Freie Universität Berlin

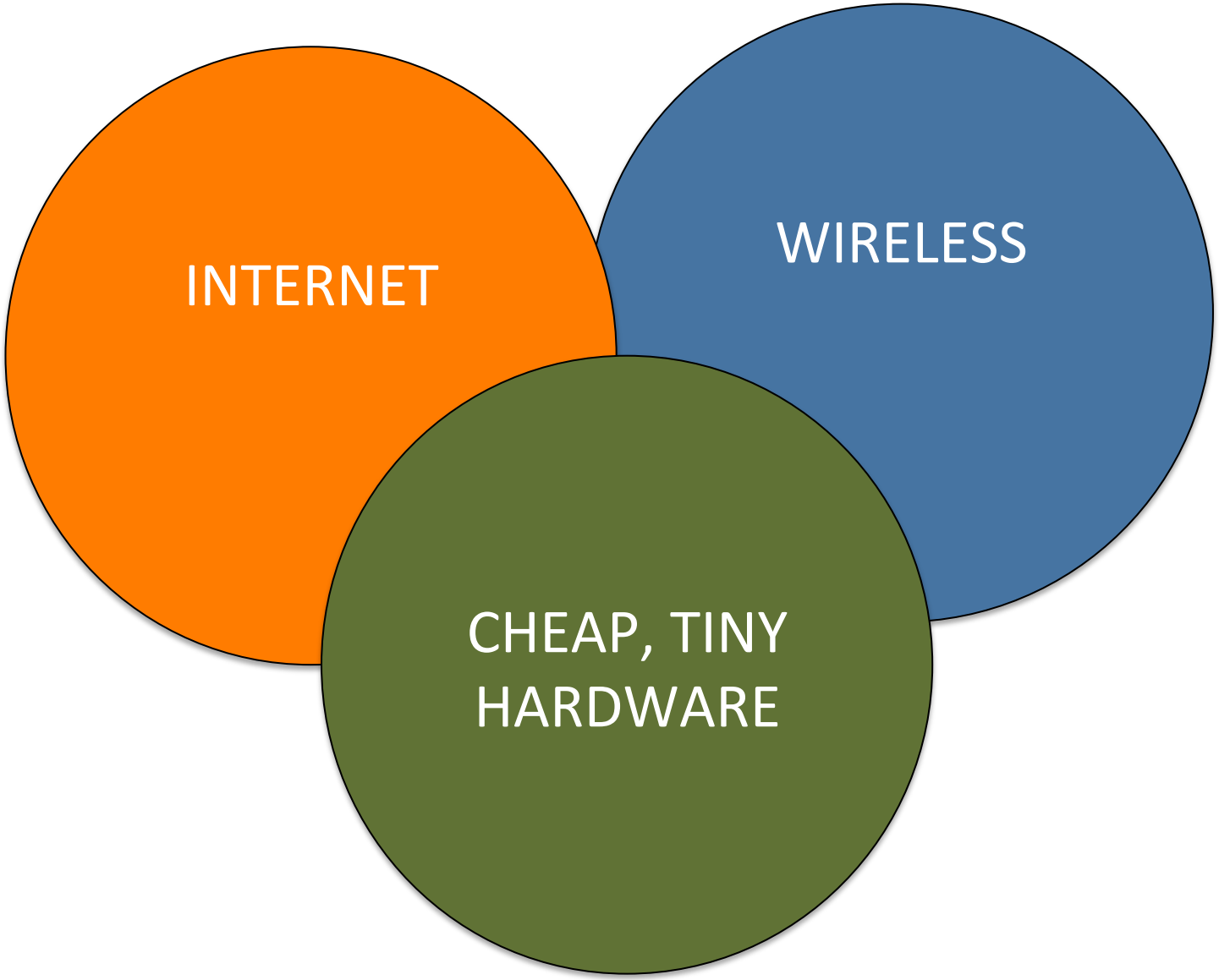
Telematics
Software Project

# Introduction

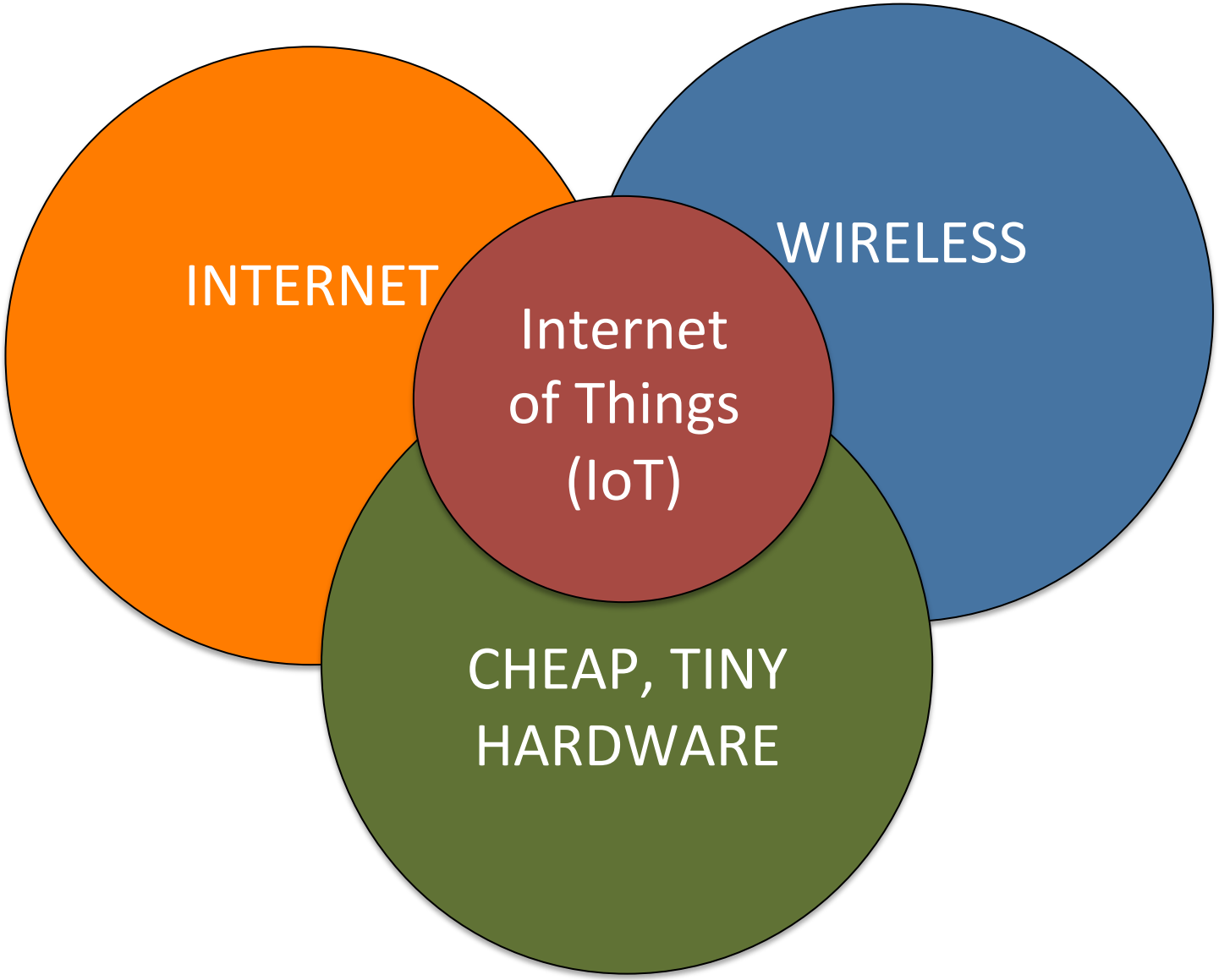
Dr. habil. Emmanuel Baccelli
INRIA / Freie Universität Berlin

