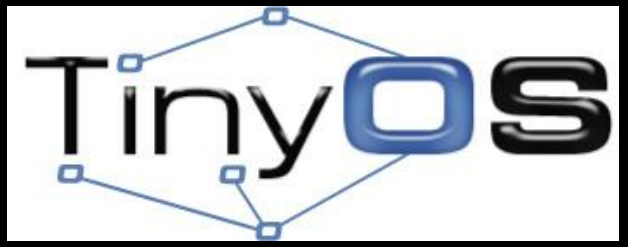
Operating Systems

For Wireless Sensor Networks

2019-1 DS – PUC-RJ
Antonio Iyda Paganelli

Summary

- Introduction
- Requirements
- Design Choices
- TinyOs and Contiki
- Conclusions



Introduction

- Wireless Sensor Networks
  - ✓Cheap, tiny, energy-efficient communicating devices
  - ✓ Unreliable or lossy channels
  - ✓ Limited and unpredictable bandwidth
  - ✓ Highly dynamic topology

- ✓**High-end IoT devices**: single-board computers like Raspeberry Pi or smartphones
- ✓**Low-end IoT devices**: Arduino duo, Econotag, Zolertia Z1, IoT-LAB M3, OpenMote, TelosB and so on …
  - Highly constrained hardware resources

[4] Hahm et at. (2016)

Low-end IoT devices

- Constrained resources
  - Energy
  - CPU
  - Memory capacity

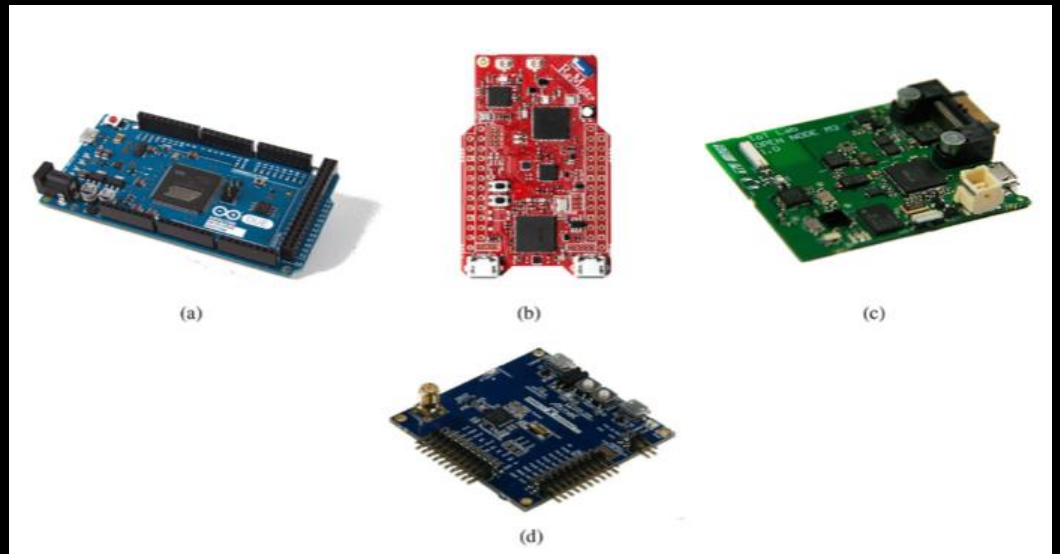


Fig. 1. Examples of low-end IoT devices. (a) Arduino due. (b) Zolertia Re-Mote. (c) IoT-LAB-M3. (d) Atmel SAM R21.

