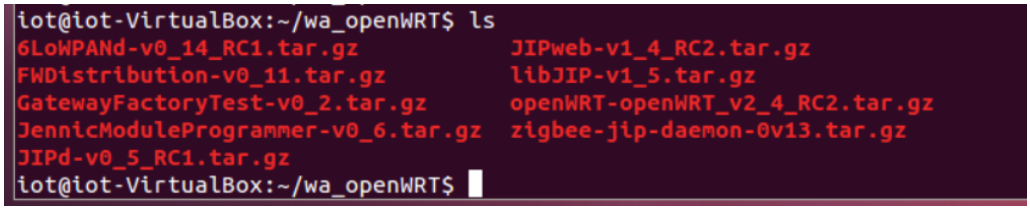


Fig 81. Building OpenWrt for the IoT Gateway hardware

- Extract OpenWrt with:

  tar xvzf openWRT-openWRT_v2_4_RC2.tar.gz
- The following is an addition to documentation in JN-AN-1194. The original make file assumes the presence of a repository svn.luci.subsignal.org, which is not available anymore. The data will be manually loaded from the new repository.

  Download **luci** from the new repository, tar/zip the downloaded directory and store it in the download area of openWrt:

$ svn co https://github.com/openwrt/luci/branches/luci-0.11 luci
tar cvf luci-0.11+svn9964.tar luci/
gzip luci-0.11+svn9964.tar

The work area will look as follows:

```
iot@iot-VirtualBox:~/wa_openWRT$ ls
6LoWPANd-v0_14_RC1.tar.gz           libJIP-v1_5.tar.gz
FWDistribution-v0_11.tar.gz         luci
GatewayFactoryTest-v0_2.tar.gz      luci-0.11+svn9964.tar.gz
JennicModuleProgrammer-v0_6.tar.gz  openWRT
JIPd-v0_5_RC1.tar.gz                openWRT-openWRT_v2_4_RC2.tar.gz
JIPweb-v1_4_RC2.tar.gz              zigbee-jip-daemon
libJIP                              zigbee-jip-daemon-0v13.tar.gz
iot@iot-VirtualBox:~/wa_openWRT$
```

Fig 82. OpenWRT with additions extracted

- Edit **openWRT/Common.mk** and comment the following line out:

  cd backfire/feeds/luci && ….

```
download_feeds:
        echo "Checking out feeds"
        cd backfire && ./scripts/feeds update

        # Now set to our known revision number
        echo "Getting known good packages dir"
        cd backfire/feeds/packages && svn up -r $(REVISION_OPENWRT_FEEDS)
#       cd backfire/feeds/luci && svn up -r $(REVISION_LUCI)
        cd backfire && ./scripts/feeds update -i
        touch download_feeds
```

Fig 83. Fix for luci

- Check out openWrt and create build environment by entering:

  cd openWRT/targets/lpc32xx

  make

  This is revision 35400.

  Ignore the error message during the make when trying to access the svn.luci.subsignal.org repository. We just did that manually from the correct repo.

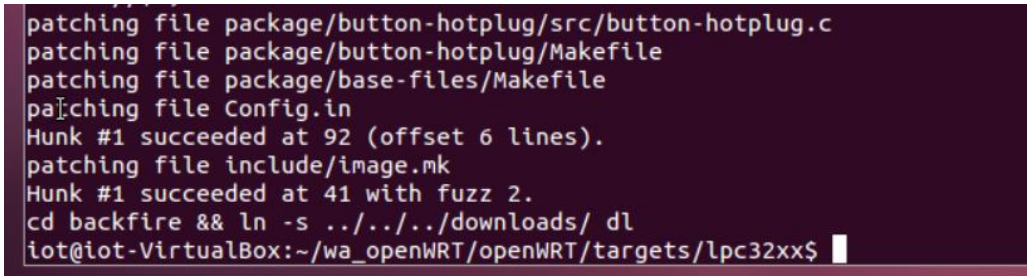


Fig 84. Building OpenWrt for IoT Gateway – make for lpc32xx output

- Enter: **cd backfire**

  Check if directory or link **dl** (=download) exists. If not, create directory **dl.**

  In this directory all downloaded tar balls will be stored; also tarballs downloaded during the make process later on.

- Copy tarballs to /**dl** or to the directory referenced by the link

  cp ../../../../*.tar.gz <Destination directory>

- A JN516x bin file needs to be loaded in the openWrt build environment. This bin file is used to program the JN516x on first boot up of a virgin gateway from the factory. Since the JN516x can always be reprogrammed afterwards, this could be any bin file. For Contiki, it makes sense to use the rpl-border-router firmware **border-router.jn516x.bin** that was generated in ch.3.8.1.2.
  - At the Windows Host, store this file in the UbuntuVMShare directory. At the Ubuntu side, the file is available in the **~/Share** directory.
  - Then do the following copy action from **openWRT/targets/lpc32xx/backfire**:

    cp ~/Share/border-router.jn516x.bin

    target/linux/lpc32xx/base-files/usr/share/JennicModuleProgrammer

- The **hotplug2** package has a modified URL (see http://stackoverflow.com/questions/25246963/build-error-openwrt-hotplug2).

  In **openWRT/targets/lpc32xx/backfire/package/hotplug2/Makefile** replace

  PKG_REV:=201
  PKG_SOURCE_URL:=http://svn.nomi.cz/svn/isteve/hotplug2

  with

  PKG_REV:=4
  PKG_SOURCE_URL:=http://hotplug2.googlecode.com/svn/trunk

```
include $(TOPDIR)/rules.mk

PKG_NAME:=hotplug2
#PKG_REV:=201
PKG_REV:=4
PKG_VERSION:=$(PKG_REV)
PKG_RELEASE:=4

PKG_SOURCE_PROTO:=svn
PKG_SOURCE_VERSION:=$(PKG_REV)
PKG_SOURCE_SUBDIR:=hotplug2-$(PKG_VERSION)
#PKG_SOURCE_URL:=http://svn.nomi.cz/svn/isteve/hotplug2
PKG_SOURCE_URL:=http://hotplug2.googlecode.com/svn/trunk
PKG_SOURCE:=$(PKG_SOURCE_SUBDIR).tar.gz
#PKG_SOURCE_URL:=http://isteve.bofh.cz/~isteve/hotplug2
```

Fig 85. Hotplug fix

- The feeds for the luci package needs to be updated because of the changed repository.

  In **feeds.conf.default**, replace

  src-svn luci http://svn.luci.subsignal.org/luci/tags/0.11.1/contrib/package

  with

  src-svn luci https://github.com/openwrt/luci/branches/luci-0.11

```
src-svn packages svn://svn.openwrt.org/openwrt/branches/packages_12.09
src-svn xwrt http://x-wrt.googlecode.com/svn/trunk/package
# Commmented out
#src-svn luci http://svn.luci.subsignal.org/luci/tags/0.11.1/contrib/package
# Modified for IoT Gateway
src-svn luci https://github.com/openwrt/luci/branches/luci-0.11
#src-svn phone svn://svn.openwrt.org/openwrt/feeds/phone
```

Fig 86. Modified luci feeds.conf.default

- Execute the following commands to install additional packages for menu configuration:

  ./scripts/feeds update

  ./scripts/feeds install tunslip6

  ./scripts/feeds install miniupnpc

  ./scripts/feeds install –a –p luci

- The OpenWrt configuration may now take place:

  Enter: **make menuconfig**

- The OpenWrt configuration menu is shown:

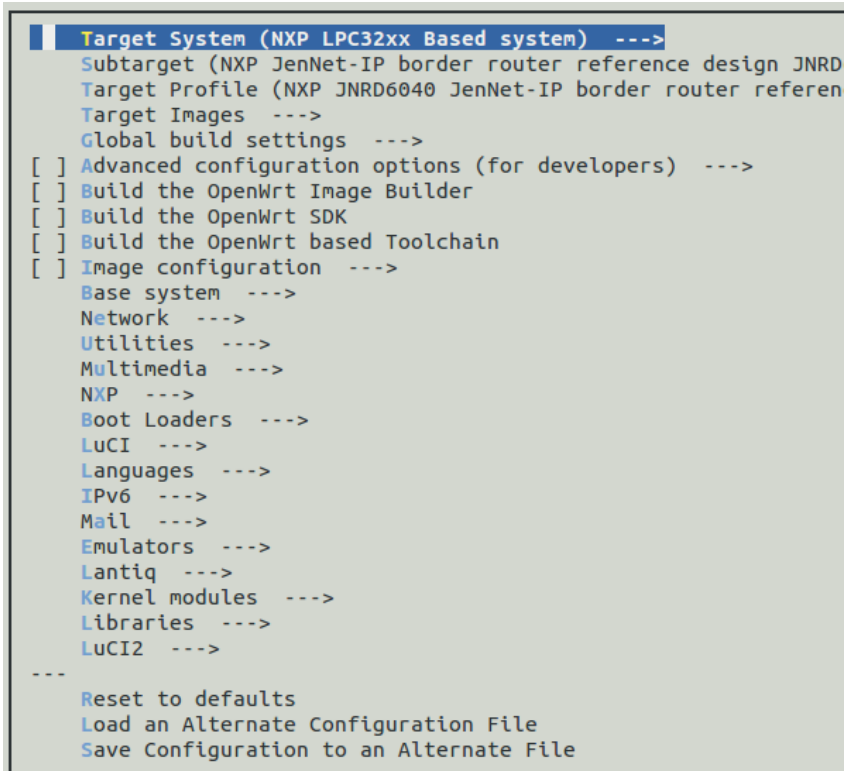


Fig 87. OpenWrt configuration menu

- Verify that the configuration items shown in Fig 87 are available, not necessarily in the same order. Missing items means that not all feeds are available.
- The following table shows the setting of the parameters that differ from the initial value. The new value indicates that the configuration item is included (**[Y]**) or not included (**[N]**).

