

An Introduction to IoT Operating Systems

IoT Lab @ AUT: OS Team
Amirkabir University of Technology

elahejalalpoor@gmail.com
parham.alvani@gmail.com

August 12, 2015



Outline

- ▶ Part I: IoT OS
- ▶ Part II: IoT Protocol Stack
- ▶ Part III: IoT Development
- ▶ Conclusion

Outline

- ▶ Part I: IoT OS
 - Introduction
 - IoT Requirements & Challenges
 - IoT OS
 - Existing OSs
- ▶ Part II: IoT Protocol Stack
- ▶ Part III: IoT Development
- ▶ Conclusion

Outline

► Part I: IoT OS

- Introduction
- IoT Requirements & Challenges
- IoT OS
- Existing OSs

► Part II: IoT Protocol Stack

- Traditional Stack
- IoT Requirements
- IoT Stack
- Comparison

► Part III: IoT Development

► Conclusion

Outline

- ▶ Part I: IoT OS
 - Introduction
 - IoT Requirements & Challenges
 - IoT OS
 - Existing OSs
- ▶ Part II: IoT Protocol Stack
 - Traditional Stack
 - IoT Requirements
 - IoT Stack
 - Comparison
- ▶ Part III: IoT Development
 - The “IoT-Lab”
 - RIOT environment
 - Compilers
 - Development environment
- ▶ Conclusion

Outline

- ▶ Part I: IoT OS
 - Introduction
 - IoT Requirements & Challenges
 - IoT OS
 - Existing OSs
- ▶ Part II: IoT Protocol Stack
- ▶ Part III: IoT Development
- ▶ Conclusion

IoT?!!

What is IoT... .

[Wikipedia]: The network of physical objects or “things” embedded with electronics, software, sensors, and connectivity to enable objects to exchange data with the manufacturer, operator and/or other connected devices based on the infrastructure of ITU's Global Standards Initiative

What is IoT... .

[ITU]: A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies

What is IoT... .

[WhatIs]: A scenario in which objects, animals or people are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

What is the IoT?

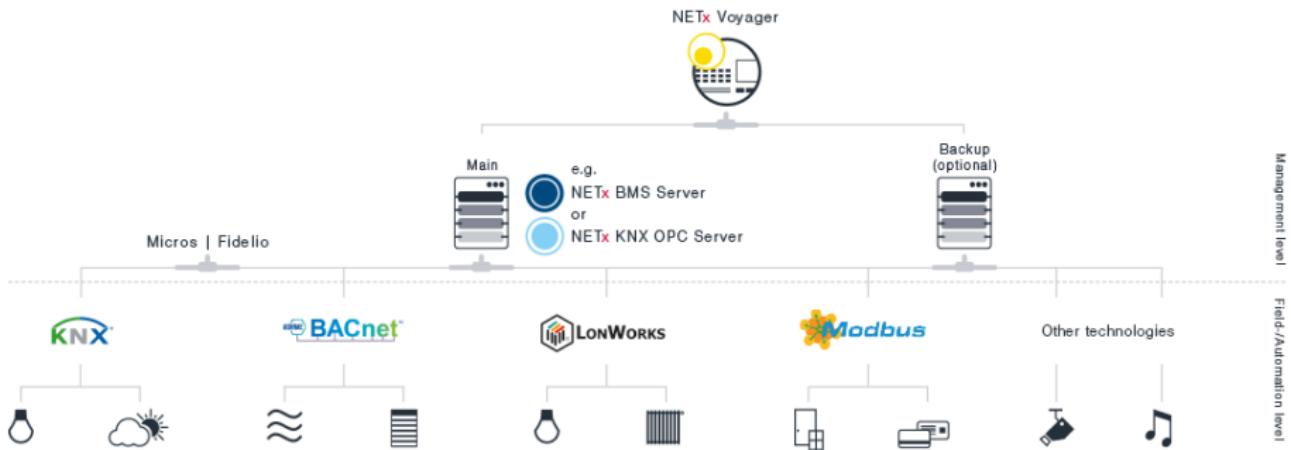
- ▶ A **thing** in IoT can be any natural or man-made object can be assigned IP(v6) address.
- ▶ So far, the Internet of Things has been most closely associated with machine-to-machine (M2M) communication.
- ▶ Although the concept wasn't named until 1999, the Internet of Things has been in development for decades.



IoT's Applications

- ▶ Environmental monitoring
- ▶ Infrastructure management
- ▶ Manufacturing
- ▶ Energy management
- ▶ Medical and healthcare systems
- ▶ **Building and home automation**
- ▶ Transportation
- ▶ ...

Building and Home Automation



Outline

- ▶ Part I: IoT OS
 - Introduction
 - IoT Requirements & Challenges
 - R1: Heterogeneous Hardware Constraints
 - R2: Autonomy
 - R3: Programmability
 - Effect of the requirements on OS
 - IoT OS
 - Existing OSs
- ▶ Part II: IoT Protocol Stack
- ▶ Part III: IoT Development
- ▶ Conclusion

R1: Heterogeneous Hardware Constraints

- ▶ Memory Requirements
- ▶ CPU Requirements
- ▶ Limited Features
- ▶ Platform Support



Memory Requirements

- ▶ Many of typical IoT devices have very little memory (typically between 5kB and some hundreds of megabytes)
- ▶ This concerns RAM as well as persistent program storage.

Effects on OS

- ▶ Kernel image should be very small
- ▶ The RAM footprint should be very low
- ▶ The OS should be modular!

