# Advanced IoT Firmware Engineering

## With Thingsquare and Contiki

thingsquare

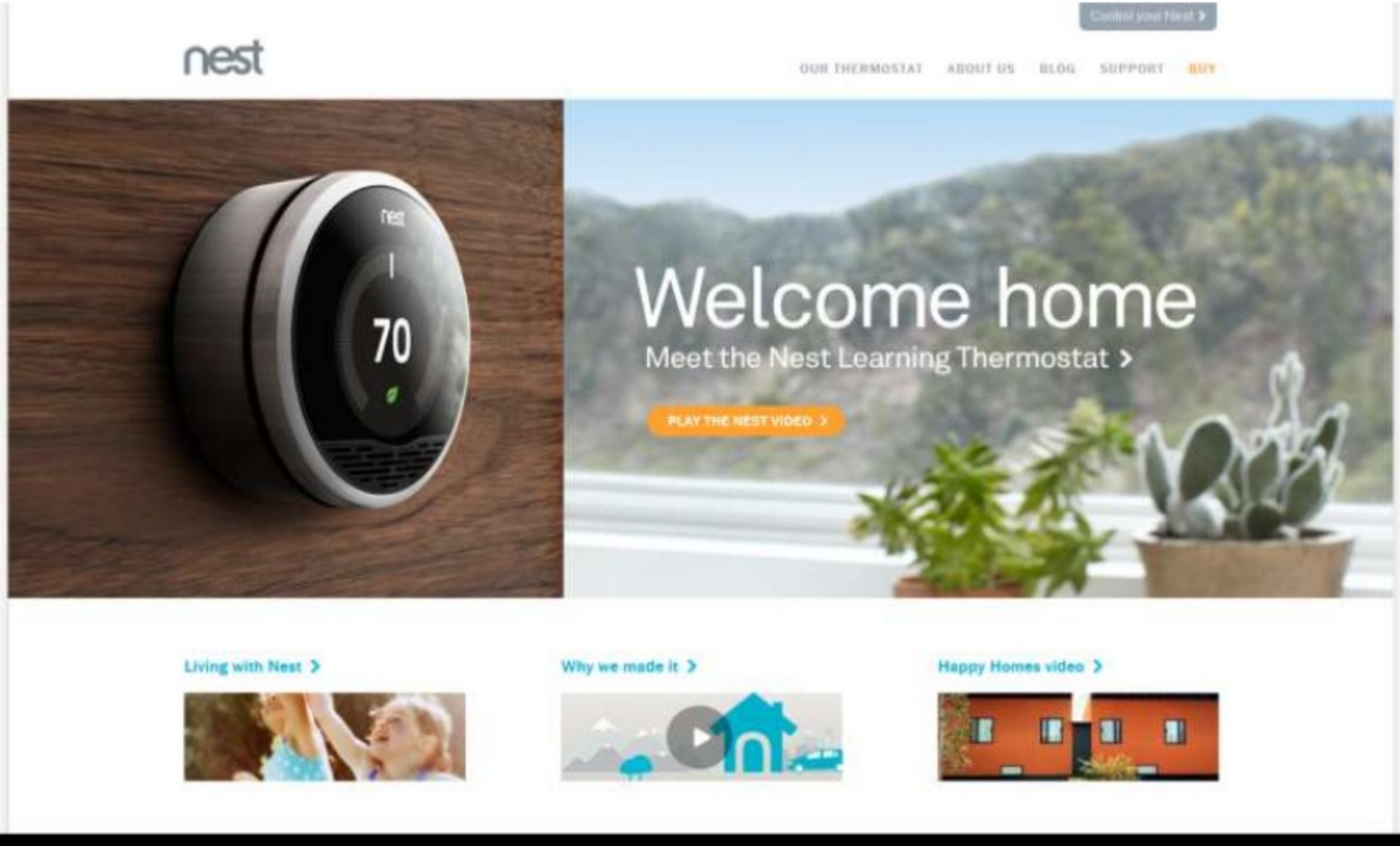
# Overview

- This is the advanced course
- Hard core firmware source code drill-down
- We will look deep into the system
- We will look into network debugging and simulation

