Contiki Development Environment Setup

Authors

Rikard Höglund rikard.hoglund@ri.se
Kiki Rizki kiki.rizki@ri.se

Purpose

The purpose of this document is to detail how to set up a development environment for Contiki with special focus on the CC2538 Texas Instrument boards. This document describes how to build a setup in which it is possible to program and compile applications using Contiki. It also covers how applications can be simulated with the Cooja network simulator. Finally it describes how applications can be flashed and loaded onto physical hardware such as the CC2538 board.

Revision history

Version	Date	Name	Comments
1.0	2014-03-23	Rikard	Initial version.
1.1	2014-03-25	Rikard	Updated commands.
1.2	2014-03-27	Rikard	Clarified some steps.
1.3	2014-04-07	Rikard	Added missing steps found from tests.
1.4	2014-04-15	Rikard	IPv6 port forwarding instructions.
1.5	2014-12-04	Rikard	Added note regarding loading drivers.
1.6	2017-03-13	Rikard	Checked links, fixed ccxml path & note on Cooja.
1.7	2018-01-15	Rikard	Added command to load drivers in newer kernels.
1.8	2018-01-15	Kiki	Sections on Python script & performance evaluation

Contents

1 Instant Contiki

Contiki has a prepared image for software development. This comes in the form of "Instant Contiki" and can be downloaded at the following website [1]. It includes the compilers necessary for compiling Contiki on a variety of hardware, other useful tools and the Cooja network simulator used for simulating Contiki nodes.

1.1 Installation

The following steps are necessary to setup and install the Instant Contiki development image file using VMware. This example uses VMware but other software such as VirtualBox is also possible to use.

1. Download VMware and the Instant Contiki image file from [1].

2. Install VMware Player.
   ```
   chmod +x VMware-Player-6.0.1-1379776.i386.bundle
   sudo ./VMware-Player-6.0.1-1379776.i386.bundle
   ```
   Follow the installation steps and skip entering license key for now.

3. Reboot

4. Unpack the Instant Contiki image file (filename may vary).
   ```
   unzip InstantContiki2.7.zip
   ```

5. Load the image in VMware by double clicking on the .vmx file.
   Instant_Contiki_Ubuntu_12.04_32-bit.vmx

6. At this point VMware may ask about Kernel headers. If so specify their location if available. Otherwise the kernel headers for your kernel have to be downloaded. Note that the kernel may have to be updated to fit with header files or vice versa. A new kernel may have to be installed to match the header files available in the software manager.

7. Follow the steps indicated to install the VMware Kernel Modules. Insert your password when asked. The window asking for the root password may appear bugged and incomplete.

8. VMware may say that "long mode" is incompatible for this machine, press OK. This can be related to BIOS settings and the hardware of the host device.

9. VMware may ask if you moved or copied the image files. Answer that they were copied.

10. If prompted install the VMware Linux tools.

11. Now you are in the Instant Contiki environment. Login using password "user".

2 Cooja

The network simulator, named Cooja, allows developers to easily test and develop software for the Contiki platform. It supports extracting statistics from nodes and monitoring the network traffic to analyze what data is being sent over the network. Cooja will dynamically compile the entire Contiki source including any changes made and start the application specified on the nodes.

2.1 Simulating applications

These are the basic steps for how to run an example application in the Cooja network simulator. Cooja uses special configuration files that define a simulation, either these can be used or custom simulations can be created.

1. Start Cooja by double clicking the link on the desktop of Instant Contiki or executing:
   ```
   cd /home/user/contiki/tools/cooja
   ant run
   ```

2. Configuration files for some example applications can be found under
   /home/user/contiki/examples/

3. Select File->Open simulation->Browse...
   Choose the following configuration file:
   /home/user/contiki/examples/er-rest-example/server-client.csc
   Press Open.

4. This simulation will load a CoAP server on one node and a CoAP client on another. Meaning Cooja will now compile and deploy the *er-example-server.c* code on one node, *er-example-client.c* on another and also a border router application from *ipv6/rpl-border-router/border-router.c* on a third node.

5. After the compilation is finished and the simulation has loaded press the Start button.

6. When the simulation starts information on each node will be printed, among it are the IPv6 addresses of each node. Note that the IP address in the *er-example-client.c* code may need to be changed for the example to work. The client needs to know the IP of the server node.