Institute of Computer Science
Computer Systems and Telematics (CST)

# The Big Picture: a Giant Collision

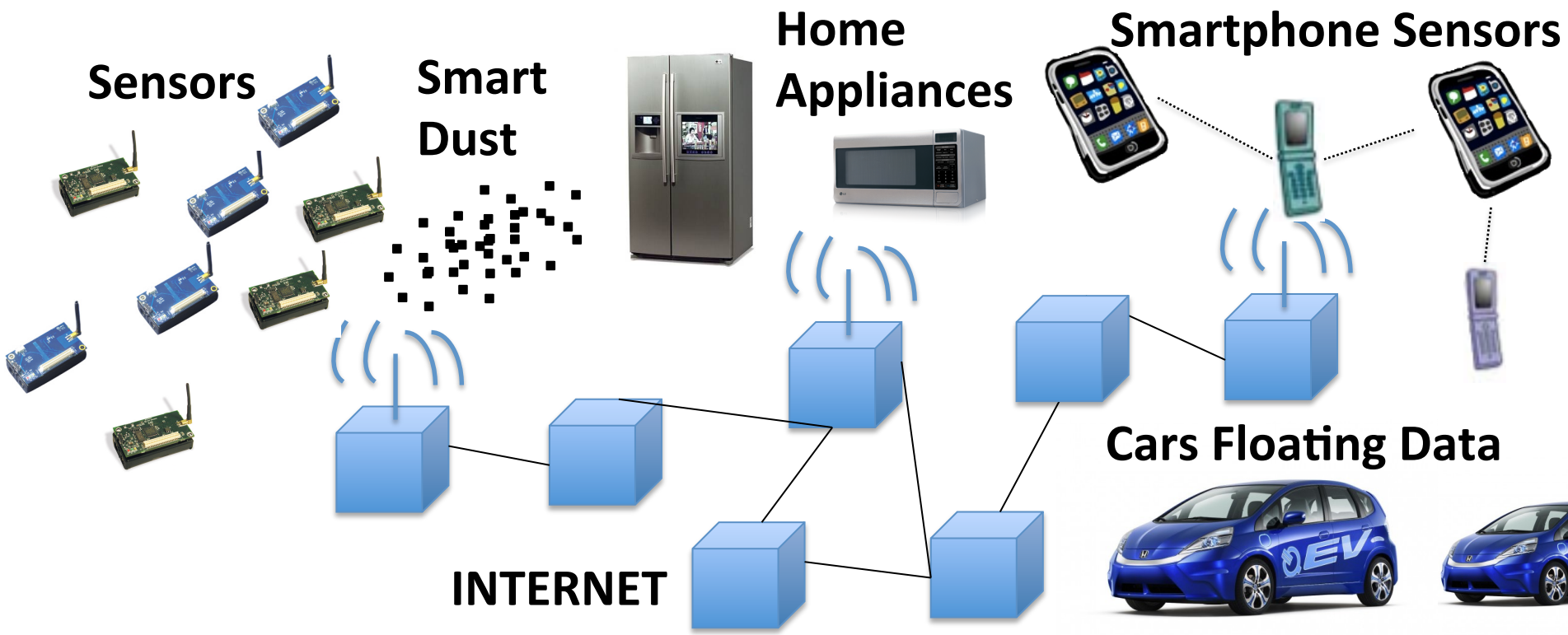


# The Big Picture: a Giant Collision



# The Internet of Things (IoT)

The buzz word for an upcoming avalanche of connected devices beyond hosts and routers