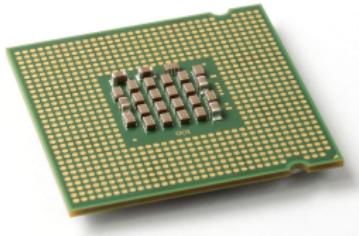


CPU Requirements

- ▶ Some of the IoT systems are MCU based (instead of CPU)
- ▶ Some of the MCUs/CPUs in a IoT system will work at a very low clock cycle.

Effects on OS

- ▶ The complexity of OS must be kept very low
- ▶ Should be scalable, to accommodate a wide range of different classes of devices



Limited Features

- ▶ IoT's hardware may have not advanced components like a Memory Management Unit (MMU) or a Floating-Point Unit (FPU).

Effects on OS

- ▶ Software for IoT must be able to run on constrained HW
- ▶ Should be scalable, to accommodate a wide range of different classes of devices



Platform Support

- ▶ IoT platforms may have very limited resources; e.g., battery, IO, storage, ...
- ▶ IoT platforms may be composed of widely different components

Effects on OS

- ▶ Must be able to leverage the capabilities of less constrained platforms
- ▶ Should be scalable, to accommodate a wide range of different classes of devices



R2: Autonomy

- ▶ Energy Efficiency
- ▶ Adaptive Network Stack
- ▶ Reliability



Energy Efficiency

- ▶ Some IoT nodes are battery powered
- ▶ Energy efficiency is one the goals of IoT

Effects on OS

- ▶ Must exploit the power saving features of the hardware and allow for large sleep cycles as much as possible



Adaptive Network Stack

- ▶ Connectivity is an ingredients part of IoT
- ▶ IoT is a part of Internet, TCP/IP based network
- ▶ IoT can also use its own special protocols

Effects on OS

- ▶ Should provide full-fledged TCP/IP implementations
- ▶ As well as a 6LoWPAN stack aiming for more constrained devices.
- ▶ It should also be modular in a way that the protocols at each layer can be easily replaced.

Reliability

- ▶ IoT systems are often deployed in critical applications in which physical access is difficult and related to high costs
- ▶ Timely response is critical in some applications

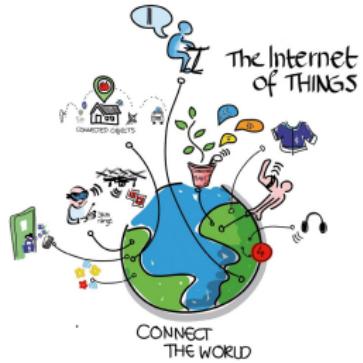
Effects on OS

- ▶ System must be robust and thus that the operating system should run very reliably
- ▶ Real-Time OS in some applications



R3: Programmability

- ▶ Standard API
- ▶ Standard Programming Languages



Standard API & Programming Languages

- ▶ Need for SW development for IoT systems
- ▶ Porting of existing software on IoT systems

Effects on OS

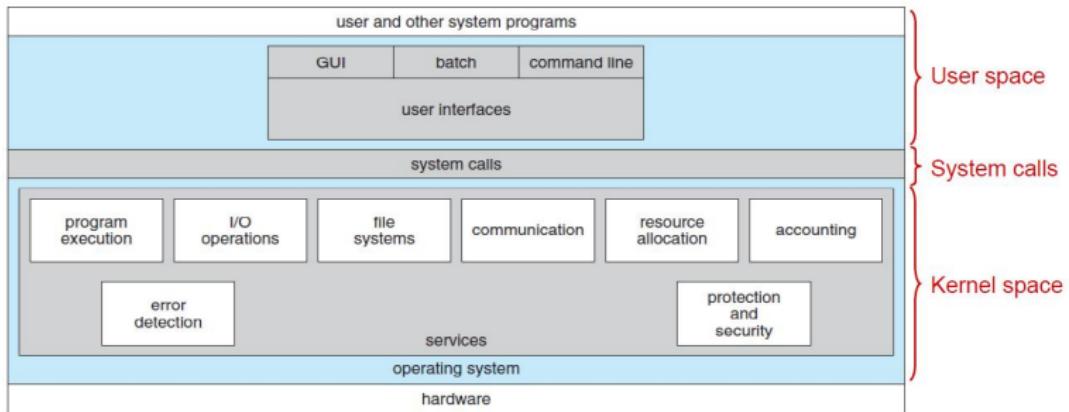
- ▶ Standard programming interface such as POSIX or STL should be provided
- ▶ Support for standard high level programming languages, e.g., C & C++, is vital.