   First observe what the server node gives as output when the simulation starts:
   "Tentative link-local IPv6 address xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx"

   Now modify this line in */home/user/contiki/examples/er-rest-example/er-example-client.c*:
   #define SERVER_NODE(ipaddr)   uip_ip6addr(ipaddr, 0xaaaa, 0, 0, 0, 0x0212, 0x7402, 0x0002, 0x0202) / cooja2 */*

7. Restart the simulation.

8. Right click the client node in the topology view and select "Click button on Sky 3" on it to generate a CoAP request to the server.

2.2 Sniffing network traffic

Cooja supports sniffing of network traffic generated by the simulation and also dumping the traffic to external pcap files that can later be reviewed by programs like Wireshark. The following steps show how to capture network traffic in Cooja.

1. Start a simulation
2. Go to Tools->Radio messages...
3. Select Analyzer->6LoWPAN Analyzer with PCAP
4. Check the directory Cooja is running in for the pcap output file. This is typically: */home/user/contiki/tools/cooja/build*
5. Open the latest generated pcap file with a tool like Wireshark.

3 Flashing boards

These steps show how to set up the Instant Contiki environment for deploying applications on boards. The following websites can provide additional helpful information [2][3][4][5][6]. This example focuses on the CC2538 model boards connected to a SmartRF board as the main platform.

1. Start Instant Contiki.

2. Connect the board to a USB port.
   Make sure the jumper settings on the board are correct and also the USB cable should connect the computer to the SmartRF board, not directly to the CC2538 module.

3. Check with `dmesg` or `lsusb` that the USB device is found correctly.

4. Check for the existence of the correct ARM compiler.
   ```
   arm-none-eabi-gcc -v
   ```

5. Edit modem settings to prevent potential problems. (Board may be detected as a modem).
   ```
   sudo nano /lib/udev/rules.d/77-mm-usb-device-blacklist.rules
   ```
   Add these lines:
   ATTRS{idVendor}=="0451", ATTRS{idProduct}=="16c8", ENV{ID_MM_DEVICE_IGNORE}="1"
   ATTRS{idVendor}=="0403", ATTRS{idProduct}=="a6d1", ENV{ID_MM_DEVICE_IGNORE}="1"

   Run the following to restart the modem manager.
   ```
   sudo service modemmanager restart
   ```

6. Restart the system.

7. Run the following to load the correct drivers for the SmartRF board.
   ```
   sudo modprobe ftdi_sio vendor=0x403 product=0xA6D1
   ```
   Note that this has to be done after every reboot or put in a startup script.

   If using a Linux kernel version of 3.12 or above, the following commands can be used.
   ```
   modprobe ftdi_sio
   echo "0403 a6d1" > /sys/bus/usb-serial/drivers/ftdi_sio/new_id
   ```

8. Install the `minicom` and `screen` serial communicator programs.
   ```
   sudo apt-get install minicom screen
   ```

9. Setup `minicom` using the following.
   ```
   sudo minicom -s
   ```

   Enter this configuration:
   Use 115200 8N1 (bps/parity/bits)
   Set device to /dev/ttyUSB1 (can vary)
   Exit and save as default configuration

10. Now the board can be reached using `minicom` or `screen`.
    ```
    sudo minicom
    sudo screen /dev/ttyUSB1 115200
    ```

3.1 UniFlash

UniFlash is one of the programs that can be used for transferring applications to a board. However it is not the program with the most functionality and appears to be unstable sometimes. Follow these steps to install UniFlash.

1. Get the UniFlash tool for writing images to the board via [7].
   (Registration required.)

2. Unpack the UniFlash installation software.

3. Run the installation binary (the filename may vary).
   ```
   chmod +x uniflash_setup_3.0.0.00031.bin
   sudo ./uniflash_setup_3.0.0.00031.bin
   ```

4. Follow the installer steps.
   Either install the full feature set or only what is needed for the specific device.
   This example uses */opt/ti/* as the installation directory.

5. Install the drivers.
   ```
   sudo /opt/ti/uniflashv3/install_scripts/install_drivers.sh
   ```

6. Start the UniFlash GUI.
   ```
   sudo /opt/ti/uniflashv3/eclipse/uniflash
   ```

7. Select the following configuration file when prompted:
   /opt/ti/uniflashv3/eclipse/plugins/com.ti.cc2538_1.1.2.0/resources/blinky/ccs/CC2538SF53 .ccxml

8. To actually transfer an application to the board do the following:
   Select Program->Load Program.
   Select the binary file (*.elf) that is to be transferred to the board.
   Start the transfer.