Configuration item	Value
Network->Firewall->miniupnpc	[Y]
Network->Web Servers/Proxies->uhttpd-mod-lua	[N]
Network->ppp	[Y]
Network->ppp->ppp-mod-pppoe	[N]
Network->ppp->ppp-mod-pptp	[Y]
Libraries->libnfnetlink	[Y]
IPv6->6in4	[N]

Configuration item	Value
IPv6->6rd	[N]
IPv6->6to4	[N]
IPv6->tunslip6	[Y]
LuCI->2.Modules->luci-mod-admin-full	[Y]
LuCI->3. Applications->luci-app-ddns	[Y]
LuCI->3. Applications->luci-app-ntpc	[Y]
LuCI->3. Applications->luci-app-radvd	[Y]
LuCI->6. Protocols->luci-proto-ppp	[Y]
LuCI->7. Server Interfaces->luci-sgi-uhttpd	[Y]
Kernel Modules->Network Support->kmod-sit	[N]
NXP->Cloud Support->JIP-Cloud-Nabto	[N]
NXP->6LoWPAN	[N]

- When finished, **Exit** and **save the** new OpenWrt configuration.
- Build OpenWrt from **openWRT/targets/lpc32xx/backfire** by entering **make.**

  For more verbosity enter: **make V=s.**

  This build will take a while!
- The tunslip6.c file in the used OpenWrt distribution is not up to date. We have to replace this with a good version. Replacement can only be done after an initial make has been done; i.e. the build environment must be available.
- On the Ubuntu VM: Make a back-up of the current “old” tunslip6 that is in the build directory:

  cp build_dir/target-arm_uClibc-0.9.33.2_eabi/contiki-2.5/tools/tunslip6.c ./tunslip6.old.c
- On the Windows Host PC: The new tunslip6.c is located in the Contiki repository in the directory **tools**.
  - Copy **tunslip6.c** from this tools directory to the share **UbuntuVMShare** on the Windows host.
  - On Ubuntu VM: copy new tunslip6.c from **~/Share** to build directory:

    cp ~/Share/tunslip6.c build_dir/target-arm_uClibc-0.9.33.2_eabi/contiki-2.5/tools/tunslip6.c

- On Ubuntu VM: Compile the new tunslip6.c with the following 2 commands:
  touch build_dir/target-arm_uClibc-0.9.33.2_eabi/contiki-2.5/tools/tunslip6.c
  make –C feeds/packages/ipv6/tunslip6 TOPDIR=$PWD compile
- Verify timestamps build data of compiled tunslip6 object in
  build_dir/target-arm_uClibc-0.9.33.2_eabi/contiki-2.5/tools

```
iot@iot-VirtualBox:~/wa_openWRT/openWRT/targets/lpc32xx/backfire/build_dir/targe
t-arm_uClibc-0.9.33.2_eabi/contiki-2.5/tools$ ls -l tunslip6*
-rwxrwxr-x 1 iot iot 22876 mrt 30 15:25 tunslip6
-rw-r--r-- 1 iot iot 33691 mrt 30 15:24 tunslip6.c
iot@iot-VirtualBox:~/wa_openWRT/openWRT/targets/lpc32xx/backfire/build_dir/targe
t-arm_uClibc-0.9.33.2_eabi/contiki-2.5/tools$
```

Fig 88. Building tunslip6

- Execute a new make for the openWrt image: **make V=s**.
- Finally, when this **make** ends, the output binaries for the IoT host are found in directory:
  openWRT/targets/lpc32xx/backfire/bin/lpc32xx

```
iot@iot-VirtualBox:~/wa_openWRT/openWRT/targets/lpc32xx/backfire/bin/lpc32xx$ ls -l
total 41696
-rw-r--r-- 1 iot iot  5509685 mrt 13 14:14 kernel-debug.tar.bz2
-rw-r--r-- 1 iot iot      404 mrt 13 14:14 md5sums
-rw-r--r-- 1 iot iot 14680068 mrt 13 14:14 openwrt-lpc32xx-JNRD6040--jffs2-sysupgrade.bin
-rw-r--r-- 1 iot iot 10616832 mrt 13 14:14 openwrt-lpc32xx-JNRD6040-root.jffs2-128k
-rw-r--r-- 1 iot iot 10616832 mrt 13 14:14 openwrt-lpc32xx-JNRD6040-root.jffs2-64k
-rw-r--r-- 1 iot iot  1073472 mrt 13 14:14 openwrt-lpc32xx-JNRD6040-uImage
drwxr-xr-x 2 iot iot    12288 mrt 13 14:14 packages
-rwxr-xr-x 1 iot iot   170584 mrt 13 14:14 u-boot.lpc32xx.bin
iot@iot-VirtualBox:~/wa_openWRT/openWRT/targets/lpc32xx/backfire/bin/lpc32xx$
```

Fig 89. OpenWRT for LPC32xx output binaries

5.2.3 Flashing OpenWrt image in the IoT Gateway

Several options are available for flashing the image in the IoT host LPC3240. The chosen option will mostly depend on what kind of image(s) currently are flashed in the IoT Gateway.

- If an operational webserver is available in the current image, flashing a new image can be done via the web interface. This requires only an IP connection to the IoT Gateway (see 5.2.3.1) .This is the preferred way to flash new firmware.
- If an OpenWrt image is available, but for some reason the web interface is not functioning, then the flashing can take place via the serial interface of the IoT Gateway. This needs opening the IoT box (see 5.2.3.2).
- If no OpenWrt image is available, but there is a working uboot available, then flashing may be done by means of USB stick (see 5.2.3.3).

- If uboot is not functional, then uboot needs to be programmed first (see 5.2.3.4). When uboot is programmed, the OpenWrt images have to be programmed via USB (see 5.2.3.3). This mode assumes that at least the primary boot loader on the LPC3240 is operational.
- Additional information of flashing Linux related firmware on LPC3240 controller can be found in http://www.lpclinux.com/LPC32xx/WebHome.

Note: after a new OpenWrt image has been flashed, SSH communication with a host that has communicated before with the IoT Gateway may fail, due to changed keys. This issue is addressed in ch.5.2.3.2.

5.2.3.1 Flashing OpenWrt using the webserver

The current OpenWrt image needs to be replaced by the new version that has just been build. The following descriptions are derived from http://wiki.openwrt.org/doc/howto/generic.sysupgrade.

It will take care of replacing both the Linux kernel and the root file system. Note that packages and configurations of the previous installation will be lost.

This way of programming assumes that the system already properly boots and has a previous OpenWrt version with operational webserver on board.

- Connect the Gateway to a LAN network.

  On the **IoT Gateway VM**, open the Firefox web browser and login to the embedded webserver of the IoT Gateway by entering its IP address. By default 192.168.1.1 assumed, but if there is no response on this address it is most likely different. See ch.5.2.3.3.

  A login screen is shown. Log in with the following data:

  Username: **root**
  Password: **snap**

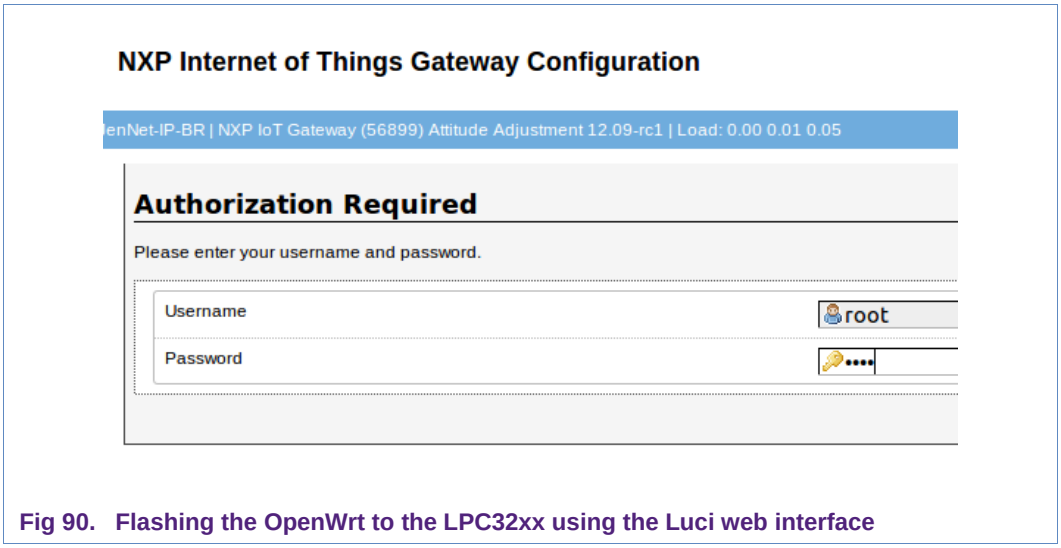


Fig 90. Flashing the OpenWrt to the LPC32xx using the Luci web interface

- Select: **System->Backup/Flash Firmware**

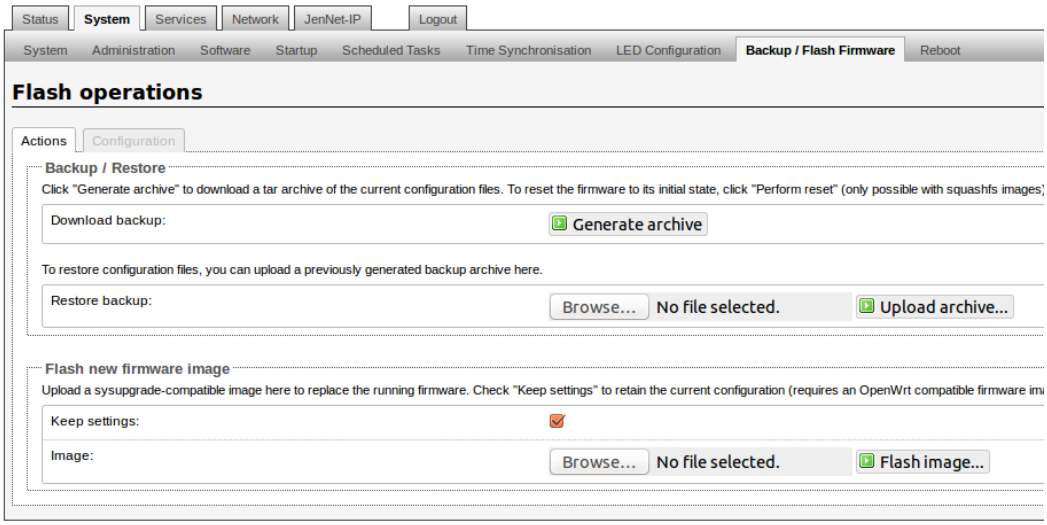


Fig 91. Luci flash tab

- If needed, make a back-up of the current configuration by pressing **Generate Archive.**

  The configuration items to be backed up can be set after pressing the tab **Configuration.**

- In **Flash new firmware image**, select **Browse…** to select the image to be programmed. For this complete system upgrade, select the **…sysupgrade.bin** file from Fig 92. This contains both the kernel and the root file system. After selection press **Open**.