Outline

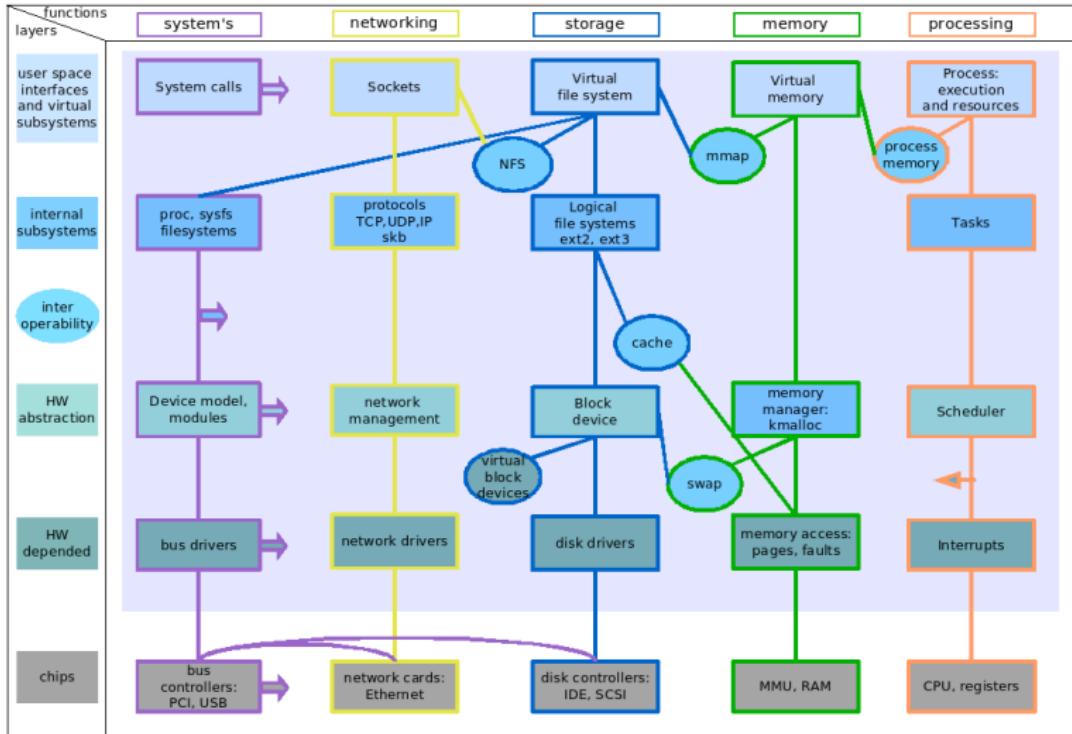
- ▶ Part I: IoT OS
 - Introduction
 - IoT Requirements & Challenges
 - IoT OS
 - General OS vs IoT OS
 - What are the main requirements in IoT OS
 - What are the main components in IoT OS
 - Existing OSs
- ▶ Part II: IoT Protocol Stack
- ▶ Part III: IoT Development
- ▶ Conclusion

General OS



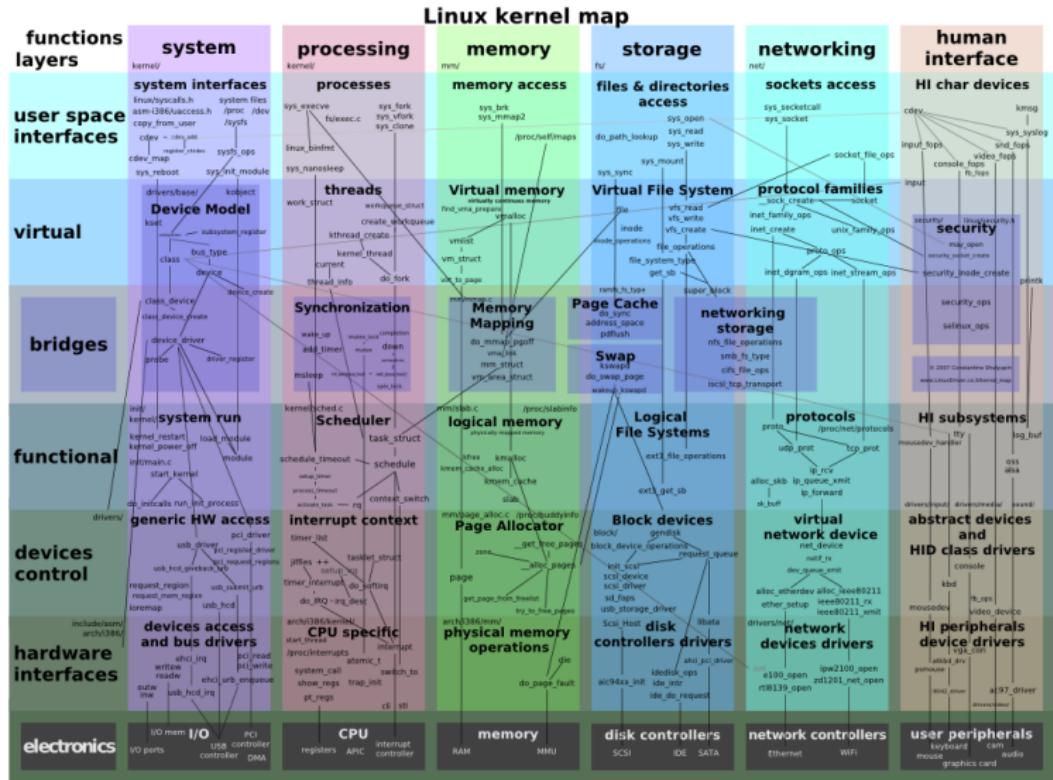
Linux Kernel Architecture (Simple Version)