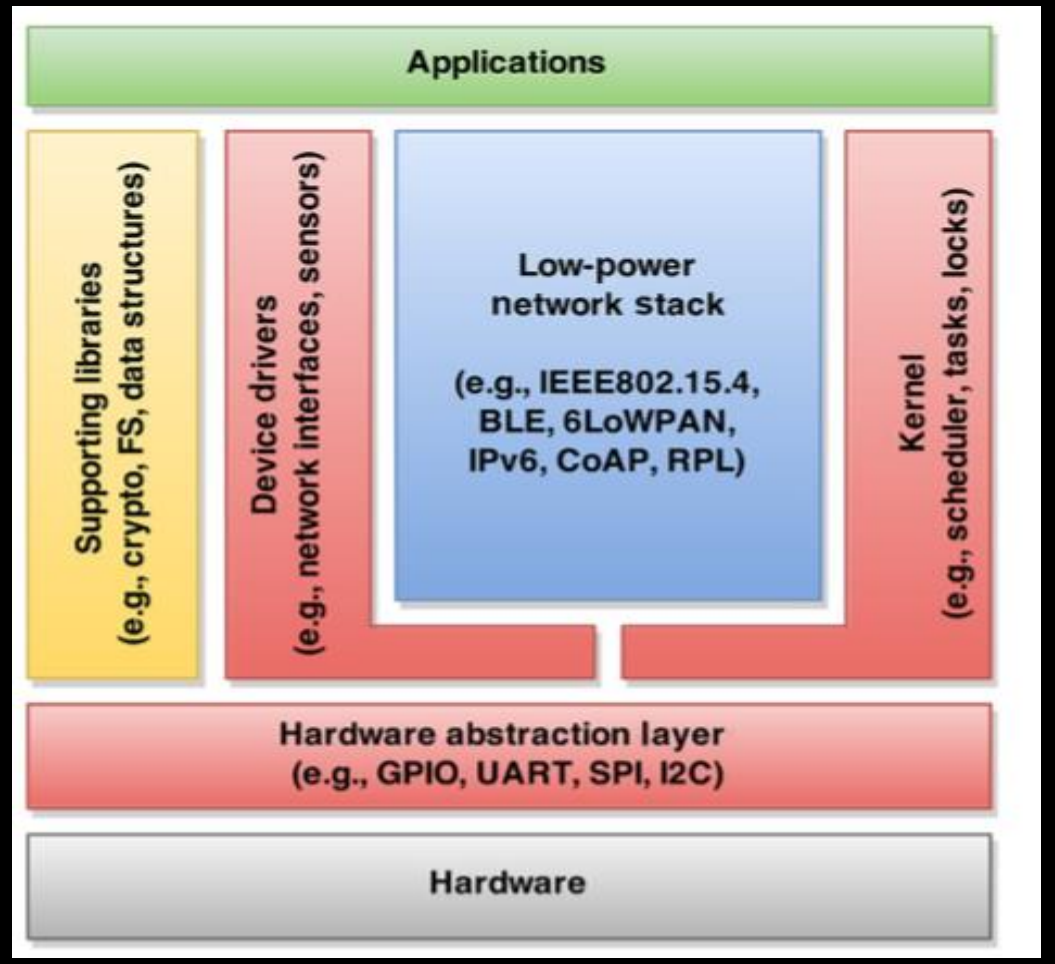
- IETF classification
  (RFC 7228 – Terminology for Constrained-Node Networks)
  - Class 0: << 10 kB RAM, << 100 kB Flash
  - Class 1: ~10 kB RAM, ~100 kB Flash
  - Class 2: more resources, but still very constrained compared to high-end IoT devices.

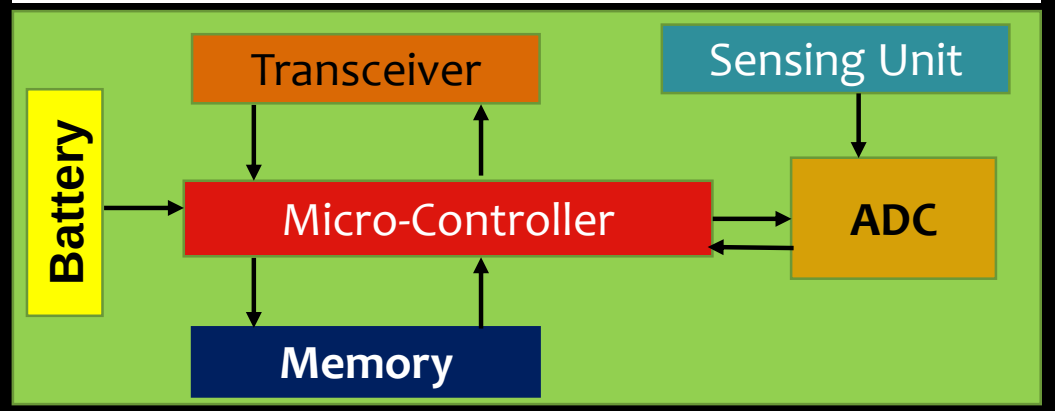
Operating Systems

- Management of shared resources:

processor, memory, timer, network interfaces, other I/O devices

- Multiplexes resources by time or by space





[3] Farroq & Kunz (2011)
[4] Hahm et at. (2016)