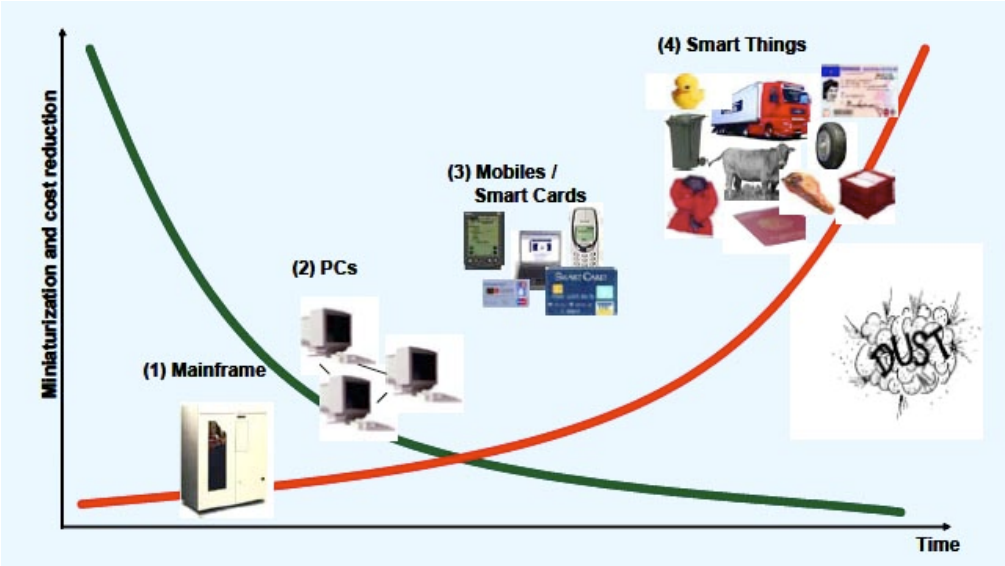
# Expected Impact of the IoT

- A new world of interconnected hardware
  - denser environment of communicating devices
  - more heterogeneity among communicating devices
  - more low-power (multi-radio) communications

# Expected Impact of the IoT

- A new world at the application layer
  - thousands of innovative applications
  - personalised environment automation
  - cloud interaction & spontaneous wireless networks

- Application scenarios:
  - Smart Metering
  - Home/Building Automation
  - Smart City
  - Smart Grid
  - ehealth
  - Logistics
  - …



# Expected Impact of the IoT

- A new world in terms of user experience
  - Internet taking a new shape
  - enabling cyber-physical reality

➔ Our interface to the Internet might no longer be predominently a screen/keyboard/mouse

# AGENDA

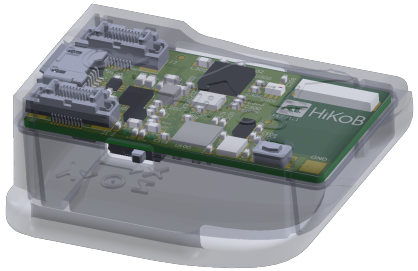
- Characteristics of the Internet of Things
  - Hardware aspects
  - Software platform aspects
  - Network stack aspects
  - Application aspects
- RIOT: a friendly software platform for the IoT
- Course Format & Requirements
- Proposed software projects topics

# IoT: from the hardware perspective

- The IoT is already here
  - Tiny, cheap & exciting new devices pop up daily

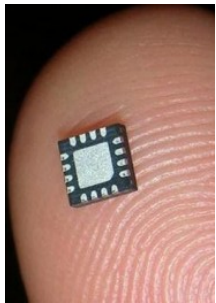
Arduino Uno board
8-bit Atmel AVR


TI eZ430 Chronos watch
16bit MSP 430




HiKoB Fox board
32bit ARM M3

Smart Dust



# IoT Hardware Characteristics

- Heterogeneity: from 8-16 bits micro-controllers, lightweight 32 bit processors
  - Slow CPU, often no FPU

- Memory constraints: **less than first computers on the ARPANET!**
  - ~8-100kb RAM
  - ~32-256kb ROM
  - Typically no MMU

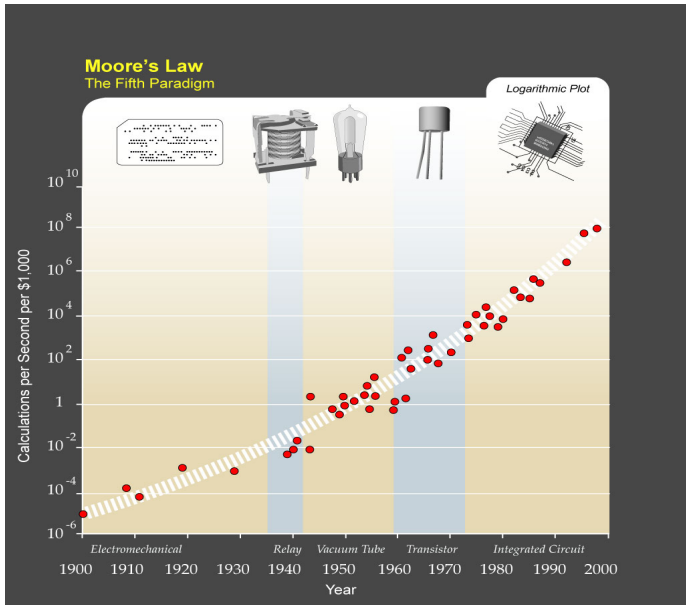
# IoT Hardware Characteristics

- Energy constraints: most devices are battery powered and must remain active for years

- Size and cost constraints: small is beautiful, cheap is necessary

# Constrained Future?

- Moore's law might not apply to IoT
  - Target is different deployment (billions of devices, millions at once)
  - Business environment where very cent counts
    - e.g. : STM32F3 (Cortex-M4)
    - 24k RAM, 128k flash : 6,78€ (digikey)
    - 40k RAM, 256k flash : 9,10€
    - ~2.3 Million for 1 Million devices...
  - IoT hardware «progress» observed since 10 years doesn't match Moore's law
- So the focus is expected to stay on contrained devices



# IoT Hardware: Basic Components

- Typically, there are three main components:
  - Component 1**: the board**
    - e.g., Arduino Uno, STM discovery board, HikoB Fox…
  - Component 2: **the radio**
    - e.g., xBee shield, cc2420 (IEEE 802.15.4), nRF8001 (Bluetooth Low energy), Dash7 radios, Zwave
    - Typicically not Wifi (802.11ah, or 802.11b in the future?)
  - Component 3: **the sensor**(s)
    - e.g., temperature + humidity (SHT11), IMU (LSM9DS0), accelerometer (LIS3DSH, SMB380)