Simplified Linux kernel diagram in form of a matrix map



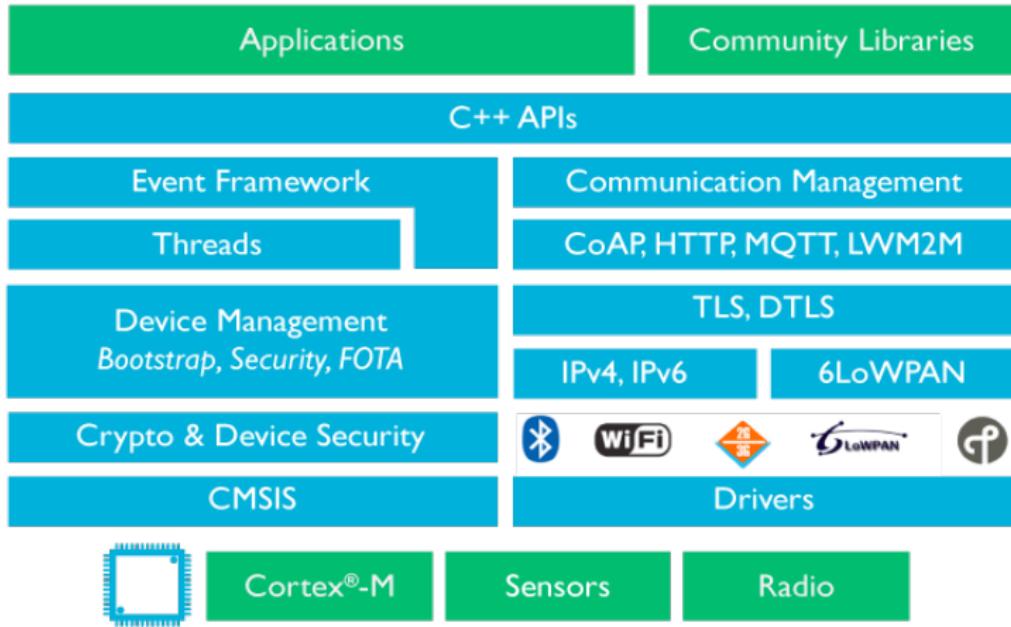
Designed with OpenOffice.org by (cc) (by-nc-sa) Constantine Shulyupin, www.linuxdriver.co.il

Linux Kernel Architecture (Almost Complete Version)



- ▶ Much more simpler than general OS/Kernel
- ▶ Do we need?!
 - Human interface
 - Full wide-range storage support
 - Virtual memory
 - Full traditional protocol stack
- ▶ Its own requirements, remarkably
 - IoT Protocol Stack Support
 - Low Complexity
 - Efficient Memory Managing
 - Real-Time Task Scheduling

IoT OS Example: ARM's mbed



Outline

► Part I: IoT OS

- Introduction
- IoT Requirements & Challenges
- IoT OS
- Existing OSs
 - OS Classification
 - Overview of Open Source OSs
 - Overview of Closed Source OSs
 - Why Not Linux?

► Part II: IoT Protocol Stack

► Part III: IoT Development

► Conclusion

OS Classification

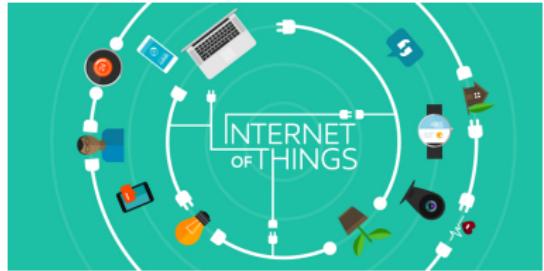
- ▶ Single-tasking vs. Multi-tasking
- ▶ Single-user vs. Multi-user
- ▶ Monolithic vs. Microkernel
- ▶ Process model vs. Thread model
- ▶ Open source vs. Commercial

OS Classification

- ▶ Single-tasking vs. Multi-tasking
 - Multi-tasking in IoT OS
- ▶ Single-user vs. Multi-user
 - Single-user in IoT OS
- ▶ Monolithic vs. Microkernel
 - Both can be used in IoT OS
- ▶ Process model vs. Thread model
 - Thread in IoT OS, since No MMU & Realtimeness
- ▶ Open source vs. Commercial
 - Both are available in IoT OS market

Overview of Open Source OSs

- ▶ FreeRTOS
- ▶ RIOT
- ▶ Contiki
- ▶ TinyOS
- ▶ Embedded Linux
- ▶ OpenWSN



- ▶ FreeRTOS is designed to be **small** and **simple**.
- ▶ It claims that is market leading RTOS
- ▶ The kernel itself consists of only three or four C files.
- ▶ It provides methods for multiple threads or tasks, mutexes, semaphores and software timers.
- ▶ Key features are **very small memory footprint**, **low overhead**, and **very fast execution**.



- ▶ RIOT is a **real-time multi-threading** operating system.
- ▶ RIOT implements a **microkernel** architecture
- ▶ RIOT is based on design objectives including:
 - Energy-Efficiency
 - Reliability
 - Real-Time Capabilities
 - Small Memory Footprint
 - Modularity
 - Uniform API Access
independent of the underlying hardware
(this API offers partial POSIX compliance)



- ▶ Contiki is an open source operating system for **networked, memory-constrained** systems
- ▶ Contiki provides three network mechanisms:
 - The uIP stack, which provides IPv4 networking,
 - The uIPv6 stack, which provides IPv6 networking,
 - The Rime stack, which is a set of custom lightweight networking protocols designed specifically for low-power wireless networks.

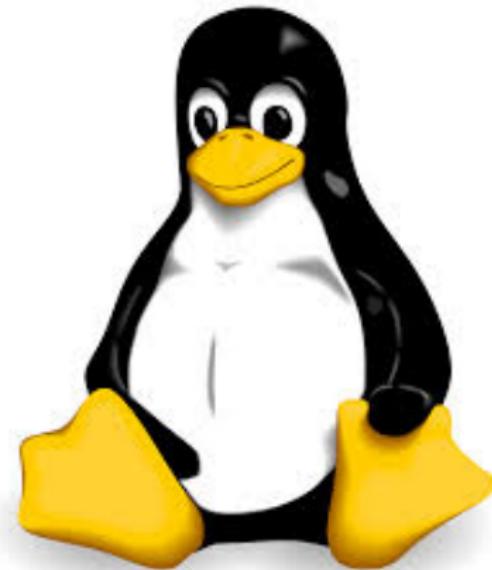


- ▶ TinyOS is a **component-based** operating system and platform targeting wireless sensor networks.
- ▶ TinyOS is an embedded operating system written in the **nesC programming language** as a set of cooperating tasks and processes.