3.2 Code Composer Studio

The Code Composer Studio is another tool that can be used to transfer applications to a board. It not only includes functionality for transferring software to boards but also a full IDE environment. It seems more stable compared to UniFlash and the results are more consistent when trying to flash a board. Follow these instructions to install the Code Composer Studio.

1. Download the Code Composer Studio from [8]. This guide uses the online installer which automatically downloads the required components form the Internet.

2. Run the installation file (the filename may vary).
   ```
   chmod +x ccs_setup_5.5.0.00077.bin
   sudo ./ccs_setup_5.5.0.00077.bin
   ```

3. Follow the installation steps. This example uses */opt/ti/* as the installation directory.
   Select the following parts for installation:

   Processor Support: Wireless Connectivity CCxxxx Cortex M Devices
   Components: Compiler Tools
   Emulators: JTAG Emulator Support

4. Install drivers by running.
   ```
   sudo /opt/ti/ccsv5/install_scripts/install_drivers.sh
   ```

5. Start the Code Composer Studio using:
   ```
   sudo su
   /opt/ti/ccsv5/eclipse/ccstudio
   ```

6. Choose a workspace folder. (The default is fine.)

7. Choose the 90 day evaluation license and follow the instructions to generate a license file. Note that this takes a number of steps and requires registration.

8. Go to Help->Code Composer Studio Licensing Information
   Follow the instructions to register the license.

9. Because the Code Composer Studio is run as root the correct compilers have to be in the $PATH of the ROOT user.

   One way to fix this is by creating symlink from the compilers directory to /usr/bin:
   ```
   sudo ln -s /home/user/CodeSourcery/Sourcery_G++_Lite/bin/arm-none-
   eabi-* /usr/bin/
   ```

   Another is to simply add the directory of the compilers to the $PATH of the ROOT user, for instance in the .bashrc configuration file add the following line:
   PATH=$PATH:/home/user/CodeSourcery/Sourcery_G++_Lite/bin

10. Before the writing to the board can start one of the platform specific C files for the CC2538 board has to be replaced. This alleviates a problem where the flashing process locks and continuously waits for the board to reset.

    For Contiki 2.7 or older, replace the following file with the replacement provided: */home/user/contiki/platform/cc2538dk/startup-gcc.c*

    Alternatively apply this `diff`:

```
--- startup-gcc.c.old   2014-03-10 18:18:07.660678790 +0100
+++ startup-gcc.c       2014-03-11 18:07:42.232666342 +0100
@@ -305,7 +305,16 @@
   unsigned long *pul_src, *pul_dst;

   REG(SYS_CTRL_EMUOVR) = 0xFF;
-
+
+  volatile uint32_t* pStopAtResetIsr = (uint32_t*)0x20003000;
+  volatile uint32_t* pIsAtResetIsr = (uint32_t*)0x20003004;
+
+  *pIsAtResetIsr = 0xAABBAABB;
+
+  volatile uint32_t ui32Timeout = 2000000;
+  while((*pStopAtResetIsr == 0xA5F01248) && (ui32Timeout--));
+    *pIsAtResetIsr = 0x0;
+
   /* Copy the data segment initializers from flash to SRAM. */
   pul_src = &_etext;

@@ -327,7 +336,7 @@
   main();

   /* End here if main () returns */
-  while(1);
+  //while(1);
 }
 /*------------------------------------------------------------------------
-----*/
```

3.2.1 Deploying applications to board by using Code Composer Studi

The following instructions detail how to compile and transfer an application to a board using Code Composer Studio.

1. Start Code Composer Studio:
   ```
   sudo su
   /opt/ti/ccsv5/eclipse/ccstudio
   ```

2. Go to File->New->Project...

3. C/C++->Makefile Project with Existing Code
   Next.

4. Select "GNU Autotools Toolchain"

5. Browser for the location of the application code.
   Select the **folder** that contains the code. (e.g. */home/user/contiki/examples/er-rest-example*)
   Finish.

6. Right click project in Project Explorer frame
   Properties.

7. Go to C/C++ Build tab.
   Uncheck "Use default build command"
   Enter "`make TARGET=cc2538dk`" as new build command
   Apply.
   OK.

8. Right click project in Project Explorer frame
   Build Project.

9. The project will now be compiled for the CC2538 board.
   Make sure the board is connected and the previous configuration steps are done.