# Focus of the project

- Focus is of course not really on hardware
- Focus is on cross-platform IoT software

# AGENDA

- Characteristics of the Internet of Things
  - Hardware aspects
  - Software platform aspects
  - Network stack aspects
  - Application aspects
- RIOT: a friendly software platform for the IoT
- Course Format & Requirements
- Proposed software projects topics

# IoT Software Platforms

- In order to confortably program the boards, a software platform (some kind of OS) is needed
- Currently, most deployed IoT hardware is powered by the vendor's rudimentary and proprietary OS (hardware-specific)
- This is inefficient for IoT application developers, so there is a push towards open source, cross-platform OS providing API/SDK

# Legacy Operating Systems

- Typical Real-Time Operating Systems:
  - FreeRTOS
  - QNX
  - RTLinux
- Problem: not designed for
  - energy-efficiency
  - constrained nodes/networks
- Traditional operating systems for WSN:
  - Contiki
  - TinyOS
- Problem: not a good fit because
  - Event-driven design
  - Single-threaded
  - No real-time properties
  - Specialized programming language

# Hello World in TinyOS

```
///////////
#include <stdio.h>
#include <stdlib.h>

module HelloworldM {
provides {
    interface Hello;
}

}
implementation {
    command void
Hello.sayhello() {
        printf("hello world!");
    }
}
```

the hello interface: Hello.nc:

```
///////////
interface Hello{
  command void sayhello();
}
///////////
```

# Hello World in Contiki

```
#include "contiki.h"
#include <stdio.h> /* For printf() */
/*---------------------------------------------------------------------*/
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*---------------------------------------------------------------------*/
PROCESS_THREAD(hello_world_process, ev, data)
{
PROCESS_BEGIN();

printf("Hello, world\n");
PROCESS_END();
}
/*---------------------------------------------------------------------*/
```

# How we wish Hello World to be

```
#include <stdio.h>

int main(void)
{
  printf("Hello World!\n");

  return 0;
}
```

# AGENDA

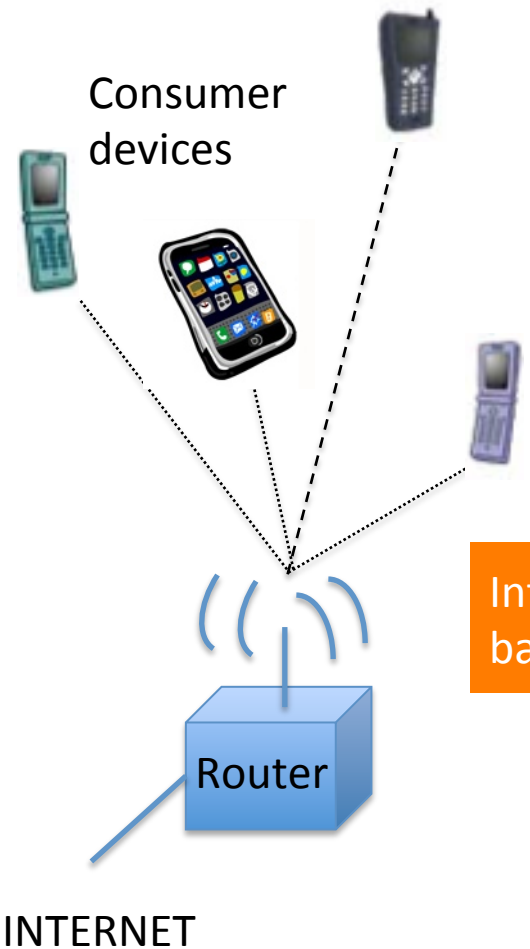
- Characteristics of the Internet of Things
  - Hardware aspects
  - Software platform aspects
  - Network stack aspects
  - Application aspects
- RIOT: a friendly software platform for the IoT
- Course Format & Requirements
- Proposed software projects topics

# IoT Networking

- Need to support several link layer technologies
  - Several radio technologies (802.15.4, Bluetooth…)
  - Some wired technologies too (PLC)

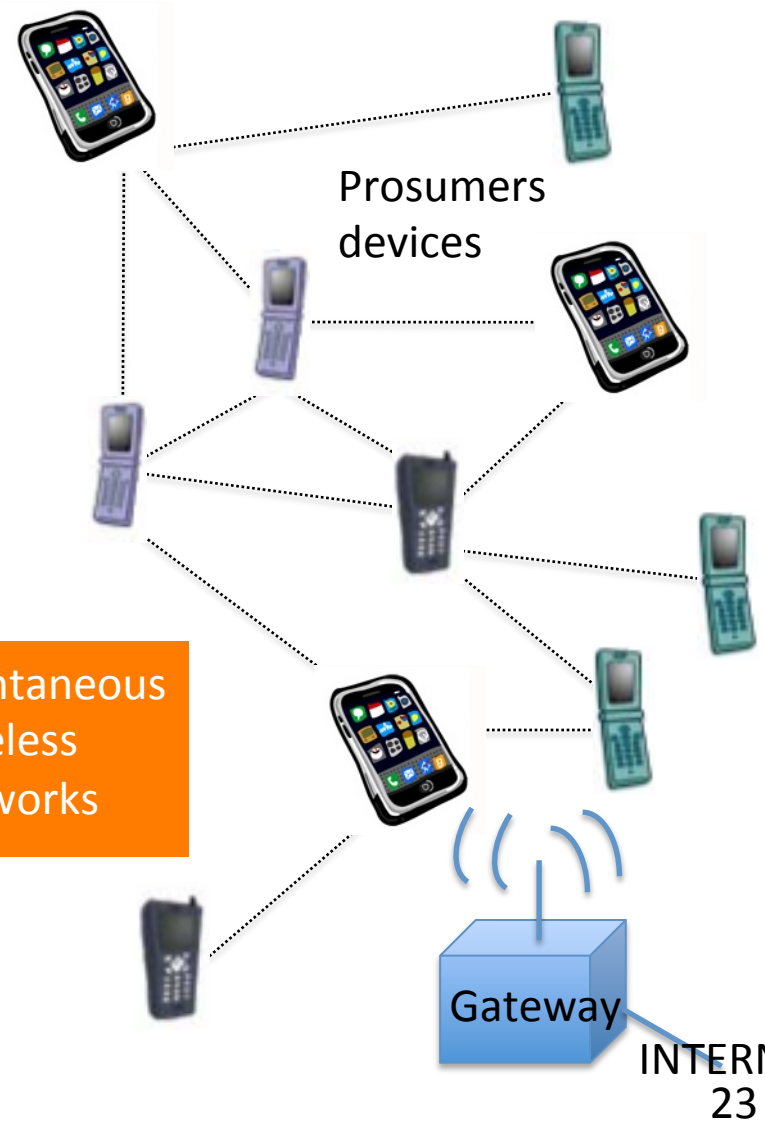
- Need to support several network architectures
  - Infrastructure-based networking
    - one hop to wireless access point
  - Spontaneous wireless networking
    - self-organized wireless networking, multi-hop, with or without help from infrastructure.