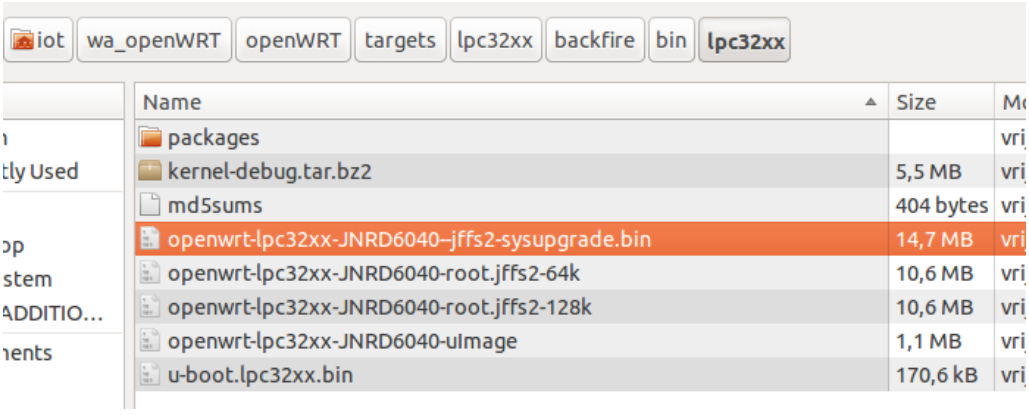


Fig 92. Image selection

- After successful uploading, select **Flash Image…**

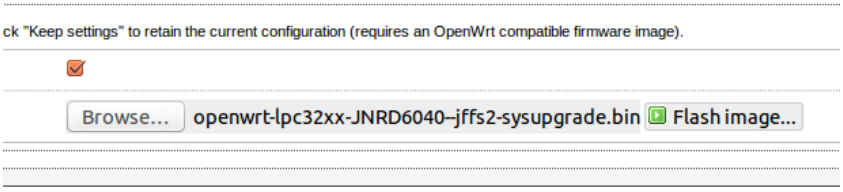


Fig 93. Image selected

- A message appears with the request to verify the checksum of the image.

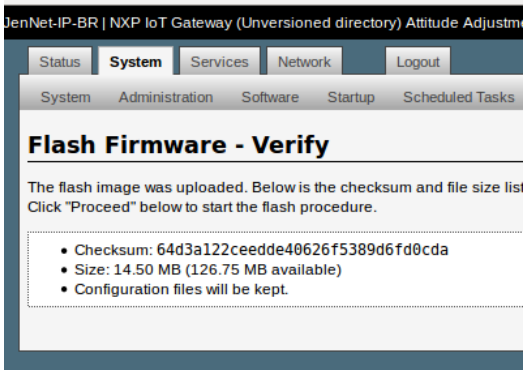


Fig 94. Flash md5 checksum for verification

- The checksum of the build is available in the output directory in the file md5sums.
- Check with: cat md5sums

```
iot@iot-VirtualBox:~/wa_openWRT/openWRT/targets/lpc32xx/backfire/bin/lpc32xx$ cat md5sums
2a8ef570122c4d80348ad52700b464c3 *kernel-debug.tar.bz2
64d3a122ceedde40626f5389d6fd0cda *openwrt-lpc32xx-JNRD6040--jffs2-sysupgrade.bin
2e2fbbda7272867c0e049f6fceef16b7 *openwrt-lpc32xx-JNRD6040-root.jffs2-128k
0b41dd39f18870338fd9298c9033758c *openwrt-lpc32xx-JNRD6040-root.jffs2-64k
c84afc27f943671bfb195504a9f29f54 *openwrt-lpc32xx-JNRD6040-uImage
bd17f7ff008e7ec5f6414851c4da2a4a *u-boot.lpc32xx.bin
iot@iot-VirtualBox:~/wa_openWRT/openWRT/targets/lpc32xx/backfire/bin/lpc32xx$
```

Fig 95. Image md5 checksum

- If equal, press the Proceed button to start flashing the image in the LPC32xx.

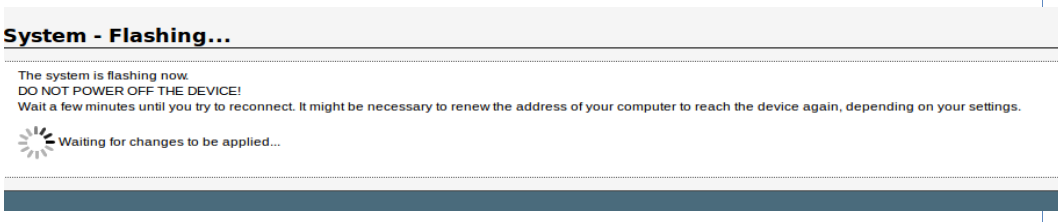


Fig 96. Flashing OpenWrt using webserver progress

- Your browser will report an error ("**The connection was reset**") while flashing is busy; ignore this message.
- Wait till the green LED at the front of the IoT Box start flashing again after about a minute (note: the green LED flashes a few times after start as well, ignore this) and press "**Try again**". Flashing the firmware is finished, when the login screen reappears.

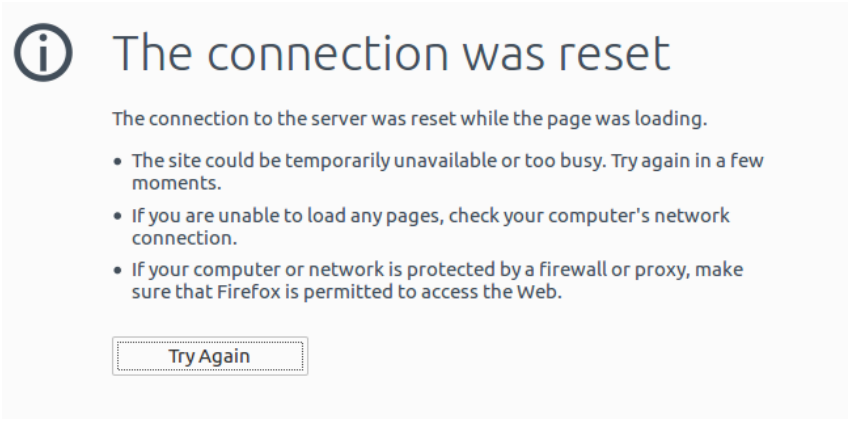


Fig 97. Lost connection during flash process

- Reboot the IoT Gateway

5.2.3.2 Flashing OpenWrt using the hardware serial interface

If programming via the web interface is not possible (e.g. current image in IoT Gateway does not have the web interface working correctly), there is a way to do the flashing under terminal control. The pre-conditions are that there is a working Linux image and a functional IP connection.

- Disconnect the power from the IoT Gateway
- Open a terminal on the Ubuntu VM.
- Open the IoT Gateway box and make a serial connection to a terminal emulator such as RealTerm or Tera Term on the Windows Host PC (see 5.2.3.5 for details).
- Connect the power to the IoT Gateway.

This shows the IoT Gateway boot activity on the screen of the Windows terminal emulator. When no new activity, press **<Enter>** for the prompt "**/#**".

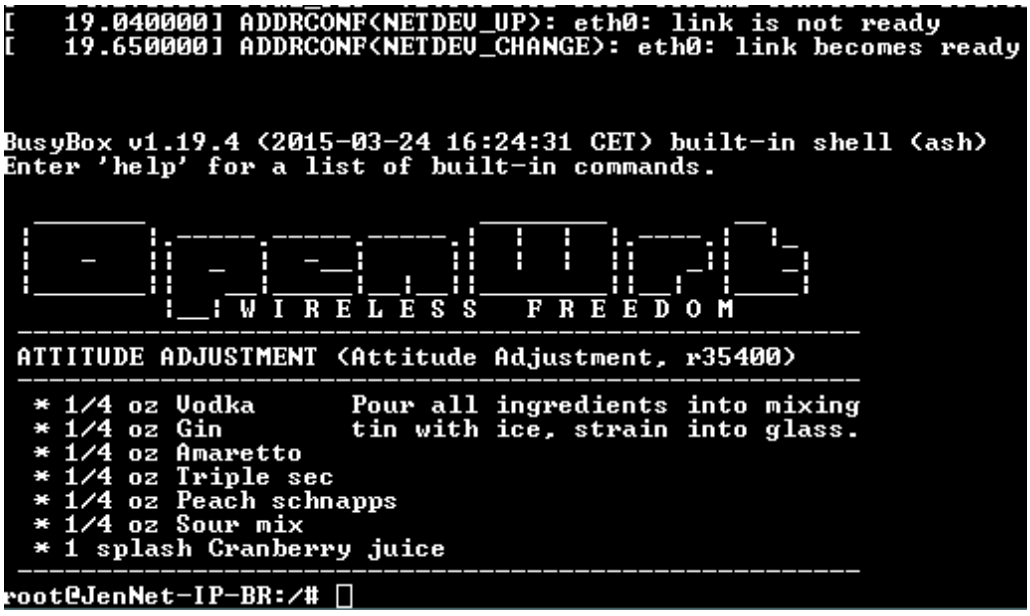


Fig 98. Initial OpenWrt boot activity as seen in the Windows terminal emulator

- On the Windows terminal emulator check if the directory **/tmp** exists on the IoT Gateway. If not then create it: **mkdir tmp**
- On the Ubuntu VM go to the directory containing the new OpenWrt image:

  cd openWRT/targets/lpc32xx/backfire/bin/lpc32xx/
- On Ubuntu VM copy the image from the host to the IoT Gateway (in this example the IP address of the IoT Gateway through the Ethernet connection is 192.168.0.153):

  scp openwrt-lpc32xx-JNRD6040--jffs2-sysupgrade.bin root@192.168.0.153:/tmp.
- If before this session, OpenWrt on the IoT Gateway has been modified/updated, the host key on the IoT Gateway may have been changed. In that case the following error message is shown as a result of strict host key checking:

