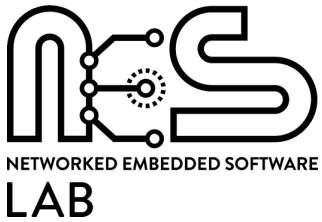


POLITECNICO
MILANO 1863

RI.
SE

Contiki-NG Cheat Sheet v.0.5

Luca Mottola
luca.mottola@polimi.it

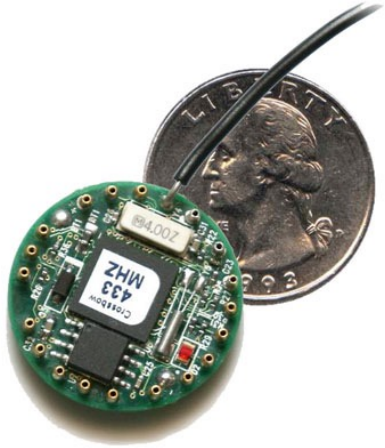
Virtual Machine

- Find the VM image at **`shorturl.at/XiLxY`**
- Recommended hypervisor: VirtualBox with Extension Pack
  - Download from: **`virtualbox.org/wiki/Downloads`**
  - The VM image should be compatible also with VMWare Fusion, VMWare Workstation, Parallels, …
  - Instructions are platform-dependent
- Use File ➔ Import Appliance…
  - Customize settings only if needed and you know what you're doing

Virtual Machine: Apple Silicon

- Although possible, this is **not recommended**
  - As it runs in emulation, **it is incredibly slow**
- You need an emulation-capable hypervisor
  - One example is UTM: **`https://mac.getutm.app`**

VM: Notes



- Why a VM?
  - Only reliable solution if you want to use **real** IoT embedded hardware
- Why 32-bit?
  - COOJA and MSPSim provide cycle-accurate emulation of MSP430* MCUs through JNI, more on this later…
- The VM is **customized for this course**
  - Do not update
- There is **no support for**
  - Newer MSP430 MCUs with large memories
  - JN and NRF platforms
  - JTAG for CC2538
- Make sure you can access the Internet from the VM
  - Easy solution: use NAT networking
- Make sure USB compatibility is limited to 2.0 and the Bluetooth interface is **not** shared

VM: Account Info

- Username: **`user`**, password: **`user`**
- Sudo password: **`user`**

VM: My Own Setup

- Simply a suggestion!
- Use the VM only for compiling
- Keep source code on your host
  - So you can edit, navigate, and search the code using your OS
- Share your host folder with the VM
- SSH into the VM from the host to start compiling, flashing devices, …
  - On VirtualBox, this may require using Bridged networking
- Requires VirtualBox Guest Additions and openssh-server to be installed on the VM

Your Own Linux Machine

- If you like, install the toolchain on your host Linux machine
  - Instructions at: **`github.com/contiki-ng/contiki-ng/wiki/Toolchain-installation`**
  - Do not try and do that on Windows or MacOS: compatibility is not guaranteed
- USB port numbers in the following instructions refer to the VM, they might change on a different installation

Contiki-NG Sources

- We use a private fork of Contiki-NG, customized for this course
  - A few specific bug fixes
- Pull the sources with: **`git clone https://bitbucket.org/neslabpolimi/contiki-ng-nsds-22.git`**
- Pull the submodules **inside** the local copy with **`git submodule update --init --recursive`**

Native Platform

- To compile: **`make TARGET=native`**
- To execute: **`./<projectfile>.native`**

COOJA

- Lives in `tools/cooja`
- To start: `ant run`

RPL Border Router in COOJA

- The border router may also run inside Cooja
- You can create a simulated setting and bridge that to the real IPv6 network outside COOJA
- Compile and deploy **`border-router.c`** inside **`examples/rpl-border-router`** as a native COOJA mote
  - Right click on the mote in the simulation
  - Open “Mote tools”
  - Start the serial console server
- Start the simulation with 100% (real time) speed limit
- To start bridging: execute **`make TARGET=cooja connect-router-cooja`** from the command line

Mosquitto

- The VM has **`mosquitto`** installed and running as a service from boot
- It waits on all available IPs, including IPv6 addresses
- If you want your messages to be routed to a publicly-available broker, add the following to **`/etc/mosquitto/mosquitto.conf`** and restart the broker!

```
connection bridge-01
address mqtt.neslab.it:3200
topic # out 0
topic # in 0
```

Mosquitto Clients

- To debug your MQTT application, you can use the command-line mosquitto clients on your VM
- To subscribe to a topic:
  `mosquitto_sub -h <hostname> -p <port> -t <topic>`
- To publish on a topic:
  `mosquitto_pub -h <hostname> -p <port> -t <topic> -m "Message"`