# Spontaneous Wireless Networks




Infrastructure-based Networks

Spontaneous Wireless Networks



# Various Network Aspects…

- …at various layers of the protocol stack

| |
|---|
| APPLICATION LAYER |
| TRANSPORT LAYER |
| NETWORK LAYER |
| LINK LAYER |
| PHYSICAL LAYER |

# MAC/PHY Layer Aspects

- Shared medium (hidden/exposed terminal problem)
- Duty cycling for energy efficiency
- Distributed transmission scheduling and channel assignment
- TDMA or CDMA for reliability and deterministic characteristics

| |
|---|
| APPLICATION LAYER |
| TRANSPORT LAYER |
| NETWORK LAYER |
| **LINK LAYER** |
| **PHYSICAL LAYER** |

# Network Layer Aspects

- Usual solution for multiple Link Layer technology support: IP
- Billions of IoT devices are expected, therefore enormous IP address needs: typical solution is to use IPv6
- Problem: IPv6 does not fit as is

| |
|---|
| APPLICATION LAYER |
| TRANSPORT LAYER |
| **NETWORK LAYER** |
| LINK LAYER |
| PHYSICAL LAYER |

# Network Layer Challenges

- IPv6 header compression
  - Constraint: must fit 10x smaller frame size!! (6LoWPAN)
- IPv6 stateless autoconfiguration for IoT devices
  - Constraint: duplicate address detection in multi-hop wireless scenarios!!
- Routing protocols for spontaneous wireless networking in the IoT
  - Constraint: not enough memory!!

| |
|---|
| APPLICATION LAYER |
| TRANSPORT LAYER |
| **NETWORK LAYER** |
| LINK LAYER |
| PHYSICAL LAYER |

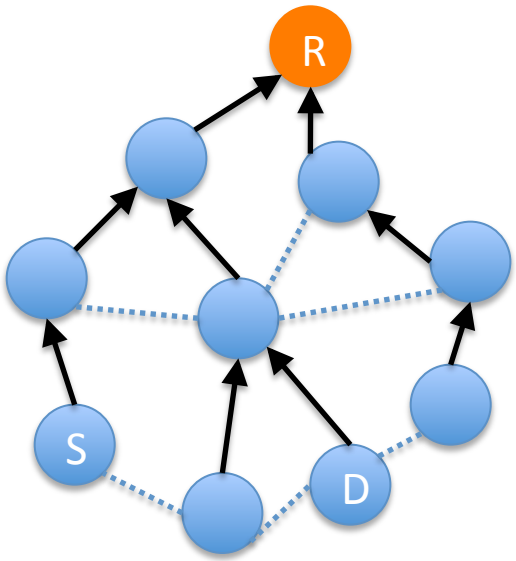
# Routing in the IoT

Due to memory constraints, not enough space to store traditional routing database

**New/modified protocols needed**

Assumptions on data traffic patterns led to tree-based IPv6 routing standard **RPL** (RFC 6550).

**Convergecast with RPL**




**Directed Acyclic Graph (DAG)**

| |
|---|
| APPLICATION LAYER |
| TRANSPORT LAYER |
| **NETWORK LAYER** |
| LINK LAYER |
| PHYSICAL LAYER |

# Transport Layer Aspects

- TCP both sucks and is bloated on multi-hop wireless paths.
- So in practice, UDP is used instead
- But UDP is not optimal either!
  - some functionalities are « outsourced» to the application
  - would benefit from being at the transport layer
  - e.g. end-to-end ACK

| |
|---|
| APPLICATION LAYER |
| **TRANSPORT LAYER** |
| NETWORK LAYER |
| LINK LAYER |
| PHYSICAL LAYER |

# AGENDA

- Characteristics of the Internet of Things
  - Hardware aspects
  - Software platform aspects
  - Network stack aspects
  - Application aspects
- RIOT: a friendly software platform for the IoT
- Course Format & Requirements
- Proposed software projects topics

# Application Layer Aspects

- Request/response mechanism:
  - HTTP is not energy/memory efficient, and does not support subscribe primitive
  - Alternative: CoAP request/response protocol for constrained devices
- Data representation:
  - ASCII-based data XML/JSON representation is not efficient in terms of bits-over-the-air
  - Alternative binary representation such as CBOR
- Resource/service management & discovery:
  - SNMP or NetConf are too bulky
  - Alternative approaches such as LWM2M, SNMP over CoAP/CBOR

| |
|---|
| **APPLICATION LAYER** |
| TRANSPORT LAYER |
| NETWORK LAYER |
| LINK LAYER |
| PHYSICAL LAYER |

# AGENDA

- Characteristics of the Internet of Things
  - Hardware aspects
  - Software platform aspects
  - Network stack aspects
  - Application aspects
- RIOT: a friendly software platform for the IoT
- Course Format & Requirements
- Proposed software projects topics