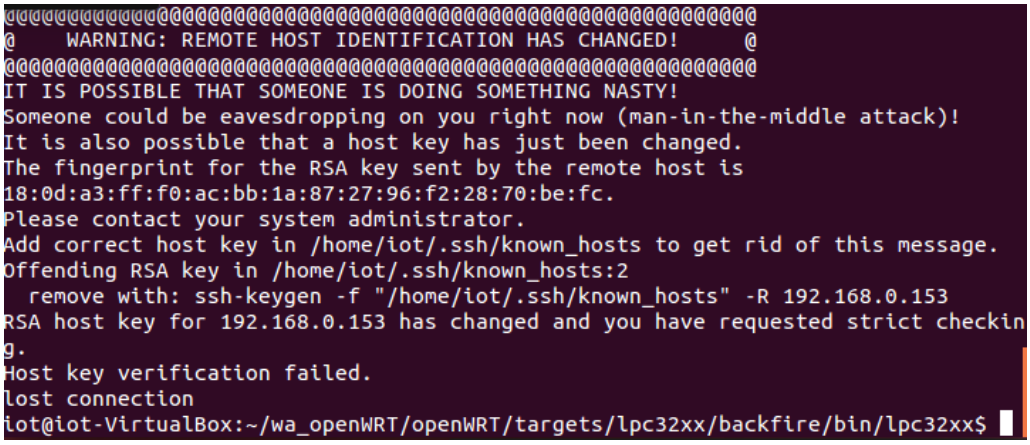


Fig 99. Host key failure on the IoT Gateway after flashing

We have to remove the old remote host key entry from 192.168.0.153 in the **known_hosts** table of the Ubuntu VM (based on http://www.thegeekstuff.com/2010/04/how-to-fix-offending-key-in-sshknown_hosts-file/).

Open a new terminal on the Ubuntu VM and

cd ~/.ssh

sed –i '2d' ./known_hosts

The '2d' has to correspond with the number reported in the line '**Offending RSA key …..known_hosts:2**'of the initial error message.

In the initial terminal window repeat the SSH copy command:

scp openwrt-lpc32xx-JNRD6040--jffs2-sysupgrade.bin root@192.168.0.153:/tmp.

Because we just removed the remote host key for 192.168.0.153, you will be prompted to proceed if you trust 192.168.0.153.

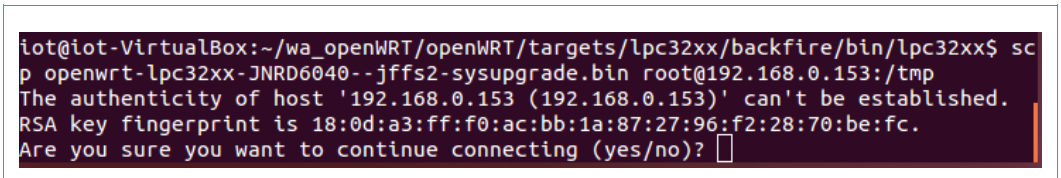


Fig 100. Security warning from IoT Gateway

Enter: **yes**

The new remote host key will be added to the **known_hosts** list. However, 192.168.0.153 may close the connection if you wait too long to enter **yes**.

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.153' (RSA) to the list of known hosts.
Connection closed by 192.168.0.153
lost connection
```

Fig 101. Automatic closure of connection

Repeat the SSH copy command:

scp openwrt-lpc32xx-JNRD6040--jffs2-sysupgrade.bin root@192.168.0.153:/tmp

You will be prompted for the password of 192.168.0.153, and then the copy action will take place.