# Quick recap

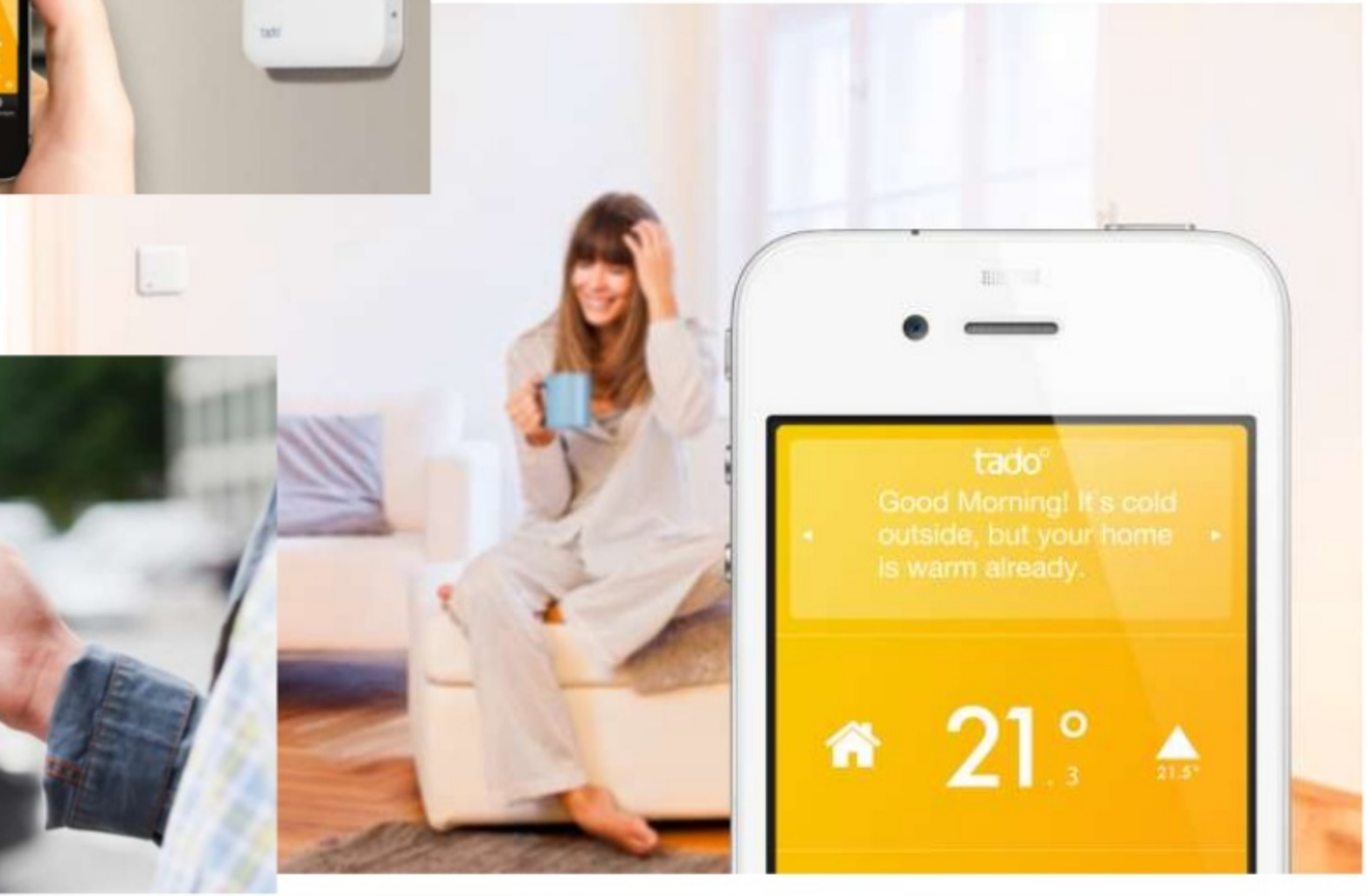
# Thingsquare

- Build connected systems – leverage the Internet of Things
- Founded in 2012
- Creators of the open source Contiki OS
- Launching in 2014
  - Thingsquare cloud backend
  - Online development environment


nest
OUR THERMOSTAT
ABOUT US
BLOG
SUPPORT
BUY
70
Welcome home
Meet the Nest Learning Thermostat >
PLAY THE NEST VIDEO >
Living with Nest >
Why we made it >
Happy Homes video >

## Works with your Nest Thermostat.


21°


tado°
Good Morning! It's cold outside, but your home is warm already.
21°
3
21.5°

thingsquare

KICKSTARTER Discover great projects Start your project Search projects BLOG HELP SIGN UP LOG IN

LIFX


Control the lighting in your house with an app.