Embedded Linux

- ▶ Embedded Linux is created using OpenEmbedded, the build framework for embedded Linux.
- ▶ OpenEmbedded offers a best-in-class cross-compile environment.



OpenWSN

- ▶ The goal of the OpenWSN project is to provide open-source implementations of a complete protocol stack based on Internet of Things standards, on a variety of software and hardware platforms.
- ▶ Under-development protocols
 - IEEE802.15.4e
 - IETF 6TiSCH
 - IETF 6LoWPAN
 - IETF ROLL



Comparison

OS	Min RAM	Min ROM	C Support	C++ Support
Contiki	< 2kB	< 30kB	Partial support	No support
Tiny OS	< 1kB	< 4kB	No support	No support
Linux	~ 1MB	~ 1MB	Full support	Full support
RIOT	~ 1.5kB	~ 5kB	Full support	Full support



Comparison

OS	Multi-Threading	Modularity	Real-Time
Contiki	Partial support	Partial support	Partial support
Tiny OS	Partial support	No support	No support
Linux	Full support	Partial support	Partial support
RIOT	Full support	Full support	Full support

Contiki

TinyOS



RIOT

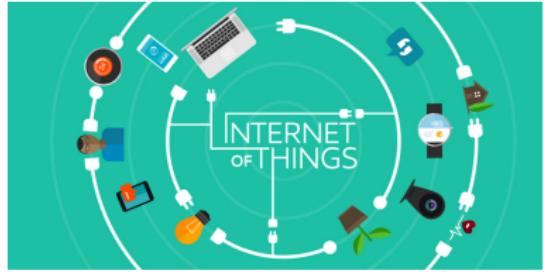
Operating Systems Availability

OS	Wsn430 Node	M3 Node	A8 Node
Contiki	Full support	Full support	No support
Tiny OS	Full support	No support	No support
Linux	No support	No support	Full support
RIOT	Full support	Full support	No support



Overview of Closed Source OSs

- ▶ ARM mbed
- ▶ Huawei LiteOS
- ▶ Google Brillo



- ▶ Automation of power management
- ▶ Software asset protection and secure firmware updates for device security & management
- ▶ Connectivity protocol stack support for Bluetooth low energy, Cellular, Ethernet, Wi-fi, Zigbee IP, Zigbee NAN, 6LoWPAN

ARM[®] mbed[™]

- ▶ The company says that its **LiteOS** is the **lightest** software of its kind and can be used to power a range of smart devices
- ▶ There is another open-source “LiteOS” project (UNIX-like OS for WSN). The relation between this & that is not known.



Google Brillo

- ▶ Brillo is derived from Android but polished to just the lower levels.
- ▶ It supports Wi-Fi, Bluetooth Low Energy, and other Android things.
 - Moreover, it will support the “Weave” protocol



Why Not Linux?

Real-Time Linux

Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want to use Linux to control an industrial welding laser, I have no problem with your using PREEMPT_RT.

- Linus Torvalds



Why Not Linux?

- ▶ Linux certainly is a robust, developer-friendly OS
- ▶ Linux has a disadvantage when compared to a real-time operating system:
 - Memory footprint
 - It simply will not run on 8 or 16-bit MCUs
- ▶ Linux will certainly have many uses in embedded devices, particularly ones that provide graphically rich user interfaces.
- ▶ There are thousands of applications for which Linux is ill suited.



Outline

- ▶ Part I: IoT OS
- ▶ Part II: IoT Protocol Stack
 - Traditional Stack
 - IoT Requirements
 - IoT Stack
 - Comparison
- ▶ Part III: IoT Development
- ▶ Conclusion

Protocol stack

- ▶ Can you build an IoT system with familiar Web technologies?

Protocol stack

- ▶ Can you build an IoT system with familiar Web technologies?
- ▶ Yes you can, although the result would not be as **efficient** as with the **newer protocols**.

Traditional Stack

- ▶ Existing Internet protocols such as HTTP and TCP are not optimized for very **low-power communication**.
- ▶ Energy is wasted by transmission of **unneeded data, protocol overhead**, and **non-optimized communication patterns**.