```
iot@iot-VirtualBox:~/wa_openWRT/openWRT/targets/lpc32xx/backfire/bin/lpc32xx$ sc
p openwrt-lpc32xx-JNRD6040--jffs2-sysupgrade.bin root@192.168.0.153:/tmp
root@192.168.0.153's password:
openwrt-lpc32xx-JNRD6040--jffs2-sysupgrade.bi 100%   15MB   1.1MB/s   00:13
iot@iot-VirtualBox:~/wa_openWRT/openWRT/targets/lpc32xx/backfire/bin/lpc32xx$
```

Fig 102. Successful copy after entering the password

Any subsequent SSH copy actions with **scp** will now be correct again because the known hosts table in Ubuntu has been correctly updated.

- After the copy action has succeeded: on Windows Host PC terminal flash the image in IoT Gateway (from **/tmp** directory):

  sysupgrade –v ./openwrt-lpc32xx-JNRD6040—jffs2-sysupgrade.bin

  The IoT Gateway will reboot after flashing which is visible on the terminal.

5.2.3.3 Flashing OpenWrt using the USB interface

In case only U-boot is working correctly on the target, programming the Root File System and the Linux Kernel has to be done via the USB interface.

- Disconnect the power from the IoT Gateway.
- Open the IoT Gateway and make a serial connection to a terminal on the Windows Host PC (see 5.2.3.5).
- Connect the power to the IoT Gateway. When the message “**Hit any key to stop autoboot**” is shown on the terminal, hit any key. The uboot prompt will be visible.

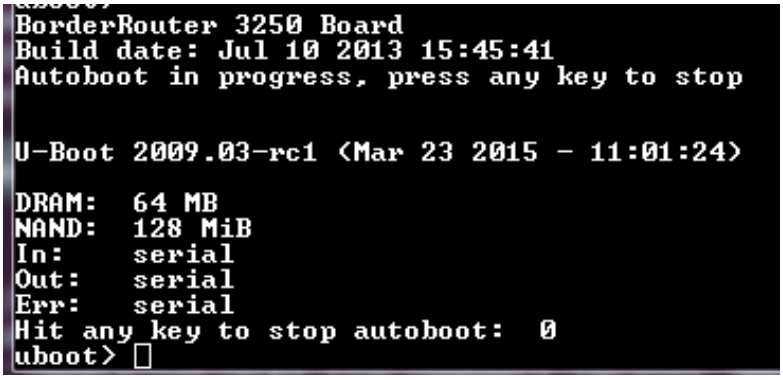


Fig 103. uboot prompt after startup

- On the Ubuntu VM, copy the output files **openWRT-lpc32xx-JNRD6040-root.jffs2-64k** and **openWRT-lpc32xx-JNRD6040-uImage** from the output bin directory (see Fig 89) to a USB stick (may also be done via ~/Share).
- On the USB stick rename

  openWRT-lpc32xx-JNRD6040-root.jffs2-64k to **rootfs.jffs2**

  and

  openWRT-lpc32xx-JNRD6040-uImage to **uImage.lpc32xx**
- Insert the USB stick in the IoT Gateway
- On the IoT Gateway terminal, run the command

  run update_rootfs_usb

  If the message "**Unknown command**" is returned, then the proper uboot needs to be programmed first (see ch.5.2.3.4).

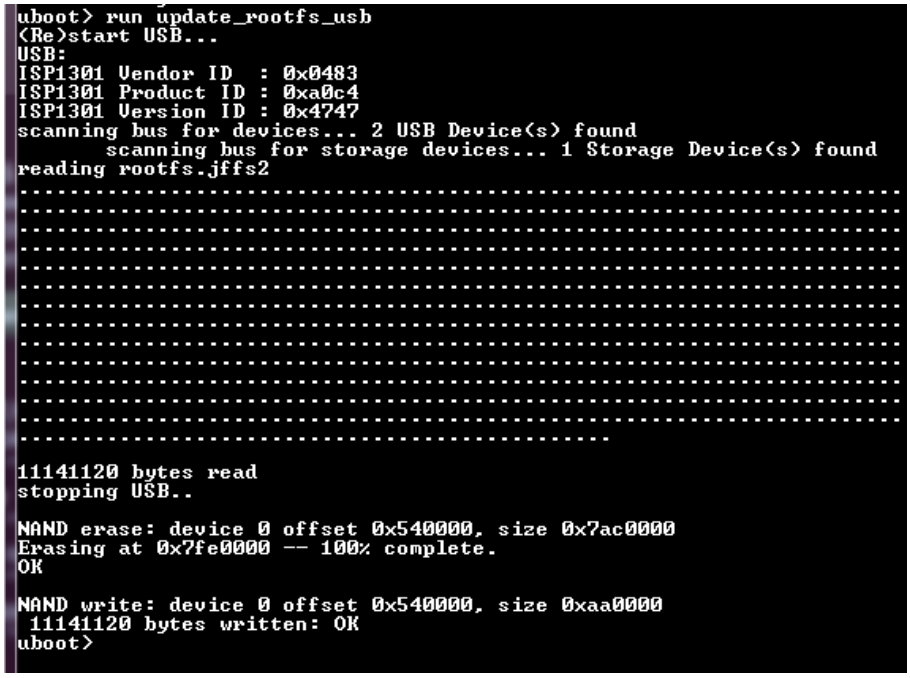


Fig 104. Update rootfs

- On the IoT Gateway terminal, run the command

  run update_kernel_usb

  If the message "**Unknown command**" is returned, then the proper uboot needs to be programmed first (see ch.5.2.3.4).

Fig 105. Update kernel

- The root file system and the kernel are now stored in flash memory. The kernel will be loaded in the file system and booted. The uboot variable named `mtdboot` is

available in the default environment. This variable will setup the boot arguments to use a root file system in NAND flash (in an MTD partition). The `mtdkernel` command will load the kernel into RAM and then `bootm` will boot it.

Enter the (sequential) command:

run mtdboot; run mtdkernel; bootm $(loadaddr)

Boot info is shown on the screen. Once no more new output is seen, hit the return key. When successful, the OpenWrt opening screen becomes available:

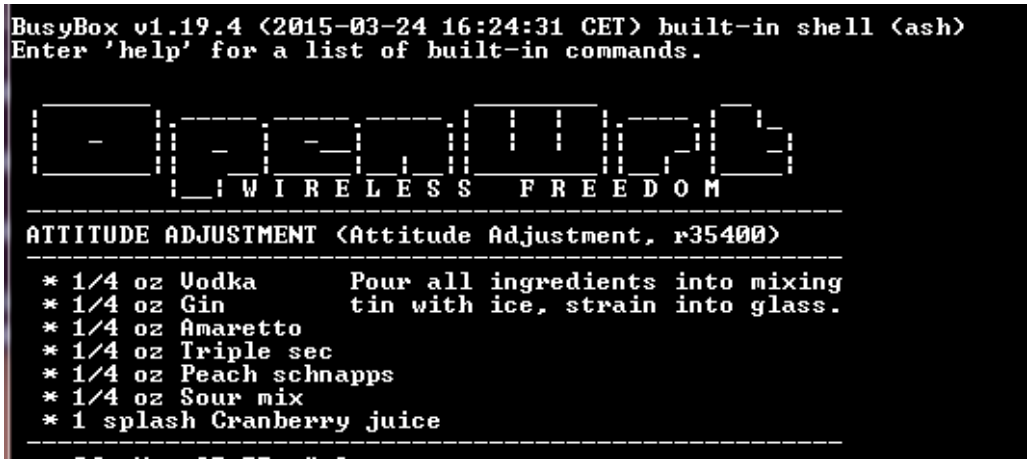


Fig 106. OpenWrt boot sequence after flashing and reboot

The IoT Gateway box may now be closed. Any further updates on OpenWrt can be flashed via the web interface (see ch.5.2.3.1)

5.2.3.4 Programming uboot in IoT Gateway

If the uboot is corrupted or an incorrect version is installed, uboot has to be properly re-flashed before being able to flash any other (user)-images.

- Open the IoT Gateway and make a serial connection to a terminal on the Windows Host PC (see 5.2.3.3). The terminal console running on the Windows Host must be able to send binary data (e.g. **TeraTerm**). Setting for the serial connection are (115200, 8, n, 1).
- Enter boot loader mode. If no uboot is present, this mode is automatically entered.

  If a uboot image is already present, reset the IoT Gateway and press any key after the message **"Autoboot in progress, press any key to stop"** is shown.

  Boot loader mode shows a **br3250** prompt.

Fig 107. Interupt the IoT gateway boot sequence

- The uboot image **u-boot.lpc32xx.bin** is available in the output bin directory (see Fig 89) and may be copied to the **~/Share**.
- On the terminal enter: **load term raw 0x83fa0000**

Fig 108. Programming uboot in IoT Gateway

- On the terminal select: **File->Send File**.
- Select uboot bin file and **check the Binary option!**
- Press **Open** to start data transfer

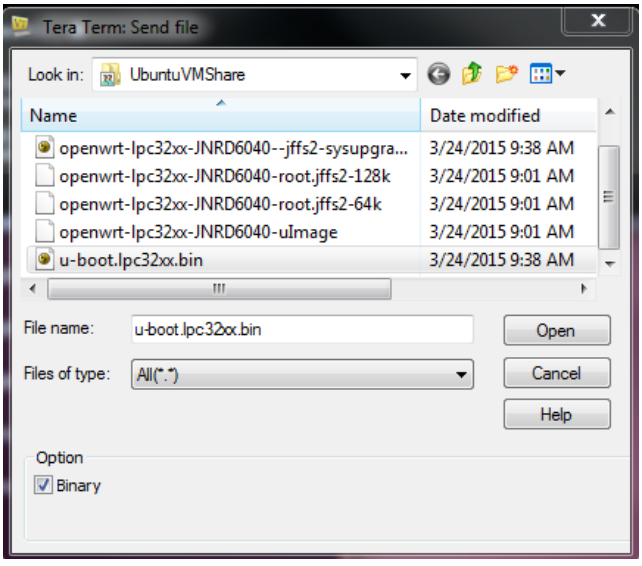


Fig 109. Copy the uboot image over the serial connection

- Progress is shown of the data transfer

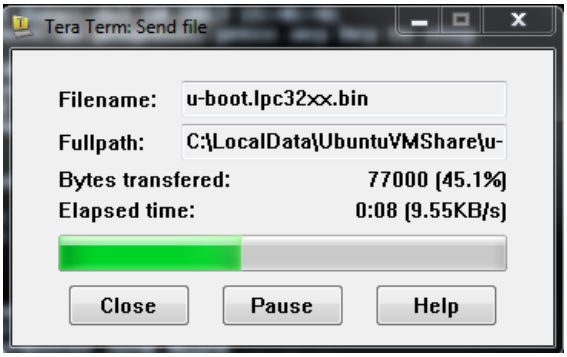


Fig 110. uboot programming progress

- When the file has been transferred, send a Break sequence (**ALT-b)**. The **BR3250>** prompt is shown again.

Fig 111. Uboot programming confirmation

- Save the image in flash by entering: **nsave**
- Set the primary boot loader in order to automatically load uboot; enter

  aboot flash raw 0x83fa0000
- Set the boot delay to 2 seconds and define prompt (use **br3250>** in this example)

  prompt br3250> 2
- Now uboot needs to be setup. Start uboot by entering:

  Boot

  Prevent autoboot by hitting any key. The **uboot>** prompt now shows.

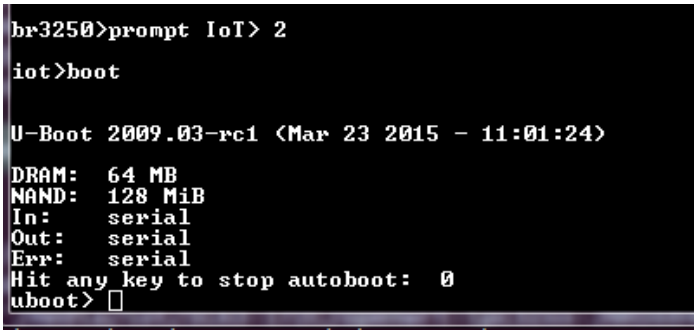


Fig 112. Interrupt boot sequence to enter uboot

- Enter

  print

  to see the uboot environment. Below last part of the output is shown.

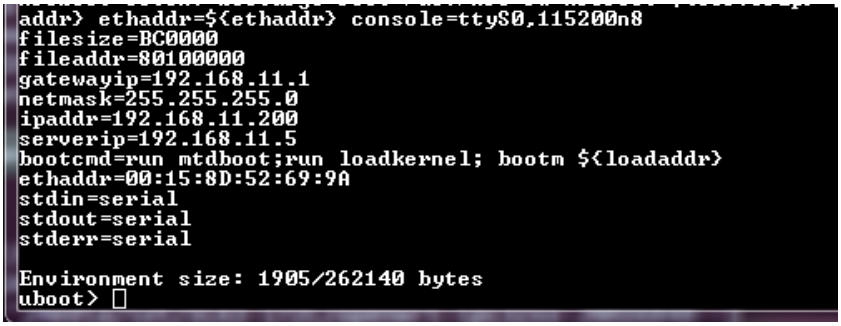


Fig 113. Check uboot environment

- The environment variable **ipaddr** must be changed to any value that is valid on the network it is connected to. In this example we change it to the value it was assigned previously by the DHCP server of the connected router.

  setenv ipaddr 192.168.0.153

  To save the setting:

  saveenv

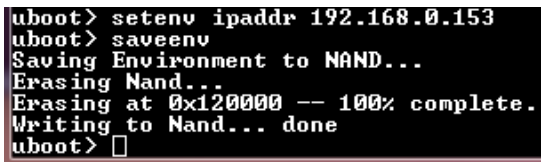


Fig 114. Update the uboot environment variable for the Ip address

- While we are still in uboot>, the root file system and Linux kernel may be loaded from a USB stick as described in 5.2.3.3.

5.2.3.5 Obtaining IPv4 address from IoT Gateway

- If the IoT Gateway is connected to a router with DHCP server, then usually the IP address of the IoT Gateway can be obtained via the web interface of the router.

  Fig 115 shows an example of the DHCP IP table of a Linksys WRT54GS router. 192.168.0.153 is the assigned IP address for the IoT Gateway.

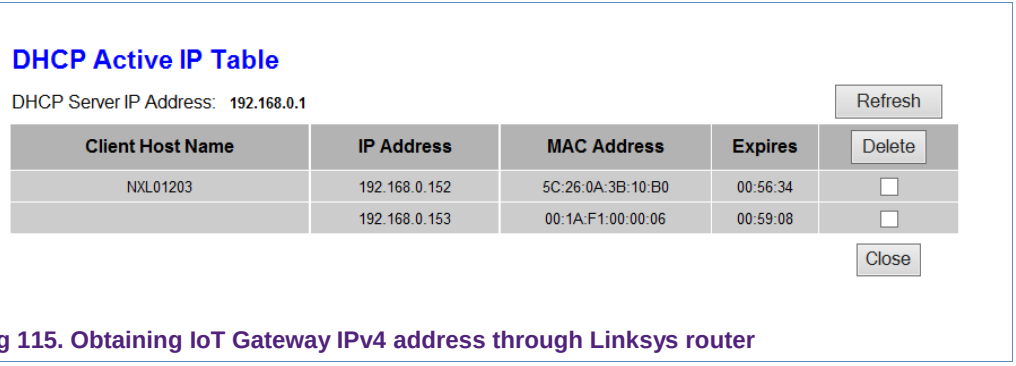


Fig 115. Obtaining IoT Gateway IPv4 address through Linksys router

- If the IP address cannot be derived from a router, then it can derived directly from the IoT Gateway via the serial interface. This assumes an operational OpenWrt image in the IoT Gateway.
- Disconnect the IoT Gateway from the mains and open the box.
- Connect a 3.3V USB-to-Serial cable on the header on the PCB (see Fig 116), pay attention to the orientation of the connector!).

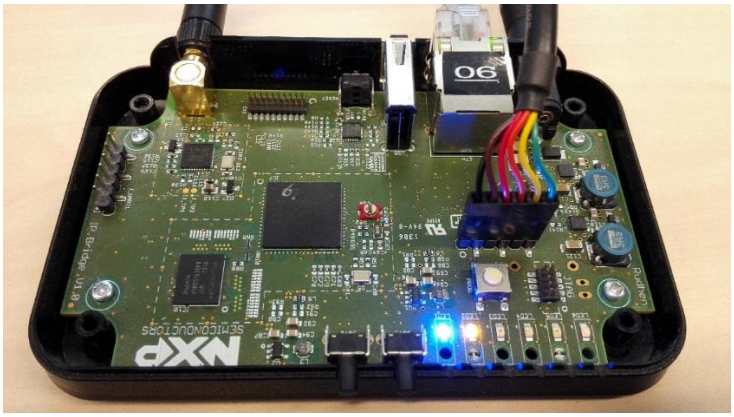


Fig 116. Obtaining IoT Gateway IPv4 address through serial line

- Connect the cable to an USB port of the host PC and open a terminal (e.g. TeraTerm, RealTerm) on the host PC to this COM port.

- Enter

  Ifconfig

  The IP address will appear on the terminal screen (eth0 adapter). In this example the IP address is 192.168.0.153.

Fig 117. ifconfig output

5.2.4 Flashing the Border Router image on JN5168

Now that the LPC32xx has the proper firmware, the JN5168 needs to be programmed with the image of the Contiki stack (see 3.8.1.2). For the JN516x in the IoT Gateway, the baud rate needs to be configured to **230400Bd** (see Fig 22).

Flashing the JN516x is done through the LPC32xx of the IoT Gateway – Host. So, a precondition for flashing the JN5168 is a working OpenWrt image in the LPC32xx and an operational Ethernet connection.

- Install the script to control the tunslip6 daemon (see ch.5.2.5.2).
- On the Windows Host, copy the JN5168 image from host (**border-router.jn516x.bin)** to the shared drive (**UbuntuVMShare** in previous example).
- On the Ubuntu VM, verify that **border-router.jn516x.bin** is available in the **~/Share** directory.
- An SSH connection should be used to remotely operate the IoT Gateway – Host from the Ubuntu VM. On the Ubuntu VM enter:

  ssh root@192.168.0.153

  (for password; see ch.5.2.3.1)

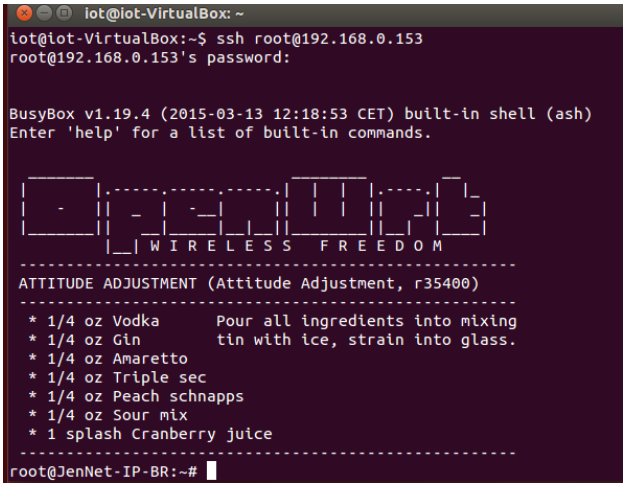


Fig 118. ssh to OpenWrt (on the LPC32xx in the IoT gateway box)

- Before flashing, tunslip needs to be disabled. The tunslip startup script needs to be installed for this (see ch. 5.2.5.2)
- On the SSH client on Ubuntu VM enter

  /etc/init.d/tunslipStartup disable
- Restart the border router by pressing the RESET button on the back. The system will now not enable tunslip at boot time.
- Restart the SSH client on the Ubuntu VM (**ssh root@192.168.0.153**)
- On the Ubuntu VM (in a separate terminal), copy the **border-router.jn516x.bin** image file to the IoT Gateway:

  scp ~/Share/border-router.jn516x.bin root@192.168.0.153:/tmp.

  This assumes the presence of a **/tmp** directory on the LPC32xx (see 5.2.3.2 to create if not available). For password; see ch.5.2.3.1.