10. Right click project in Project Explorer frame
    Debug As->Debug Configurations...
    Select as Target Configuration the *CC2538SF53.ccxml* file. This file can be found under:
    /opt/ti/ccsv5/eclipse/plugins/com.ti.cc2538_1.1.2.0/resources/blinky/ccs/CC2538SF53.ccxml

11. Go to Program tab
    Project: Press Workspace... and select the project.
    Program: Press Workspace... and select the executable .elf file that is to be loaded.
    Apply.
    Debug.

12. The code will now be loaded onto the board. When the transfer is finished press the red square in the top left to stop debugging. This procedure will take some time, up to 5 minutes.

13. Now connect to the board with screen or minicom and observe the output.
```
sudo minicom
sudo screen /dev/ttyUSB1 115200
```

14. Note that the board may have to be reset using the physical reset button. In addition to that often the flashing procedure has to be repeated twice for the program to actually be loaded onto the board. In fact one reliable method is to simply flash the boards two times in a row to ensure that the application is written properly.

3.3 Deploying applications using cc2538-bsl script from Contiki

Started from Contiki 3.0, flashing CC2538-based board (e.g., SmartRF, Zolertia Firefly, OpenMote) can be done by using a Python-based tool called *cc2538-bsl*. It decreases the time of flashing to about 20 seconds.

1. Go to cc2538-bsl directory in /home/user/contiki/tools

2. Clone cc2538-bsl code from

   https://github.com/JelmerT/cc2538-bsl

3. For some boards, before downloading the code to the board, a certain button (e.g. SELECT button for OpenMote) should be pressed and hold to activate the backdoor for flashing.

4. For some boards supported by Contiki (i.e. the board-related tools and driver are available in /home/user/contiki/platform/ directory), the following command can be directly executed:

   make <binary file name>.upload TARGET=<board type>

   or for some board types like Zolertia:

   make <binary file name>.upload TARGET=<board type> BOARD=<board specific type>

   For example: *make border-router.upload TARGET=zoul BOARD=firefly*

   Note that in order to download the code to the board, the binary file should be available.

5. If the previous step is not applicable, a more general command can be used:

   sudo python cc2538-bsl.py -e -w -v -p <device port> <binary file name>.bin

   For example: *sudo python cc2538-bsl.py -e -w -v -p /dev/ttyUSB1 er-example-server.bin*

4 PC to board network communication

This is an example of how a border router application can be loaded onto one board and a CoAP server on another in order for a computer to be able to access the server through the border router. This page provides some example code on how to run the tunslip tool [9]. The steps below show how to set up and configure the link between the PC and board.

1. Load the following applications onto two boards according to the previous instructions.
   /home/user/contiki/examples/ipv6/rpl-border-router/border-router.c
   /home/user/contiki/examples/er-rest-example/er-example-server.c

2. Compile and setup the `tunslip` tool (both for IPv4 and IPv6).

   ```
   cd /home/user/contiki/tools
   make tunslip
   sudo ln -s /home/user/contiki/tools/tunslip /usr/bin/tunslip

   cd /home/user/contiki/tools
   gcc tunslip6.c -o tunslip6
   sudo ln -s /home/user/contiki/tools/tunslip6 /usr/bin/tunslip6
   ```

3. Create the tunnel to the board with the rpl-border-router application (USB port may vary):
   ```
   sudo tunslip6 -B 115200 -s /dev/ttyUSB1 aaaa::1/64
   ```

4. Look for the output "Server IPv6 addresses:"
   The border router will have prefix aaaa for its IPv6 address.

5. Try to ping the border router. Also try its webserver at
   http://[aaaa::xxxx:xxxx:xxxx:xxxx]/

6. There are several ways to detect the IP of the CoAP (er-rest-example) server board.

7. One is to connect to it using:
   ```
   sudo screen /dev/ttyUSB1 115200
   ```
   Note the line "Rime configured with address xx:xx:xx:xx:xx:xx:xx:xx".
   This has to be converted to an IPv6 address with the aaaa prefix.

   To convert it take the Rime address of form *04:0f:07:b2:00:12:4b:00* and concatenate it to:
   040f:07b2:0012:4b00.

   Now flip bit number 7 to get:
   060f:07b2:0012:4b00.

   Finally add the prefix aaaa:: to form:
   aaaa::060f:07b2:0012:4b00.