IoT Requirements

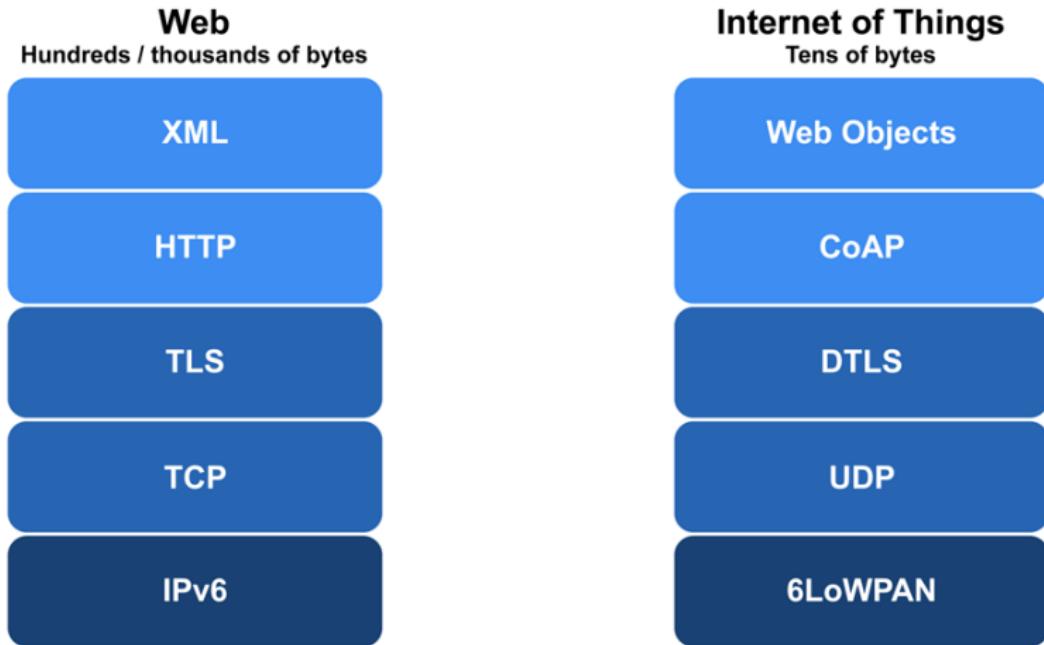
- ▶ A Low Power Communication Stack.
- ▶ A Highly Reliable Communication Stack.
- ▶ An Internet-Enabled Communication Stack.

- ▶ LOW-POWER PHYSICAL LAYER **IEEE 802.15.4**
- ▶ POWER-SAVING LINK LAYER **IEEE 802.15.4E**
- ▶ CONNECTING TO THE INTERNET - **IETF 6LoWPAN**
- ▶ ROUTING - **IETF ROLL**
- ▶ TRANSPORT LAYER AND ABOVE - **IETF CoAP**

IoT Stack

<i>Protocol</i>	<i>Transport</i>	<i>Messaging</i>	<i>2G,3G,4G (1000's)</i>	<i>Low Power and Lossy (1000's)</i>	<i>Compute Resources</i>	<i>Security</i>	<i>Success Stories</i>	<i>Arch</i>
CoAP	UDP	Rqst/Rspnse	Excellent	Excellent	10Ks/RAM Flash	Medium - Optional	Utility field area ntwks	Tree
Continua HDP	UDP	Pub/Subsrbs Rqst/Rspnse	Fair	Fair	10Ks/RAM Flash	None	Medical	Star
DDS	UDP	Pub/Subsrbs Rqst/Rspnse	Fair	Poor	100Ks/RAM Flash +++	High-Optional	Military	Bus
DPWS	TCP		Good	Fair	100Ks/RAM Flash ++	High-Optional	Web Servers	Client Server
HTTP/ REST	TCP	Rqst/Rspnse	Excellent	Fair	10Ks/RAM Flash	Low-Optional	Smart Energy Phase 2	Client Server
MQTT	TCP	Pub/Subsrbs Rqst/Rspnse	Excellent	Good	10Ks/RAM Flash	Medium - Optional	IoT Msging	Tree
SNMP	UDP	Rqst/Response	Excellent	Fair	10Ks/RAM Flash	High-Optional	Network Monitoring	Client-Server
UPnP		Pub/Subscrbs Rqst/Rspnse	Excellent	Good	10Ks/RAM Flash	None	Consumer	P2P Client Server
XMPP	TCP	Pub/Subsrbs Rqst/Rspnse	Excellent	Fair	10Ks/RAM Flash	High-Mandatory	Rmt Mgmt White Gds	Client Server
ZeroMQ	UDP	Pub/Subscrbs Rqst/Rspnse	Fair	Fair	10Ks/RAM Flash	High-Optional	CERN	P2P

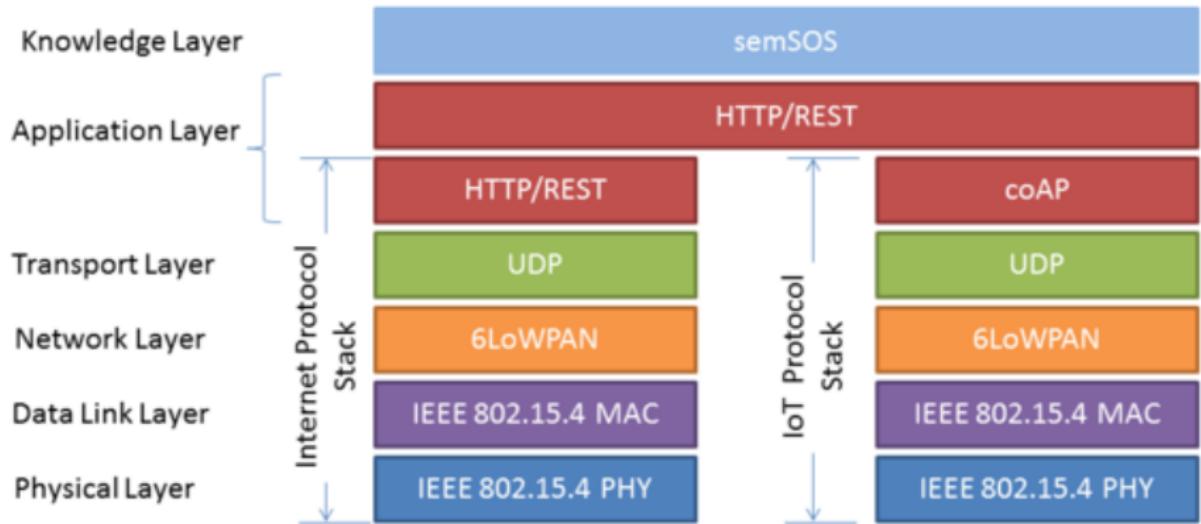
Comparison



- Inefficient content encoding
- Huge overhead, difficult parsing
- Requires full Internet devices