Operating Systems

- Class 0 – too small. Bare metal software, very specialized
- Class 1 and Class 2
- Moore's Law is not expected

Model	ROM	RAM	Sleep	Price
F2002	1kB	128B	1.3μA	\$0.94
F1232	8kB	256B	1.6μA	\$2.73
F155	16kB	512B	2.0μA	\$6.54
F168	48kB	2048B	2.0μA	\$9.11
F1611	48kB	10240B	2.0μA	\$12.86

(a) TI MSP430 Microcontrollers

Model	ROM	RAM	Sleep	Price
LM2S600	32kB	8kB	950μA	\$2.73
LM3S1608	128kB	32kB	950μA	\$4.59
LM3S1968	256kB	64kB	950μA	\$6.27

(b) TI ARM CortexM3 Processors

[4] Hahm et at. (2016)
[5] Levis (2012)

Context of OSs for WSN

Hard upper bounds on:

- State
- Code space
- Processing cycles
- Energy
- Network bandwidth
- Full connectivity
- Lack of broadcast / multicast

Additionally,

- Cost
- Size
- Weight
- Other scaling factors

(energy harvesting)

RFC 7228

Requirements

- Small memory footprint
- Support for heterogeneous hardware
- Network connectivity
- Energy efficiency
- Security
- Real-time capabilities

[4] Hahm et al (2016)

Concurrency and Flexibility

[6] Tobias Reusing (2012)

Design choices

Technical

- General Architecture and Modularity
- Scheduling Model
- Memory Allocation
- Network Buffer Management
- Programming Model
- Programming Languages
- Driver Model and Hardware Abstraction Layer
- Debugging Tools
- Feature Set
- Testing

Non-Technical

- Standards (open)
- Certification
- Documentation
- Maturity of Code
- License of Code
- Provider of the OS

[4] Hahm et al (2016)

General Architecture and Modularity

- Monolithic
  - Single image
  - Small OS memory footprint
  - Interfaces for service modules
  - Modules interaction costs are low

  - Hard to understand and modify
  - Difficult to maintain
  - Unreliable

[4] Hahm et al (2016)

General Architecture and Modularity

- Microkernel
  - Minimum functionality provided by the kernel
  - Most of functionalities provided by user-level servers
  - Small OS memory footprint
  - Better reliability
  - Ease of extension and customization

⚠

- Poor performance. Many cross-boundary kernel – user
- More complex design

[4] Hahm et al (2016)

General Architecture and Modularity

Virtual Machine

- Export VM to user programs
- VM has all the needed hardware features
- Main advantage is portability

Poor performance

Layered

- Implement services in layers
- Manageability
- Reliability
- Easy to understand

Not very flexible

[4] Hahm et al (2016)

Scheduling Model

Affects energy efficiency, real-time capabilities, and programing model

Preemptive

Non-preemptive

A preemptive scheduler **assigns CPU time to each process** while in the cooperative model, **tasks have to yield themselves**.

Many cases, preemptive scheduler requires a **systick** and it usually prevents the device to enter the deepest power-save mode.

[4] Hahm et al (2016)

Memory Allocation

Memory is usually a very scarce resource

Static

- Requires some over-provision
- Less flexible to changing requirements during run time’

Dynamic

- System design more complicated
- Time-wise nondeterministic fashion
- Handle out-of-memory situations
- Memory fragmentation

[4] Hahm et al (2016)

Network Buffer Management

Central component of an IoT OS is the network stack

Copying Memory

It seems expensive

Passing pointers

Who is responsible to allocate the memory?

Upper layers – application development more complex and less convenient.