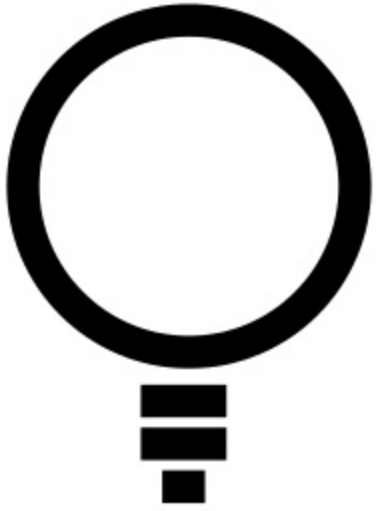
PRE-ORDER NOW

~~SHIPPING JUNE 2013 (SOLD OUT)~~
~~SHIPPING SEPTEMBER 2013 (SOLD OUT)~~
SHIPPING & LIMITED RETAIL: Q1 2014

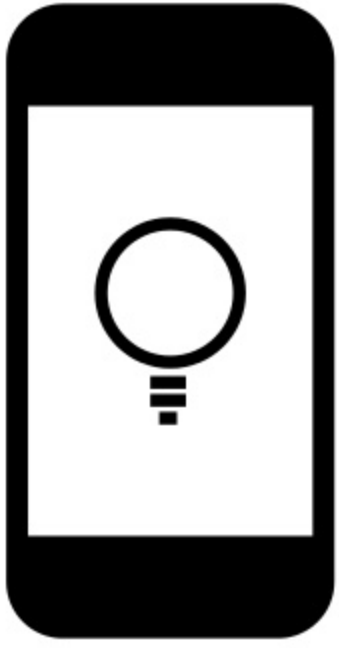

streetlinenetworks.com


zolertia.com

# What is the common denominator?







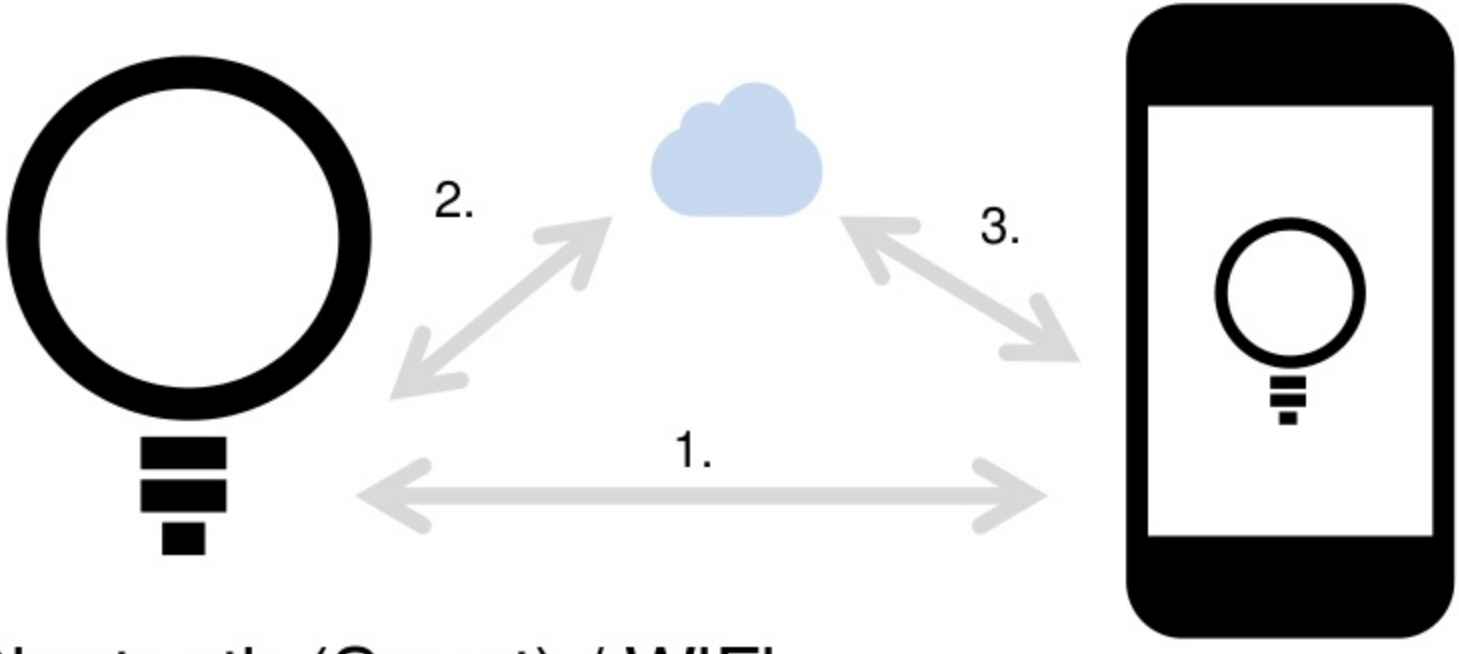
# IPv6 / 6lowpan

- Much more low power than WiFi
- Automatic meshing
- Very long range
  - Sub-GHz communication
- Drawbacks
  - Lack of infrastructure in homes

# IPv6 / 6lowpan contd.

- IPv6 addresses are large
  - 6lowpan compresses headers
- Automatic meshing: RPL
  - Automatically form large (1000+) node networks
  - Self-suppression of control trafic