```
iot@iot-VirtualBox:~$ scp ~/Share/border-router.jn5168.bin root@192.168.0.153:/t
mp
root@192.168.0.153's password:
border-router.jn5168.bin                      100%   71KB  70.6KB/s   00:00
iot@iot-VirtualBox:~$
```

Fig 119. The JN516x border router image in the OpenWrt file system

- The flashing of the binary on JN516x can now be done by entering the following command on the SSH client:

  JennicModuleProgram.sh /tmp/border-router.jn516x.bin

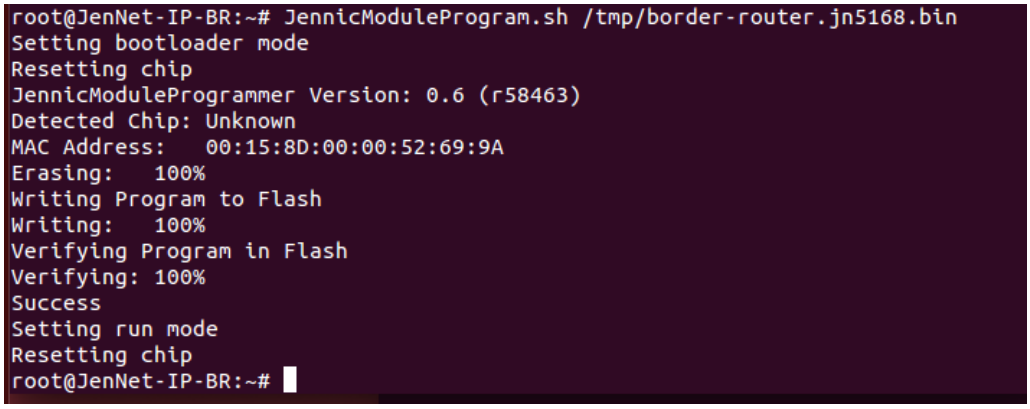


Fig 120. Flashing JN516x image in IoT Gateway

- Re-enable tunslip by entering on the SSH client

  /etc/init.d/tunslipStartup enable

- Restart the border router by pressing the RESET button on the back. Tunslip will be enabled at boot time.

5.2.5 Configuration of IoT Gateway

Firmware images are now available in both the LPC32xx host and JN516x. The next step is to set the proper configuration for required behavior.

5.2.5.1 Verification of tunslip6

Before doing any further configuration or tests, insure that tunslip6 is working correctly. The main reason to do so is that we have patched the build for the correct version.

- Reset the IoT Gateway
- On the Ubuntu VM open a terminal and connect to the IoT Gateway with:

  ssh root@192.168.0.153

  This will show the OpenWrt welcome screen.

- Enter the tunslip command:

  tunslip6 –v –X –B 230400 –s /dev/ttyTX0 aaaa::1/64

  The initial output should be as in Fig 121

```
root@JenNet-IP-BR:~# tunslip6 -v -X -B 230400 -s /dev/ttyTX0 aaaa::1/64
Shall configure now the IP of the TUN (did you connect a Border Router?)
********SLIP started on ``/dev/ttyTX0''
opened tun device ``/dev/tun0''
Start TUN now (is it border router?)
ifconfig tun0 inet `hostname` mtu 1280 up
/bin/sh: hostname: not found
ifconfig tun0 add aaaa::1/64
default route not changed
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-0
-00
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1280  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Waiting for prefix
*** Address:aaaa::1 => aaaa:0000:0000:0000
Waiting for prefix
Obtained prefix: aaaa::
TSCH: starting as coordinator, asn-0.0
RICH: initializing scheduler interface
App: 27034 starting
Server IPv6 addresses:
 aaaa::215:8d00:52:699a
 fe80::215:8d00:52:699a
TSCH: {asn-0.0 link-0-3-42240-26384 16-0-2768243200 ch-26} !skipped link 0 0 1
6LoWPAN: bc sent to 0, st 0 1 (62 bytes)
TSCH: {asn-0.3 link-0-3-0-0 255-7-1 ch-26} bc-1 62 tx 0, st 0-1
6LoWPAN: bc sent to 0, st 0 1 (62 bytes)
TSCH: {asn-0.6 link-0-3-0-0 255-7-1 ch-26} bc-1 62 tx 0, st 0-1
TSCH: enqueue EB packet
```

Fig 121. Verification of tunslip6

- The terminal will keep on displaying new Contiki's TSCH, RPL and 6LoWPAN messages.
- Make a note of the IPv6 server address. It will be used to access the 6LoWPAN network over IPv6. In this example the IPv6 address of the IoT Gateway is [**aaaa::215:8d00:52:699a**]

5.2.5.2 Installation startup script for tunslip6 daemon

The script is used to automatically start the tunslip6 daemon after switching on the IoT Gateway. It has to be manually added to the file system of the IoT host.

- On Ubuntu VM: Open terminal for communication with IoT Gateway

  ssh root@192.168.0.153

- Create the file **tunslipStartup** with the content as shown in Fig 122 and store it in **/etc/init.d** of the IoT Gateway (e.g. create with **gedit** on Ubuntu VM and transfer to IoT Gateway with **scp** command).