# Wishlist for an IoT operating system

- No dominant programming model for the IoT
  - Event-driven sensor network OS, non-standard programming language: Contiki, Tiny OS
  - Multi-threading micro-kernels: RIOT, FreeRTOS

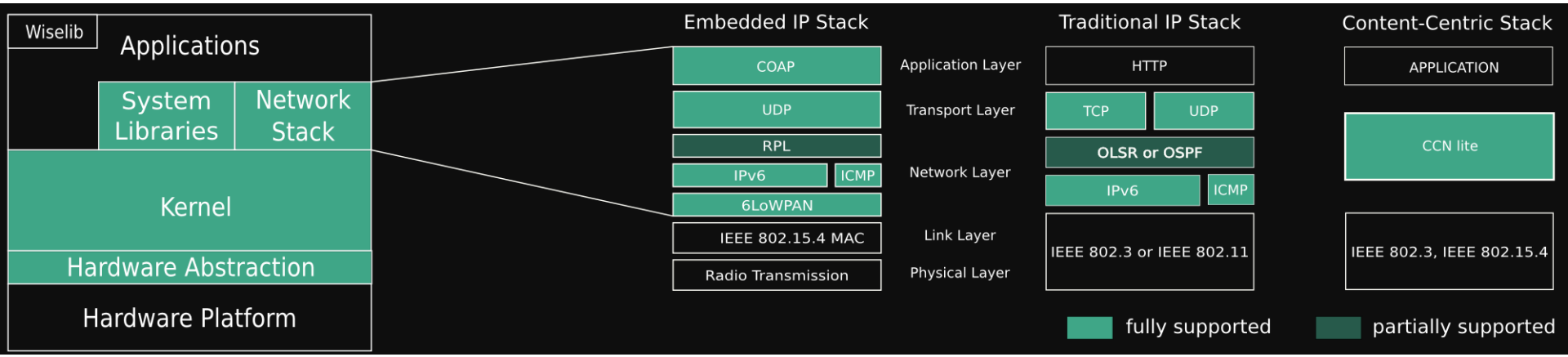
- But the rest of the Internet has a common programing model based on multi-threading, POSIX, and standard programming languages

# Wishlist for an IoT operating system

- No dominant network stack for the IoT yet
  - Vertical silos: Zigbee, ISA100, WirelessHART
  - Layered stack: 6LoWPAN, RPL, UDP, CoAP

- But the rest of the Internet has a dominant network stack based on layers & IP protocols

# RIOT is a friendly OS for IoT




- Microkernel (for robustness)
- Modular structure to adapt to varying requirements
- Tickless scheduler (for energy efficiency)
- Deterministic kernel behaviour (for real-time capability)
- Low latency interrupt handling
- Full multithreading

# RIOT is a tool for app developers

- POSIX API (e.g. sockets as you're used to)
- Develop your application in C or C++
- Use advanced debugging tools
  - gdb, Valgrind, profiler etc.
  - emulate RIOT directly on your Linux/BSD machine
- Develop once, run everywhere
  - 16bit platforms (e.g. MSP430), 32 bit platforms (e.g. ARM Cortex), intial port to x86 on the way

# Hello World in RIOT

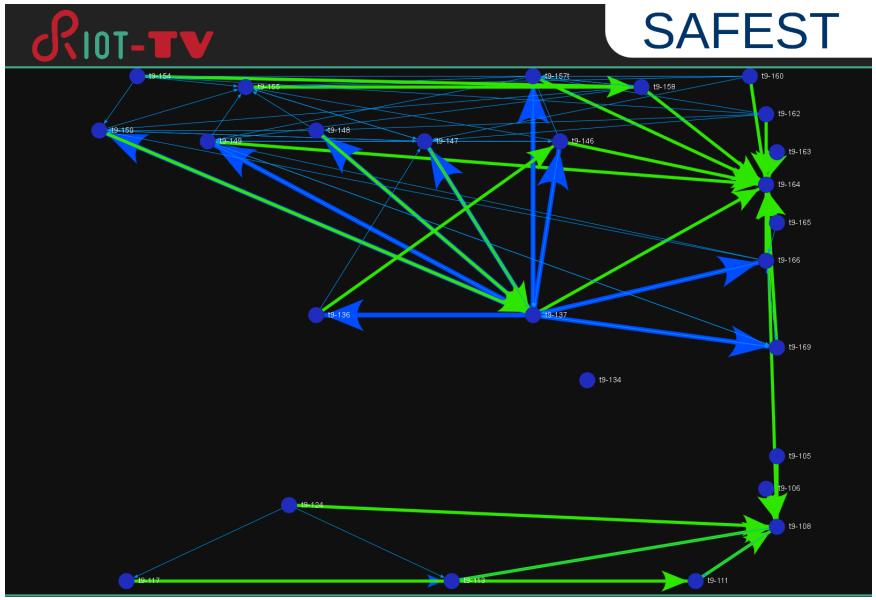
```
#include <stdio.h>

int main(void)
{
  printf("Hello World!\n");

  return 0;
}
```

# RIOT is a tool for research

- Experiments on testbeds
  - Network protocols like 6LoWPAN, RPL, CCN etc.
  - Transport protocols
  - MAC protocols
  - Applications
  - Distributed processing