# Putting it together



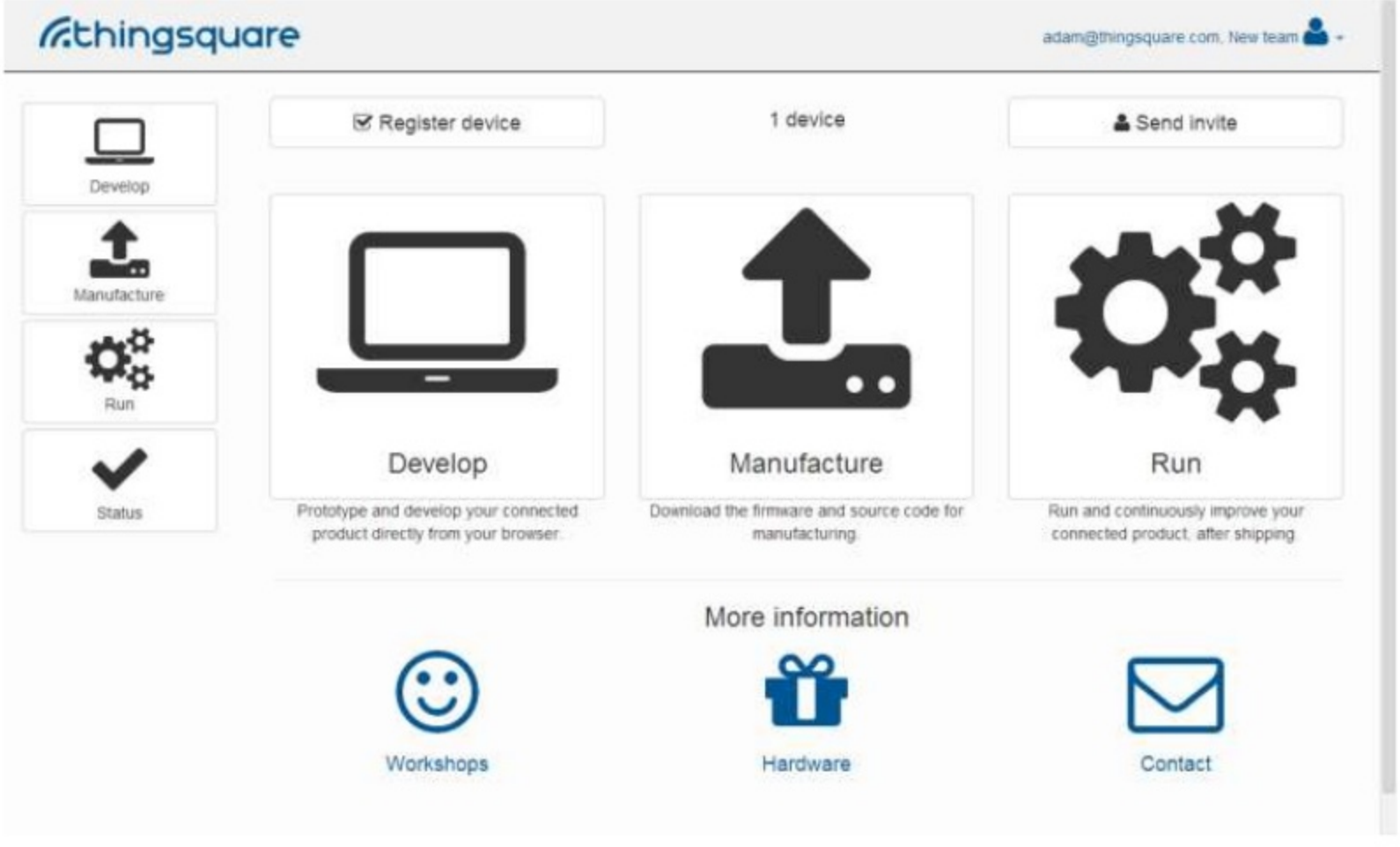

1. Bluetooth (Smart) / WiFi
2. WiFi / 6lowpan or