```
#!/bin/sh  /etc/rc.common
START=99
STOP=99
start(){
        echo start tunslip6
        # commands to launch application
        tunslip6 -v -X -B 230400 -s /dev/ttyTX0 aaaa::1/64
}
stop() {
        echo stop tunslip6
        # commands to stop application
}
```

Fig 122. Startup script for tunslip6 daemon

- **/etc/init.d** contains the scripts that configure the start/stop behavior of the daemons in the OpenWrt system (see http://wiki.openwrt.org/doc/techref/initscripts)
- Make script executable with:

  chmod a+x /etc/init.d/tunslipStartup
- To ensure tunslip6 is started after booting:

  /etc/init.d/tunslipStartup enable

  tunslip6 becomes active after the next startup.
- If it is needed to disable the tunslip (e.g. for flashing JN516x) run:

  /etc/init.d/tunslipStartup disable
- For immediate start control you can use the command:

  /etc/init.d/tunslipStartup start

  Stopping can only be done via the disable at boot, with the command mentioned above.

5.2.5.3 IoT Gateway border router settings

The radvd (IPv6 Router Advertisement Daemon) in the IoT Gateway needs to be properly configured. This will make the 6LoWPAN network ‘visible’ for other devices on the IPv6 network.

- Restart/reset the IoT Gateway. Tunslip6 must be enabled either via the startup script (ch.5.2.5.2) or via manual start (ch.5.2.5.1)
- On the Ubuntu VM: Open the Firefox web browser to access the webserver of the IoT Gateway (192.168.0.153 in this example). Login as described in ch.5.2.3.1.
- Select **Network->Radvd**

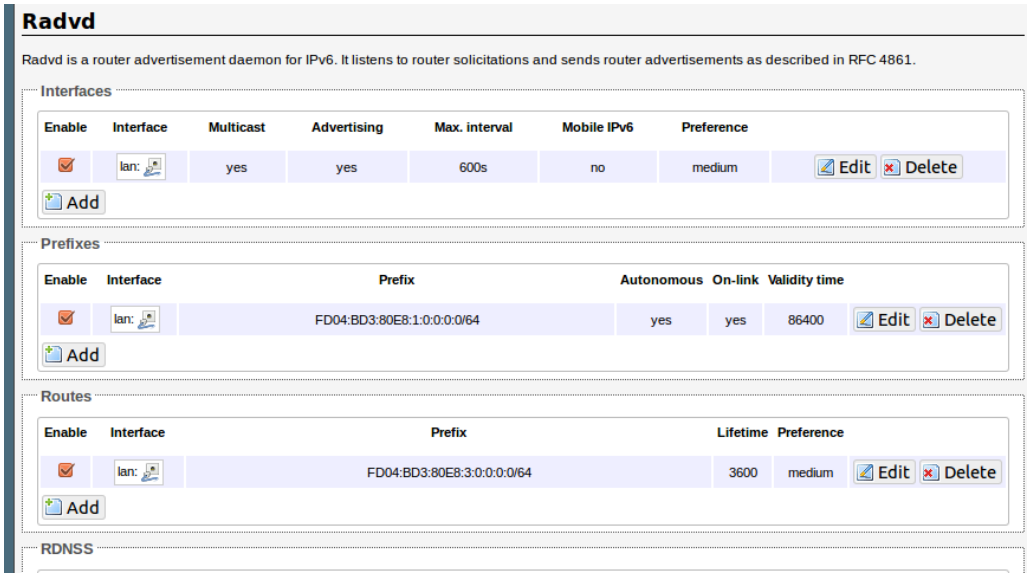


Fig 123. IoT Gateway border router settings via luci (OpenWrt web interface)

- The only items enabled must be **Interfaces**, **Prefixes** and **Routes**
- Leave the **Prefix** in **Prefixes** as it is, even if it has a different value than shown in Fig 123.
- On the item **Routes**, enter **Delete**, then **Add**
- Settings as in Fig 124 then **Save & Apply**

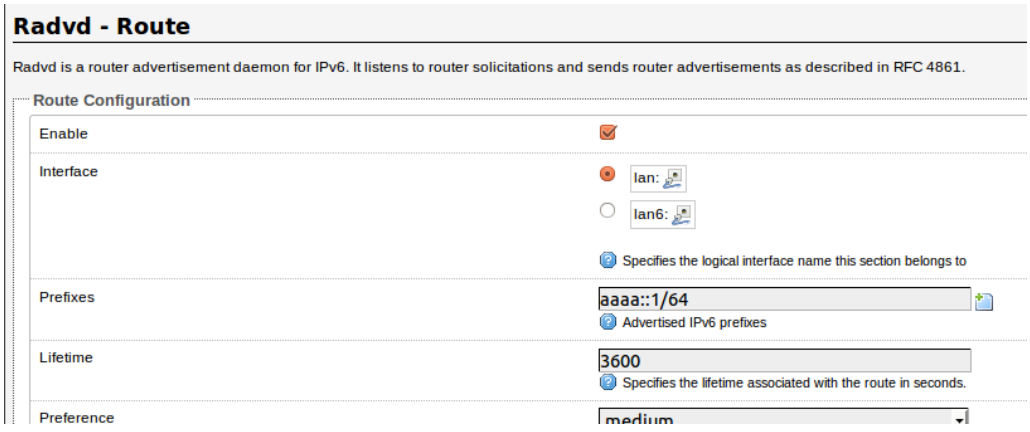


Fig 124. Adding a route

- Verify the settings in the item **Routes**:

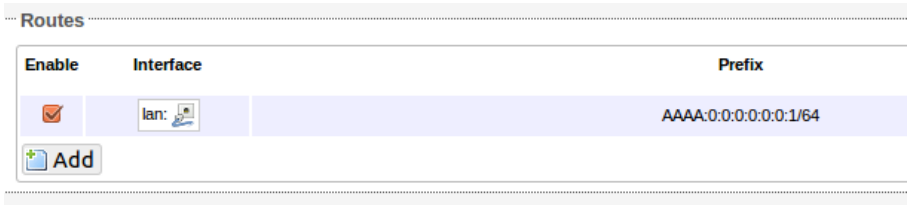


Fig 125. Adding route verification

- In **Radvd** overview menu: select **Save & Apply**
- Close the browser
- The radvd setting can be verified in **/etc/config/radvd** on the IoT Gateway as well.

```
root@JenNet-IP-BR:/etc/config# cat radvd

config interface
        option interface 'lan'
        option AdvSendAdvert '1'
        option ignore '0'
        option IgnoreIfMissing '1'
        option AdvSourceLLAddress '1'
        option AdvDefaultPreference 'medium'

config rdnss
        option interface 'lan'
        list addr ''
        option ignore '1'

config dnssl
        option interface 'lan'
        list suffix ''
        option ignore '1'

config prefix
        option ignore '0'
        option interface 'lan'
        option AdvOnLink '1'
        option AdvAutonomous '1'
        list prefix 'fd04:bd3:80e8:1::/64'
        option AdvRouterAddr '1'

config route
        option ignore '0'
        option interface 'lan'
        list prefix 'aaaa::1/64'
        option AdvRouteLifetime '3600'
        option AdvRoutePreference 'medium'

root@JenNet-IP-BR:/etc/config#
```

Fig 126. IoT Gateway border router settings

5.2.5.4 Verification of IoT Gateway

A number of simple tests can be done to verify the links established with the IoT border router. These tests assume that the **tunslipStartup** script enables tunslip6 after booting (see ch.5.2.5.2), otherwise start it manually.

The set-up consists of an IoT Gateway and 2 end nodes. The IoT Gateway is connected to a LAN. If connected via a router, verify that the router can handle IPv6.

The following addresses are present:

IoT Gateway	IPv4	134.27.208.47	See ch.5.2.3.5
	MAC JN5168	00:15:8D:52:69:9A	At bottom of box or when programming JN5168 in the IoT Gateway (Fig 120)
	IPv6	aaaa::215:8d00:52:699a	Derived from MAC or after tunslip (see ch.5.2.5.1)
End Node 1	MAC JN516x	00:15:8D:35:CA:C7	Obtained when flashing (see Fig 18)
	IPv6	aaaa::215:8d00:35:cac7	Derived from MAC
End Node 2	MAC JN516x	00:15:8D:32:D6:A0	Obtained when flashing
	IPv6	aaaa::215:8d00:32:d6a0	Derived from MAC

- Power on nodes and IoT Gateway. Wait about 30 seconds for IoT Gateway to boot.
- First test if the IoT Gateway and the end nodes can be pinged (e.g. from Ubuntu VM). It make take a while before both nodes are visible on their IPv6 address. Notice difference in ping time between IoT Gateway and the end nodes.