- Network simulation
  - native port with desvirt to emulate various networks of multiple RIOT instances
  - RIOT-TV for visualization

```
--> ./bin/rpl_udp_router.elf tap0
RIOT native interrupts/signals initialized.
RIOT native uart0 initialized.
LED_GREEN_OFF
LED_RED_ON
RIOT native board initialized.
RIOT native hardware initialization complete.

kernel_init(): This is RIOT! (Version: 2013.08-622-ga723-tbilisi)
Scheduler...[OK]
kernel_init(): jumping into first task...
UART0 thread started.
uart0_init() [OK]
RPL router v1.1
> ps
ps
        pid | name                | state    Q | pri | stack ( used) location  | runtime | switches
          0 | idle                | pending  Q |  31 |  8192 ( 1100) 0x8068ce0 | 99.927% |        1
          1 | main                | running  Q |  15 | 16384 ( 3268) 0x8064ce0 |  0.012% |        6
          2 | uart0               | bl rx    _ |  14 |  8448 (  896) 0x806ad20 |  0.005% |       10
            | SUM                 |            |     | 33024
>
```

# RIOT is an open source community

- Github
  - https://github.com/RIOT-OS/RIOT
- Developers mailing list
  - devel@riot-os.org
- On the web
  - www.riot-os.org

# AGENDA

- Characteristics of the Internet of Things
  - Hardware aspects
  - Software platform aspects
  - Network stack aspects
  - Application aspects
- RIOT: a friendly software platform for the IoT
- Course Format & Requirements
- Proposed software projects topics

# Proposed Appoach for your Project

- Contribute to the RIOT open source community developement
- Choose a topic in a proposed category (system, network stack, application)
- Code & interact with the community

# Course Format & Requirements

- 85% attendance, at least enough to provide a short, informal weekly update on behalf of your team
- Attendance in the first three events, and mandatory intermediate status presentations
- Talk/presentation/demo
- Written report
- The outcome of your software project should be a concrete contribution to the RIOT code base, and take the shape of one or more pull request(s) to the RIOT github.

# AGENDA

- Characteristics of the Internet of Things
  - Hardware aspects
  - Software platform aspects
  - Network stack aspects
  - Application aspects
- RIOT: a friendly software platform for the IoT
- Course Format & Requirements
- Proposed software projects topics

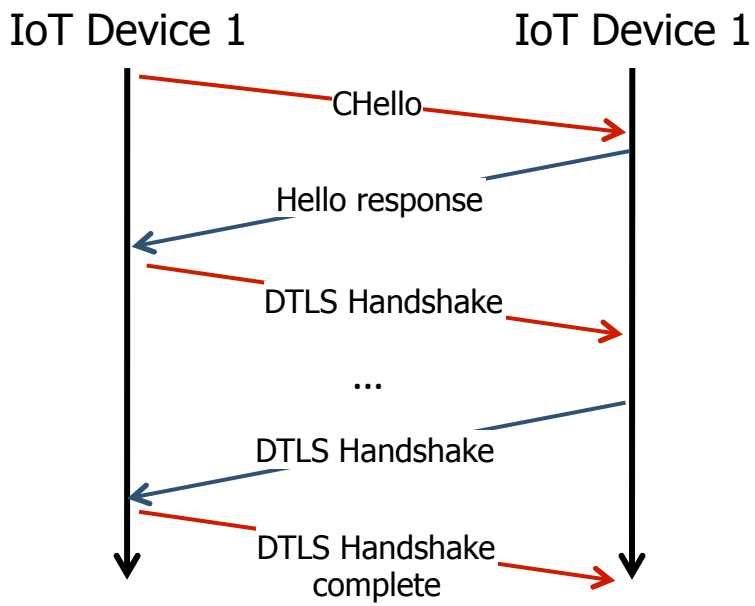
# Three Categories of Topics

- System
- Network
- Application

- In the following: a subset of proposed topics

# Topic 1: DTLS Implementation

- **Project Category**: Network
- **Summary**: Datagram Transport Layer Security (DTLS) RFC 6347 is the standard to provide end-to-end security between IoT devices.



# Topic 1: DTLS Implementation

- **Key challenges**: limiting code size and memory consumption. See:

www.ietf.org/id/draft-ietf-lwig-tls-minimal-01.txt

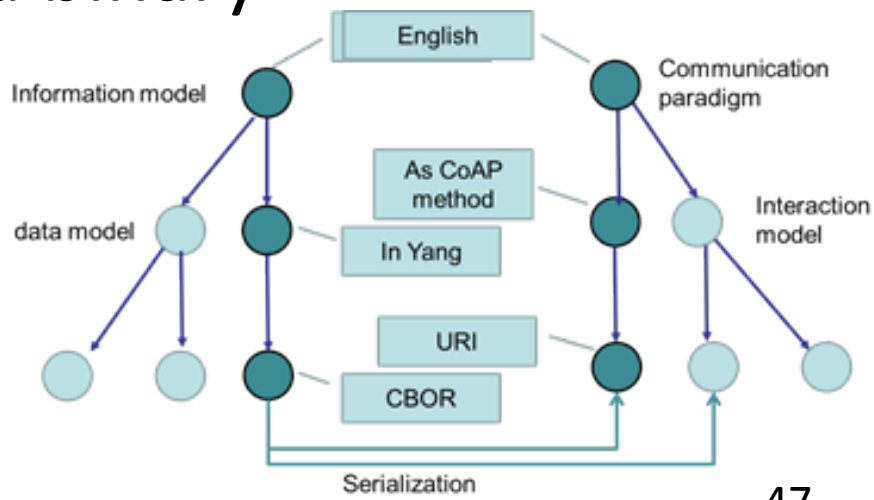
- **Expected results**: Transport Layer Security can be used over the available UDP implementation in RIOT

- **Skills**: C, git, COAP, some notions of cryptography

# Topic 2 : Implementation of CBOR

**Project Category**: Application (Application Layer)

**Summary**: CBOR (RFC7049) stands for Concise Binary Object representation and is an alternative for JSON for constrained devices using a binary representation.