through a smart hub
3. RESTful API

# Big Red Internet Button

+

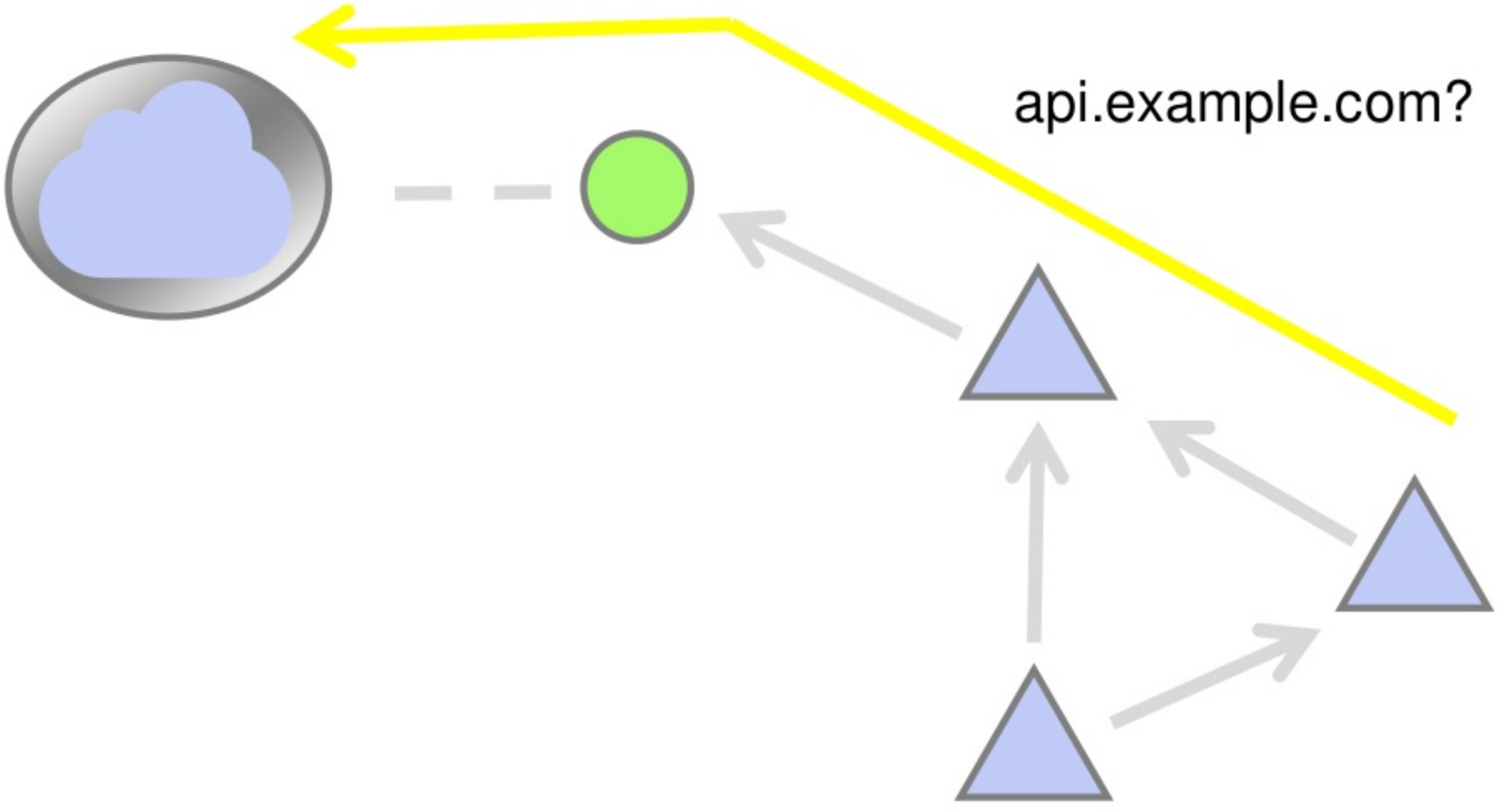

= IoT!