- Efficient objects
- Efficient Web
- Optimized IP access

Comparison

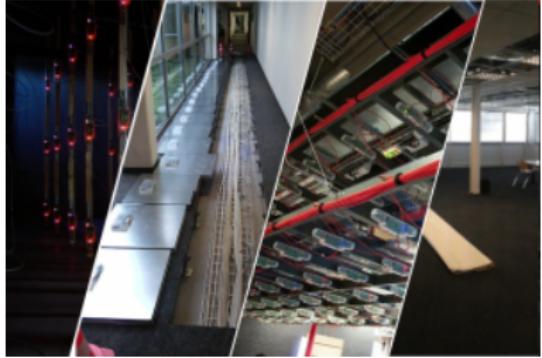


Outline

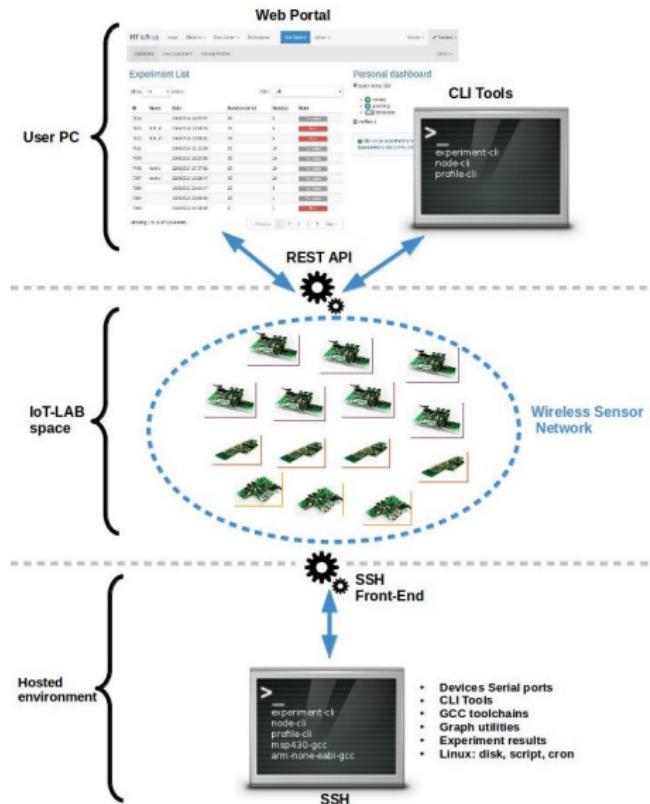
- ▶ Part I: IoT OS
- ▶ Part II: IoT Protocol Stack
- ▶ Part III: IoT Development
 - The “IoT-Lab”
 - RIOT environment
 - Compilers
 - Development environment
- ▶ Conclusion

The “IoT-Lab”

- ▶ IoT-LAB provides a very large scale infrastructure suitable for testing small wireless sensor devices and heterogeneous communicating objects
- ▶ IoT-LAB provides full control of network nodes and direct access to the gateways to which nodes are connected, allowing researchers to monitor nodes energy consumption and network-related metrics.



The “IoT-Lab”

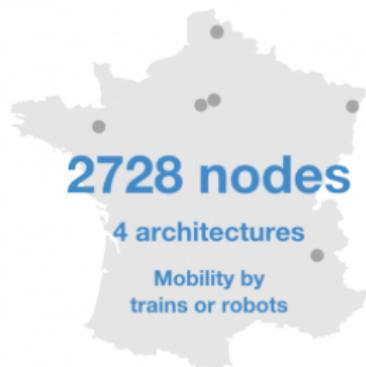


The “IoT-Lab” topologies and environments

- ▶ IoT-LAB testbeds are located at six different sites across France which gives forward access to **2728 wireless sensors nodes**.
- ▶ 512 WSN430 Node (800MHz)
- ▶ 632 WSN430 Node (2.4GHz)
- ▶ 938 M3 Node
- ▶ 550 A8 Node
- ▶ 96 Open Host Node

The “IoT-Lab” Different nodes

- ▶ The IoT-LAB hardware infrastructure consists of a set of IoT-LAB nodes.
- ▶ A global networking backbone provides power and connectivity to all IoT-LAB nodes and guarantees the out of band signal network needed for command purposes and monitoring feedback.
- ▶ Nodes' Operating Systems
 - Contiki
 - RIOT



The “IoT-Lab” A part of FIT

- ▶ IoT-LAB is a part of the FIT (Future Internet of the Things) platform.
- ▶ FIT is a set of complementary components that enable experimentation on innovative services for academic and industrial users.



The “IoT-Lab” Experiments



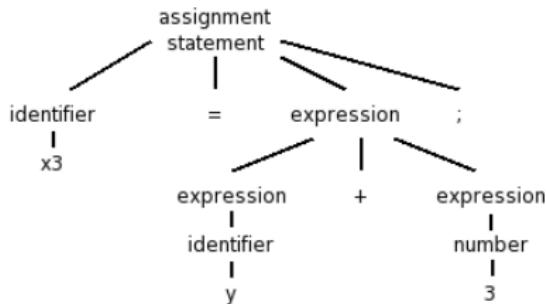
RIOT environment

- ▶ RIOT features the native port with networking support.
- ▶ This allows you to run any RIOT application on your Linux or Mac computer and setup a virtual connection between these processes.