8. Another more simple method is to get it from the website of the border router.
   There it can be found under “Routes” in this format:

aaaa::xxxx:xxxx:xxxx:xxxx/128 (via fe80:: xxxx:xxxx:xxxx:xxxx)
The first IP (with prefix aaaa) is the address of the device connected to the border router.

9. Try to ping the board running the CoAP (er-rest-example) server.
   ```
   ping6 aaaa::xxxx:xxxx:xxxx:xxxx.
   ```
   Note that it may take time for a connection to be made between the boards.

10. For testing the CoAP server the Copper Firefox plugin can be useful [10]. It allows accessing CoAP resources directly from Firefox using *coap://* instead of *http://*.

11. Also test using command line tools such as Californium.
    ```
    java -jar cf-client-0.13.7.jar GET
    coap://[aaaa::xxxx:xxxx:xxxx:xxxx]/.well-known/core
    ```

4.1 Connecting to nodes simulated in Cooja

In the same way that a PC can be connected to a physical board using tunslip a tunnel can also be created into the Cooja simulation, providing access to the simulated nodes.

1. Create and start a simulation in Cooja that includes a node running the rpl-border-router application.

2. The command for accomplishing this is similar to creating a tunnel to a USB device:
   ```
   sudo tunslip6 -a 127.0.0.1 aaaa::1/64
   ```

3. After that follow the same steps as above (from step 4) to find out the addresses of the nodes in the simulation.

4. The node can now be reached using ping or any other application such as a CoAP client from outside the simulation using the IP addresses provided by the border router.

4.2 Note on IP addressing

It should be noted that the IP addressing differs depending on the setup. When connecting directly between two boards the IP prefix is *fe80* and when connecting through a border router the prefix is *aaaa*.

For instance say that the CoAP client (*er-example-client.c*) is running on one board and the CoAP server (*er-example-server.c*) is running on another and the boards are communicating directly with each other. The board with the server has Rime address *04:0f:07:b2:00:12:4b:00*. In this case the correct IP address to configure the client to connect to is *fe80::060f:07b2:0012:4b00*.

In the same case as above but having a board running as border router (*border-router.c*) and one as CoAP server the IP to be used is instead *aaaa::060f:07b2:0012:4b00*. So if a client application on a computer is used to connect to the server (through the border router) that is the IP to use.

When using Cooja these issues are simplified since nodes running in Cooja get a complete IPv6 address from the start and do not rely on the Rime addresses and conversion steps (see step 2.1.6).

4.3 IPv6 forwarding to reach boards from other devices

To enable communication with the boards from devices other than the directly connected computer the following steps should be applied. This is useful for testing services running on boards from remote devices. The setup is two computers (A & B) connected with a wired Ethernet link and computer A should have two boards connected to it using USB.

Computer connected to boards (A)

1. Connect two boards for testing to the computer (A). One running as border router and one with a server application.

2. Ensure the boards are correctly connected and that `tunslip6` is running.

3. Try to ping the addresses of the boards using `ping6`. Refer to earlier parts of this document for finding the addresses and how to accomplish the previous two steps.

4. Find the IPv6 address of the local network interface.
   ```
   /sbin/ifconfig
   ```

5. Ensure that you can ping the other computer (B).

6. Enable IPv6 forwarding
   ```
   sudo sysctl -w net.ipv6.conf.all.forwarding=1
   ```

Computer acting as client (B)

1. Ensure that you can ping the other computer (A).

2. Add a route to the board network using **one** of these commands:
   ```
   /sbin/ip -6 route add aaaa::/64 via $ADDRESS
   /sbin/route -A inet6 add aaaa::/64 gw $ADDRESS
   ```

   Where $ADDRESS is the IPv6 address of the other device (A).

3. The traffic will now be routed correctly and accessing the boards should work.

4. Now try to ping the border router and server board addresses.

5. If pinging the boards work try directly accessing the service running on the server board (for instance CoAP) using the proper application.

5 Performance evaluation using Contiki

The following sections describe methods for performance evaluation of Contiki applications.

5.1 Memory measurement

The following command can be used for memory measurement:

size <binary file name>.elf

Note that <binary file name>.elf and <binary file name>.<board type> are basically the same. Therefore, the memory measurement results for these two files will be the same.