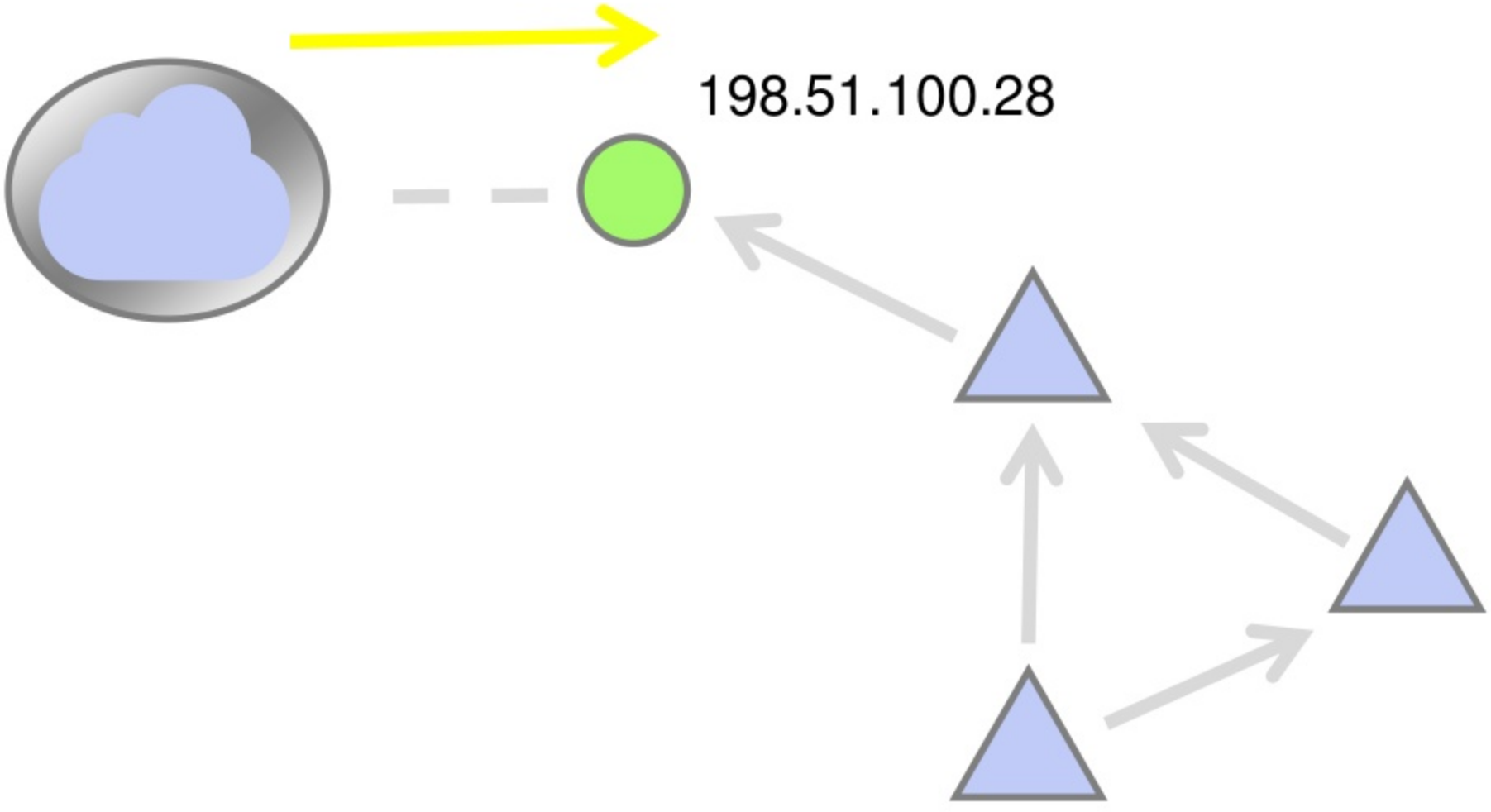
# The Thingsquare cloud

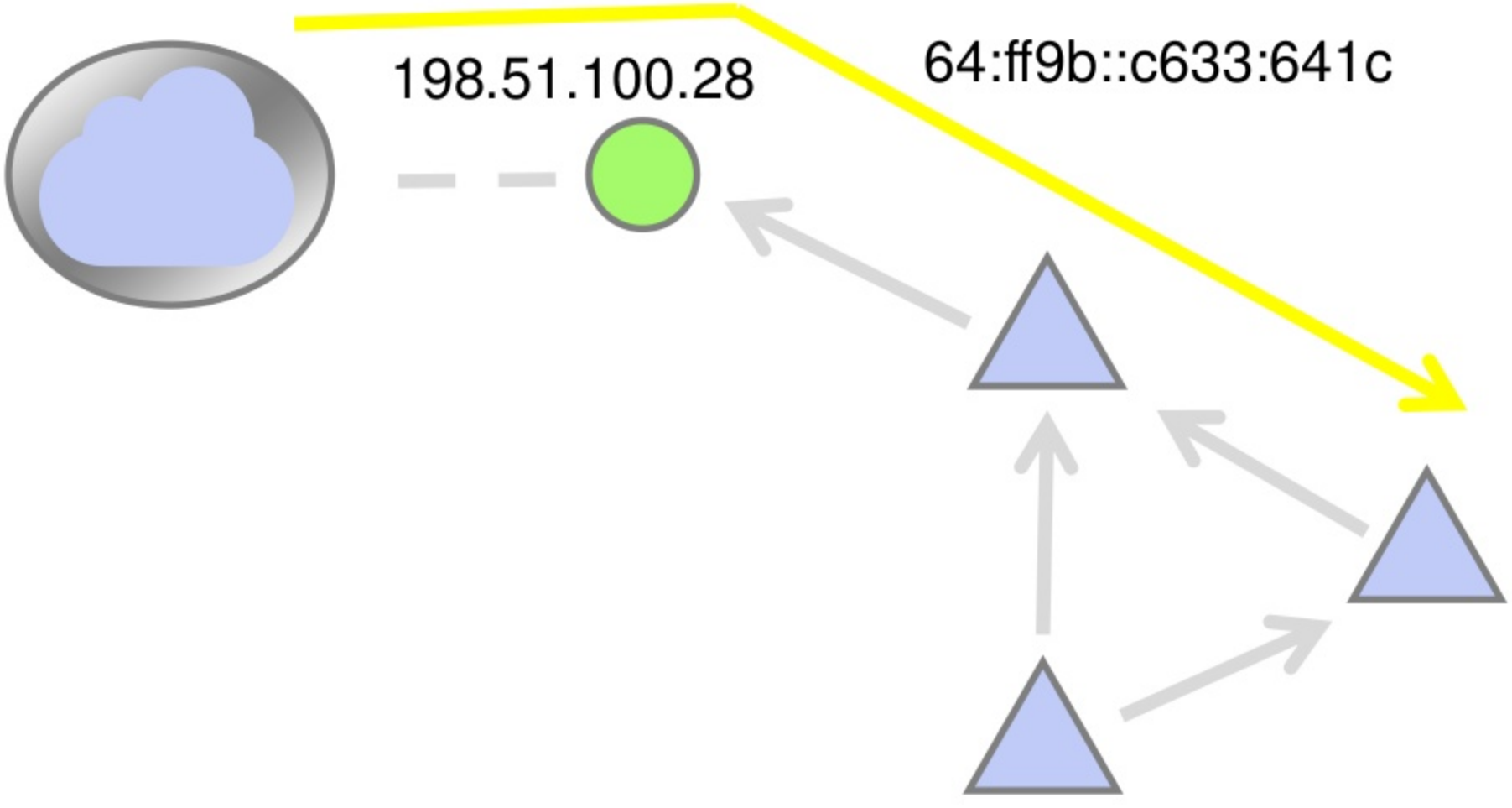


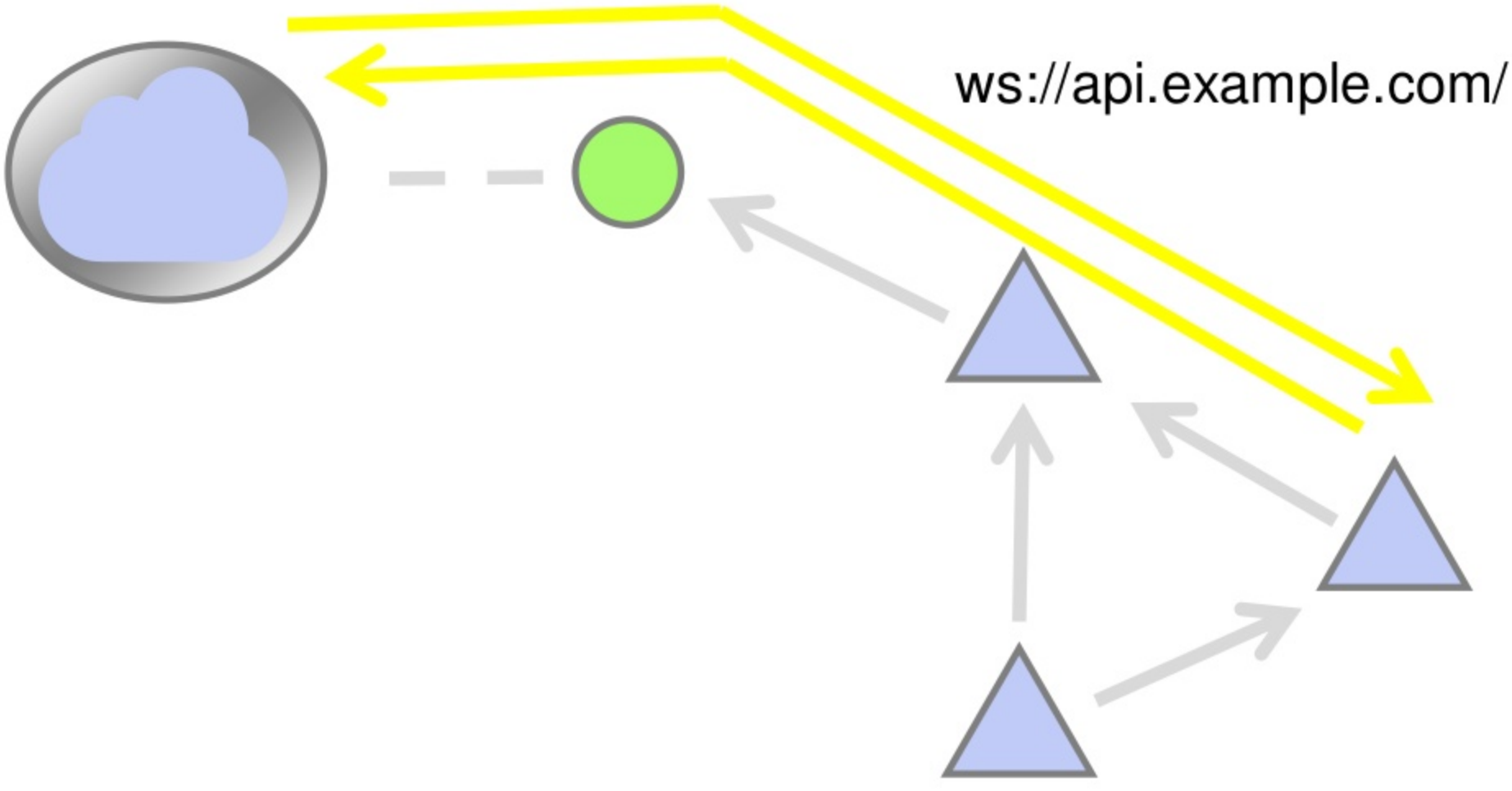
# What we just did

- Did an HTTP POST directly from the chip
- Posted data via a webhook to a cloud service


api.example.com?