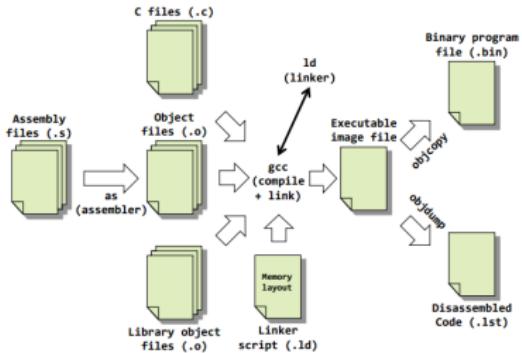
Compilers

- ▶ Family: ARM
 - gcc-arm-embedded toolchain
 - CodeBench toolchain
 - Linaro toolchain
- ▶ Family: ATmega
 - Atmel AVR Toolchain
- ▶ Family: MSP430
 - MSPGCC toolchain



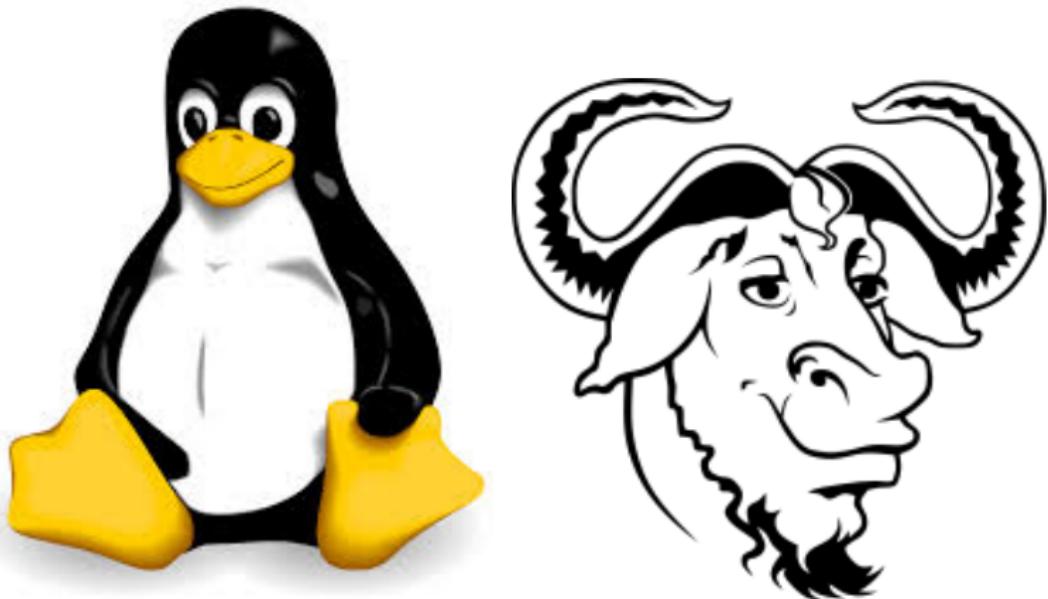
ARM: gcc-arm-embedded toolchain

- ▶ ARM is maintaining a GNU toolchain with a GCC source branch targeted at Embedded ARM Processors, namely Cortex-R/Cortex-M processor families



Development environment

- ▶ Most of the IoT OS developed on **Linux** and use **traditional make** as build system.



Outline

- ▶ Part I: IoT OS
- ▶ Part II: IoT Protocol Stack
- ▶ Part III: IoT Development
- ▶ Conclusion
 - Summary
 - Open problems

Summary

- ▶ Different applications of IoT ⇒ Different OS Requirements (e.g., Energy, Memory, Realtime, ...) ⇒ Different OSs
- ▶ A few OSs should be examined
- ▶ In the next step, we want to develop an evaluation environment and focus on RIOT
 - We have already enough experience on Linux

Open problems

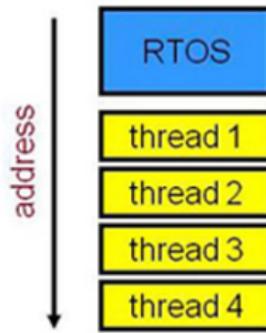
- ▶ Native Multi-Threading
- ▶ Hardware Abstraction
- ▶ Dynamic Memory Management
- ▶ Fulfill Strict Energy Efficiency

P=NP?

Questions?

Multi-Tasking, Thread Model (IoT OS)

- ▶ Most RTOS products on the market are thread model.
- ▶ Tasks are now called threads.
- ▶ All the tasks code and data occupy the same address space, along with that of the RTOS itself.
- ▶ Or every tasks can run in its own thread and has its own memory stack.



Multi-Tasking, Process Model (IoT OS)

- ▶ By using the MMU, each task (now called a process) occupies its own private address space
- ▶ This model has the great benefit that each process has no access to, or even awareness of, other processes memory or that of the OS itself
- ▶ The memory needs to be remapped using the MMU

What are the main components in IoT OS

- ▶ Networking
- ▶ Memory Manager
- ▶ Task Scheduler

▶ Programming Model for an IoT OS

- All tasks are executed within the same context and have no segmentation of the memory address space.
- Every process can run in its own thread and has its own memory stack.