5.2 Energy measurement

A Contiki tools called Energest (also known as *powertrace* or *compower*) can be used for energy measurement. Energest counts the number of ticks spent on certain part of code. There are 4 types of operation can be measured by Energest:

1. CPU → measurement how many ticks spent on CPU usage
2. LPM → measurement how many ticks spent on Low Power mode/sleeping mode
3. Transmit → measurement how many ticks spent for transmitting data
4. Listen → measurement how many ticks spent for listening to communication channel. Note that *LISTEN* result can be misleading as the radio will always listen to the channel as long as the device is not in *low power mode*.

Energy measurement can be performed:

1. Activate Energest mode by setting ENERGEST_CONF_ON to 1 in

   /home/user/contiki/platform/<board type>/<configuration file name>

   For example: /home/user/contiki/platform/cc2538dk/contiki-conf.h

   Find similar line and change it to the following line:

   #define ENERGEST_CONF_ON 1

   Note that sometimes there are other preprocessors need to be change as well (e.g. *WITH_COMPOWER*), try to check the existence of these kind of preprocessors if Energest is still not activated after changing the flag of *ENERGEST_CONF_ON*.

2. Put the following lines in the code to be measured:

   energest_init();

ENERGEST_ON(ENERGEST_TYPE_CPU);   // activate for any operation that needs to be measured

ENERGEST_ON(ENERGEST_TYPE_LPM);

ENERGEST_ON(ENERGEST_TYPE_TRANSMIT);

ENERGEST_ON(ENERGEST_TYPE_LISTEN);

…

powertrace_print("#P Start >");

<part of code that is being measured>

powertrace_print("#P End >");   // the measurement result

3. Get the number of tick spent for each operation from the measurement result. The measurement results printed by Energest looks like:

   #P End > x_1 *P* x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 $\boldsymbol{x_{10}}$ $\boldsymbol{x_{11}}$ $\boldsymbol{x_{12}}$ $\boldsymbol{x_{13}}$ x_{14} x_{15} *(…)*

   For example: *#P End > 2083 P 0.18 5 26973 487011 320 264730 0 0* ***11747 0 0 11747*** *0 0 (radio 51.56% / 100.00% tx 0.06% / 0.00% listen 51.50% / 100.00%)*

   Take only the value positioned in the same places as the bold text (x_{10} to x_{13}) as the measurement results for CPU, LPM, TRANSMIT and LISTEN respectively.

4. To convert the result in ticks to energy, we need to know the voltage and current spent for certain operation. These values can be found in the datasheet.

   $\text{Energy}_{\text{CPU}}$ = $\text{Voltage}_{\text{CPU}}$ * $\text{Current}_{\text{CPU}}$ * <# of ticks from the measurement> / <# of ticks in one second>

$$Energy_{CPU} = \frac{Voltage_{CPU} \times Current_{CPU} \times Number\ of\ ticks\ from\ measurement_{CPU}}{Number\ of\ ticks\ in\ one\ second}$$

$$Energy_{LPM} = \frac{Voltage_{LPM} \times Current_{LPM} \times Number\ of\ ticks\ from\ measurement_{LPM}}{Number\ of\ ticks\ in\ one\ second}$$

$$Energy_{TX} = \frac{Voltage_{TX} \times Current_{TX} \times Number\ of\ ticks\ from\ measurement_{TX}}{Number\ of\ ticks\ in\ one\ second}$$

$$Energy_{LIST} = \frac{Voltage_{LIST} \times Current_{LIST} \times Number\ of\ ticks\ from\ measurement_{LIST}}{Number\ of\ ticks\ in\ one\ second}$$

Note: Print *CLOCK_SECOND* to get the value of *number per ticks in 1 second*.

6 References

[1] http://contiki-os.org/start
[2] http://github.com/contiki-os/contiki/tree/master/platform/cc2538dk
[3] http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_apps#Test_REST_example
[4] http://embedded-funk.net/compiling-contiki-2-7-demo-for-cc2538dk-on-ms-windows/
[5] http://processors.wiki.ti.com/index.php/UniFlash_Quick_Start_Guide
[6] http://e2e.ti.com/support/development_tools/code_composer_studio/f/81/t/239320.aspx
[7] http://processors.wiki.ti.com/index.php/Category:CCS_UniFlash
[8] http://processors.wiki.ti.com/index.php/Download_CCS
[9] http://anrg.usc.edu/contiki/index.php/RPL_Border_Router#Tunslip_utility
[10] http://addons.mozilla.org/en-US/firefox/addon/copper-270430/