198.51.100.28


198.51.100.28
64:ff9b::c633:641c


ws://api.example.com/

# What we will do today and tomorrow

- Drill down into the firmware
  - Network protocols
  - Radio drivers
  - Duty cycling
  - Timers
  - Boot-up code
  - Debugging
  - Cooja

# You should already know

- What Contiki is
- What IPv6 is
- What RPL is
- What CSMA means
- What an interrupt is

# What you will know

- How Contiki works, under the hood
- How to write a radio device driver
  - And make it interrupt-safe
- How the serial stack works
- How Contiki boots
  - How a firmware upgrade on the CC2538 works
- How to port Contiki to new platforms
- What knobs to turn in the Contiki netstack
- How to make full use of the Contiki RPL stack
- How to sniff and debug the network

# Brand new: Contiki 3.x

- We will use some of the APIs and mechanisms that will go into Contiki 3.x
  - But aren't in there yet
- We'll use Thingsquare's Contiki internal version

# IoT software challenges

- Severe limitations in every dimension
  - Memory
  - Processing power
  - Communication bandwidth
  - Energy, power consumption
- The number of different platforms is large
  - Different microcontrollers, different radios, different compilers

# IoT system challenges

- Communication-bound, distributed systems
  - Exceptionally low visibility
    - Hard to understand what is going on
  - Application development difficult
  - Debugging difficult
- Large-scale
- Wireless communication

# Toolchains

# A toolchain

- Editor
- Compiler
- Standard libraries
- Linker
- Debugger
- Flash programmer

# Contiki tool chains

- IAR, TI CCS, Atmel Studio, or other embedded IDEs
  - Doable, but not ideal
- Cygwin under Windows
  - Installation can be a hassle
  - Compilation is (somewhat) slow
- Native under Mac OS
  - Installation may be hassle
  - Lack of flash programmer tools may be prohibitive
- Native under Linux
  - Drivers for flash programmers may be problematic
- Instant Contiki

# The Contiki build system

- A set of Makefiles
- Makefiles set C #defines
  - These will have to be replicated if compiling via an IDE
  - Using make is usually best (and easiest)

# Install Instant Contiki

- VMWare player (Windows, Linux)
- Virtualbox (Mac OS X)
- InstantContiki2.7.zip (from USB stick)
- Unzip InstantContiki2.7.zip in separate directory
- Copy thingsquare-firmware-course-2014-02-05-73c67ea.zip into Instant Contiki and unzip it

# Update contiki

- cd contiki
- git pull

# Code walkthrough

(In another window)

# Contiki programming principles

# Software development

- Write your program in a separate C file
- Cross-compile the full system binary
- Upload to target
- Debug via serial printouts, LEDs
  - On occasion: gdb / IAR debugger
- Use Cooja to simulate

# The project directory

- Create a new directory
- Copy Makefile from examples/hello-world
- Modify Makefile
- Create new C code file

# Contiki firmware images

- In a terminal, go to the project directory

```
make TARGET=platform file
```

- This will build the full source code tree for your project. Eg,

```
make TARGET=openmote hello-world
```

# Uploading the firmware

- Some platforms have a convenient way to upload the firmware

```
make TARGET=sky hello-world.upload
```

- Some platforms are significantly trickier

# Save target

- If you are using the same target a lot,

```
make TARGET=cc2538dk savetarget
```

- Then, simply

```
make blink.upload
```

# Running as the native target

- Sometimes using the native target is useful

```
make TARGET=native hello-world
```

- Run the program

```
./hello-world.native
```

# Other commands

- Connect to serial port

```
make TARGET=exp5438 login
make TARGET=exp5438 COMPORT=COM12 login
```

- Clean out previous compilation

```
make TARGET=exp5438 clean
```

- List available commands

```
make TARGET=exp5438 help
```

# Hands on 1: blink

# Blink

- Create project directory, copy Makefile and project-conf.h from examples/hello-world
- Open blink.c with text editor of choice (emacs, vi, gedit, …)
- Log in to demo.thsq.io, click on the Develop button
- Copy the contents of blink.c from the browser into blink.c
- Compile, in the terminal:
  - **`make TARGET=thsq-cc2538dk blink.bin`**

# Upload

- Copy blink.bin to shared folder
- Use TI SmartRF Flash Programmer 2 to burn to flash
- Watch the LEDs blink

# Hands on 2: UDP broadcast

# udp-broadcast.c

- Copy udp-broadcast.c from demo.thsq.io into udp-broadcast.c
- `make TARGET=thsq-cc2538dk udp-broadcast.bin`

# More Contiki principles

# Naming

- Function names prefixed by their module name
  - Example: **`memb_alloc()`**
- Configuration parameters: MODULE_CONF_NAME
  - Example: QUEUEBUF_CONF_NUM

# Caller allocation

- In general, the caller always allocates memory
  - Pass pointer to struct
- Example:
  - HTTP socket
  - See big-red-button.c
- Exception: low-level uIP TCP connections

# Editing core files

- Sometimes you need edit core files
- Don’t edit core files in the Contiki tree
  - Makes it difficult to update to new versions
- Instead, copy the core file to your project directory and edit there
- Run **`make clean`** before compiling next time

# Contiki's complexity

- Contiki may be complex
- Sometimes this is due to the problems being complex
- Sometimes this is just because of random historical reasons

# More



http://thingsquare.com