# Topic 2 : Implementation of CBOR

**Key challenges**: finding a representation of mixed objects in C while keeping a low memory footprint

**Expected results**: A complete CBOR implementation for RIOT tested with at least one existing implementation

**Skills**: C, git, knowlege of JSON is a plus

# Topic 3: 8bit Support

- **Project Category**: System
- **Summary**: RIOT is currently designed and implemented for 16- and 32-bit platforms. In the IoT context 8bit platforms (e.g. the Arduino Uno) play an important role. In this project RIOTs core should be analyzed and concepts for porting RIOT to 8-bit platforms should be prototypical implemented.

Arduino Uno board
8-bit Atmel AVR

# Topic 3: 8bit Support

**Key challenges**: Analysis of RIOT's core with respect to 8-bit compatibility

**Expected results**: A prototype implementation on a 8-bit platform (e.g. Arduino Uno)

**Skills**: C, git, knowledge of a 8-bit is a plus

# Topic 4: Software Updates

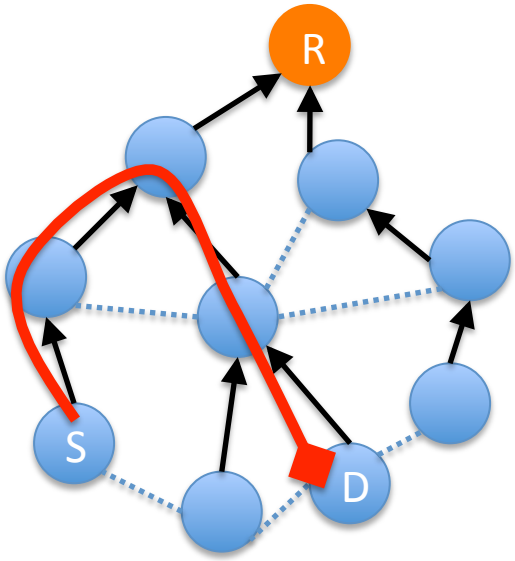
- **Project Category**: System
- **Summary**: For massive deployment of constrained devices an automatism for 'over-the-air' SW updates is inevitable. The targeted approach for this project is to dynamically link RIOT's modules so they can be exchanged at runtime.

# Topic 4: Software Updates

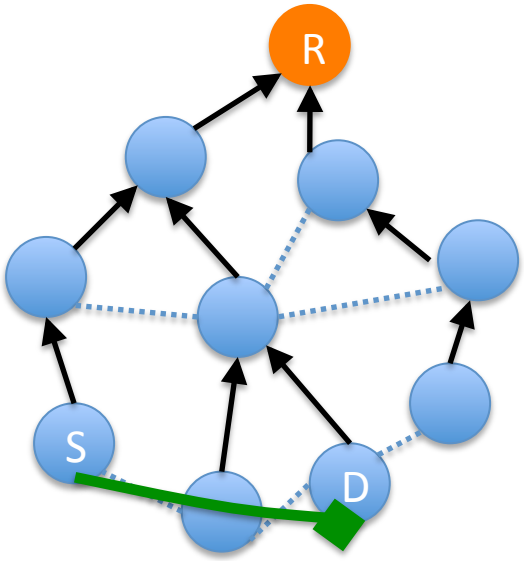
- **Key challenges**: Dynamic linking on constrained devices
- **Expected results**: A concept for dynamically linking modules to RIOT on a target platform
- **Skills**: C, git, gcc

# Topic 5: P2P-RPL Implementation

- **Project Category**: Network
- **Summary**: P2P-RPL (RFC 6997) is the standard providing on-demand route discovery with RPL




**Path with basic RPL**




**Path with P2P extension**

# Topic 5: P2P-RPL Implementation

- **Key challenges**: Distributed processing aspects. Limiting code size & memory consumption.

- **Expected results**: P2P-RPL mode of operation can be used as the routing protocol in RIOT (demonstrated both on virtual network and on testbed)

- **Skills**: C, git, basic knowlegde of wireless and computer networks

# Topic 6: eLua port

- **Project Category**: Application
- **Summary**: is a bare-metal embedded distribution of the Lua programming language. This project aims to create a generic port of eLua to RIOT so that eLua's extensive Lua interpreter, libraries and applications can be run on any RIOT powered device with enough ressources.

# Topic 6: eLua port

- **Key challenges**: Dynamic linking on constrained devices
- **Expected results**: eLua runs as an application on top of RIOT
- **Skills**: C, Lua, git

# List of All Proposed Topics

- More proposed projects, links and more info at:

https://github.com/RIOT-OS/RIOT/wiki/SWP14

- Questions? Talk/write to us

- Feel free to propose other projects
  - but do it fast => deadlines looming

# Next Steps

- Monday April 21st at 18:00. Deadline to send your top 3 favorite topics (in order of preference).
  - Send email to emmanuel.baccelli@fu-berlin.de
- Tuesday April 22nd at 14:00. Topic assignments.
- Sunday April 27th at 18:00. Deadline to hand in your project proposal + planned schedule (max. 5 pages)
  - Send email to emmanuel.baccelli@fu-berlin.de

# Presentations / Report / PR

- April 29th: 3 min project proposal/schedule presentation
- May 13th: 5 min status presentation per team
- June 17th: 10 min status presentation per team
- July 15th: Final presentation/demo (15-20mn per project)
- July 15th: Report due.
- July 15th: Pull request(s) due.

Remark: all of the above is **MANDATORY**

# Additional Information

- On all other Tuesdays
  - Hack session
  - Continuous, informal project updates
    - Tip: Make sure your team is represented often to chat with us informaly about your progress

- Check the course page for info, news, updates

http://www.mi.fu-berlin.de/inf/groups/ag-tech/teaching/2014_SS/P_19517g_SWP_Telematik/index.html

# And remember to…

- Register on e-Campus!

  https://www.ecampus.fu-berlin.de/

- Register on the mailing list!

  http://lists.riot-os.org/mailman/listinfo/devel

(preferably with a non-FU address)

# Special guest

- On the session of May 6th at 14:00
  - Presence mandatory!
- Talk from Udoo member Michelangelo Guarise
- http://www.udoo.org