```
iot@iot-VirtualBox:~$ ping6 aaaa::215:8d00:52:699a
PING aaaa::215:8d00:52:699a(aaaa::215:8d00:52:699a) 56 data bytes
64 bytes from aaaa::215:8d00:52:699a: icmp_seq=1 ttl=63 time=20.7 ms
64 bytes from aaaa::215:8d00:52:699a: icmp_seq=2 ttl=63 time=17.6 ms
64 bytes from aaaa::215:8d00:52:699a: icmp_seq=3 ttl=63 time=15.8 ms
^C
--- aaaa::215:8d00:52:699a ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 15.805/18.051/20.748/2.045 ms
iot@iot-VirtualBox:~$ ping6 aaaa::215:8d00:35:cac7
PING aaaa::215:8d00:35:cac7(aaaa::215:8d00:35:cac7) 56 data bytes
64 bytes from aaaa::215:8d00:35:cac7: icmp_seq=1 ttl=62 time=207 ms
64 bytes from aaaa::215:8d00:35:cac7: icmp_seq=2 ttl=62 time=240 ms
64 bytes from aaaa::215:8d00:35:cac7: icmp_seq=3 ttl=62 time=184 ms
^C
--- aaaa::215:8d00:35:cac7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 184.099/210.928/240.911/23.303 ms
iot@iot-VirtualBox:~$ ping6 aaaa::215:8d00:32:d6a0
PING aaaa::215:8d00:32:d6a0(aaaa::215:8d00:32:d6a0) 56 data bytes
64 bytes from aaaa::215:8d00:32:d6a0: icmp_seq=6 ttl=62 time=179 ms
64 bytes from aaaa::215:8d00:32:d6a0: icmp_seq=7 ttl=62 time=167 ms
64 bytes from aaaa::215:8d00:32:d6a0: icmp_seq=8 ttl=62 time=201 ms
64 bytes from aaaa::215:8d00:32:d6a0: icmp_seq=9 ttl=62 time=280 ms
^C
--- aaaa::215:8d00:32:d6a0 ping statistics ---
9 packets transmitted, 4 received, 55% packet loss, time 8037ms
rtt min/avg/max/mdev = 167.970/207.442/280.300/43.773 ms
iot@iot-VirtualBox:~$
```

Fig 127. Verification of IoT Gateway - ping

- The next verification step checks if CoAP resources on a node can be monitored or modified from a host.

  This test makes use of the NXP evaluation boards DR1174 (base board) and DR1199 (generic expansion with buttons, LEDs and potentiometer).

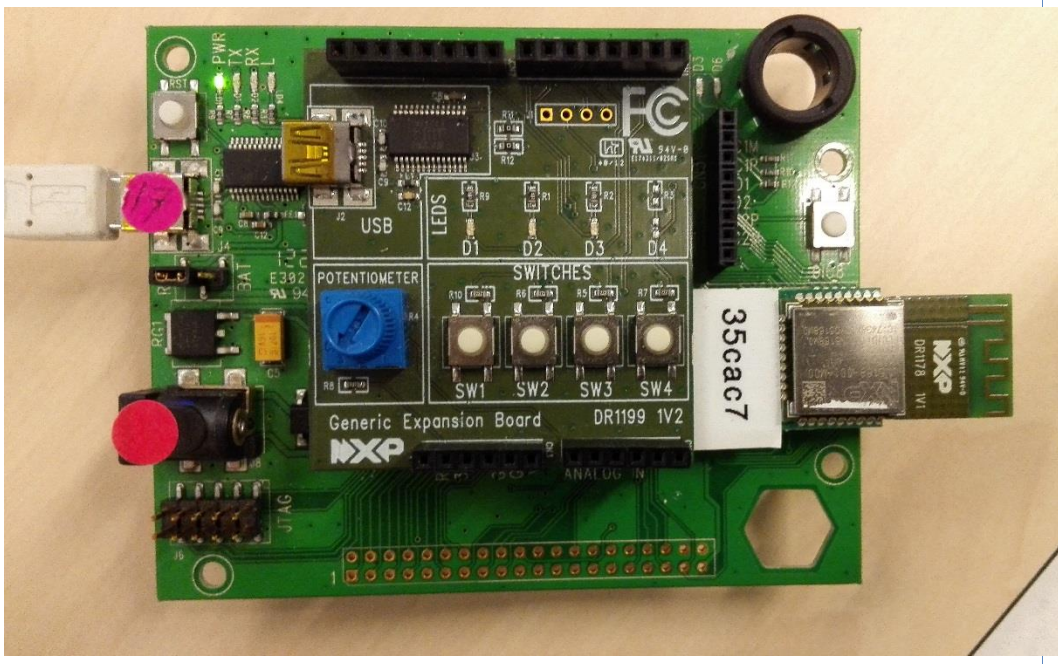


Fig 128. Base board and shield used for verification

- The binary flashed on this target will be available in **\examples\rich\dr1199-node** of the EIT_ICT_RICH/contiki repository. This rich node application provides a CoAP interface of the resources of the expansion board. In this example the mac address of the JN516x on the base-board had the extension 35cac7.
- On the Ubuntu VM: Open Firefox and enter the following url:

  coap://[aaaa::215:8d00:35:cac7]:5684/
- Click on the top level (home icon) on the left side directory tree. This will initiate a CoAP discovery action. A few retries may be needed while the node is trying to connect to the border router.

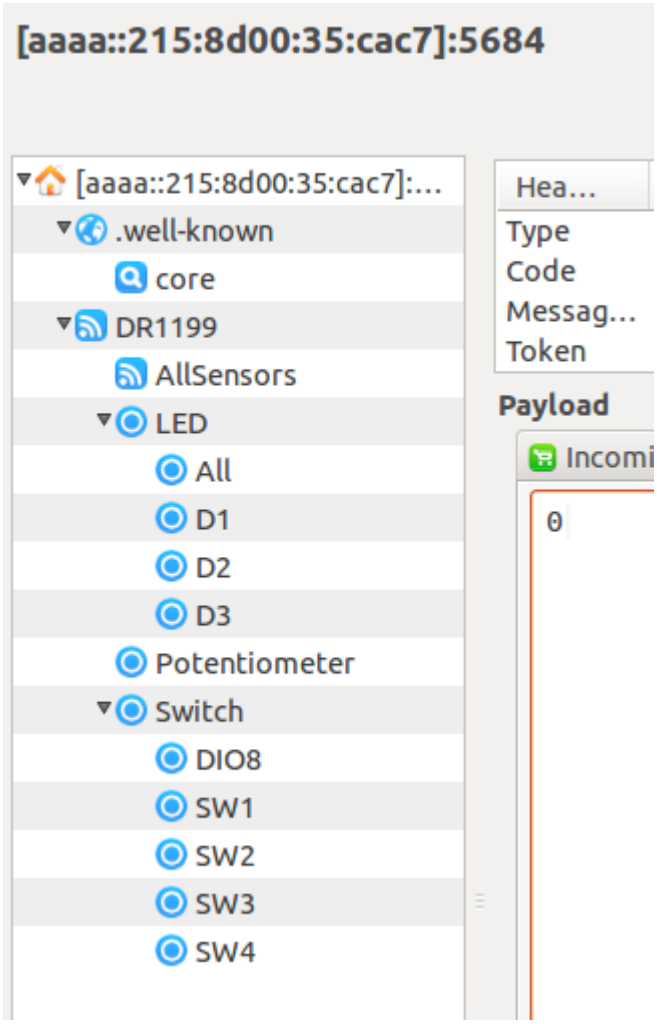


Fig 129. CoAp resources as viewed through the Copper add-in to Firefox

- Select **Potentiometer** from the left hand directory structure.
- Press the **GET** button from the tool bar, and the **Incoming** button above the payload plane.
- The value of the potentiometer on the expansion board is shown. Modify the potentiometer setting and press **GET** again to see the value change.

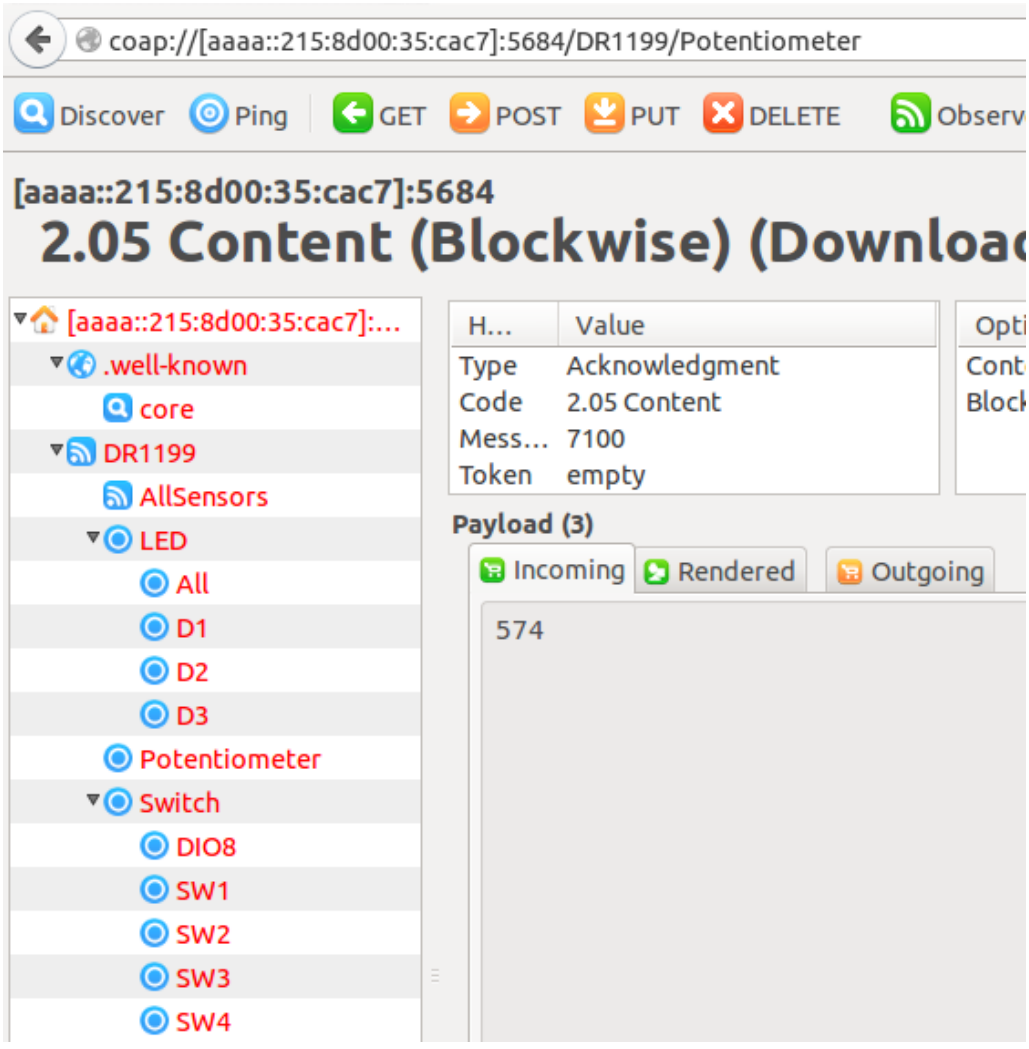


Fig 130. Value of potentiometer resource example

- Select one of the **LED** entries from the left hand directory structure. Enter **1** in the payload field and press the **POST** or **PUT** button. The corresponding LED on the expansion board should (after some delay) switch on.
- Select one of the **AllSensors** entry from the left hand directory structure. Press the Observe button. Whenever a switch value changes or the potentiometer value changes, this is directly shown in the payload field (parsed JSON payload).

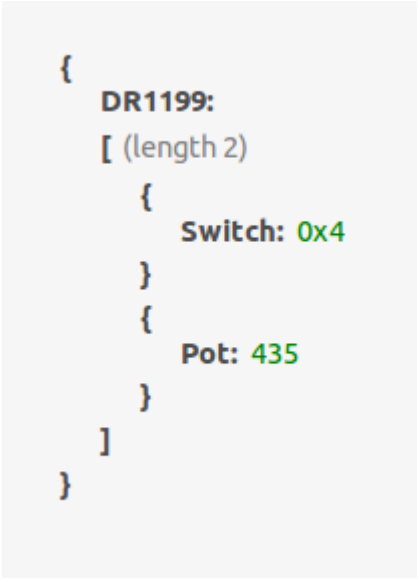


Fig 131. All sensor resources (JSON) in CoAP

- The example above used Firefox running on the VirtualBox VM, so that everything is done within the same environment. This may severely slow down the execution speed. It is also possible to install and run Firefox (with Copper plug-in) directly under your native OS; i.e. under Windows on a Windows PC (see 3.5).

6. List of figures

7. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of releaseJanuary 2016

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of releaseJanuary 2016