[4] Hahm et al (2016)

Programming Model

How an application developer can model the program

Event-driven systems

- Widely used for WSN OSs
- Every task has to be triggered by an event
- Can be implemented by a simple event loop
- Shared-stack model
- Memory efficiency

Multithreaded systems

- Each task has its own thread context
- Communication between tasks by using inter-process communication API
- Easier to design

[4] Hahm et al (2016)

Programming languages

Supported programming languages for applications

Standard programming languages

- Usually ANSI C or C++
- Well-established and mature development tools
- Portability
- Debugging tools

OS-specific language

- Performance of safety-relevant enhancements
- Specific language structures
- Debugging facilities specific for those devices

[4] Hahm et al (2016)

Hardware Abstraction Layer

Systems equipped with a variety of different peripheral devices

Improves :

- System design
- Portability
- Extensibility

However, introduces some overhead in code and runtime

[4] Hahm et al (2016)

Feature set

Kernel

- Scheduler
- Mutual exclusion / sync. mechanisms
- Timers

Higher level

- Shell logging
- Cryptographic functions
- Network stacks

Over the air updates

Dynamic loading and linking

[4] Hahm et al (2016)

Testing

- Hardware
- Distributed nature
- Deeply embedded and very constrained

- Continuous integration
  - Build and integration tests
  - Unit tests
  - Regression tests

➔ Widely used approach

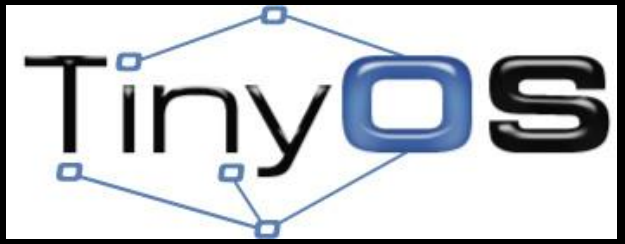
Hardware / Network emulation as well as simulation tools

[4] Hahm et al (2016)

Overview of Potential Open Source OSs for the IoT

Name	Architecture	Scheduler	Programming model	Targeted device class[a]	Supported MCU families or vendors	Programming languages	License	Network stacks
Contiki	Monolithic	Cooperative	Event-driven, Protothreads	Class 0 + 1	AVR, MSP430, ARM7, ARM Cortex-M, PIC32, 6502	C[b]	BSD	uIP, RIME
RIOT	Microkernel RTOS	Preemptive, tickless	Multithreading	Class 1 + 2	AVR, MSP430, ARM7, ARM Cortex-M, x86	C, C++	LGPLv2	gnrc, OpenWSN, ccn-lite
FreeRTOS	Microkernel RTOS	Preemptive, optional tickless	Multithreading	Class 1 + 2	AVR, MSP430, ARM, x86, 8052, Renesas[c]	C	modified GPL[d]	None
TinyOS	Monolithic	Cooperative	Event-driven	Class 0	AVR, MSP430, px27ax	nesC	BSD	BLIP
OpenWSN	Monolithic	Cooperative[e]	Event-driven	Class 0 – 2	MSP430, ARM Cortex-M	C	BSD	OpenWSN
nuttX	Monolithic or microkernel	Preemptive (priority-based or round robin)	Multithreading	Class 1 + 2	AVR, MSP430, ARM7, ARM9, ARM Cortex-M, MIPS32, x86, 8052, Renesas	C	BSD	native
eCos	Monolithic RTOS	Preemptive	Multithreading	Class 1 + 2	ARM, IA-32, Motorola, MIPS ...	C	eCos License[f]	lwIP, BSD
uClinux	Monolithic	Preemptive	Multithreading	>Class 2	Motorola, ARM7, ARM Cortex-M, Atari	C	GPLv2	Linux
ChibiOS/RT	Microkernel	Preemptive	Multithreading	Class 1 + 2	AVR, MSP430, ARM Cortex-M	C	Triple License[g]	None
CoOS	Microkernel RTOS	Preemptive	Multithreading	Class 2	ARM Cortex-M	C	BSD	None
nanoRK	Monlothic (resource kernel)	Preemptive	Multithreading	Class 0	AVR, MSP430,	C	Dual License	None
Nut/OS	Monolithic	Cooperative	Multithreading	Class 0 + 1	AVR, ARM	C	BSD	native

TinyOS



University of California in Berkeley – open source – TinyOS Alliance

Current version 2.1.2 (August 20, 2012)

35.000 downloads a year[1]

Two basic principles: (1) smaller code and data (2) bug prevention

Kernel uses less than 400 bytes of program memory

10:1 ratio of ROM:RAM

Power saving states for MCU and components (radio)

Event-driven programming model

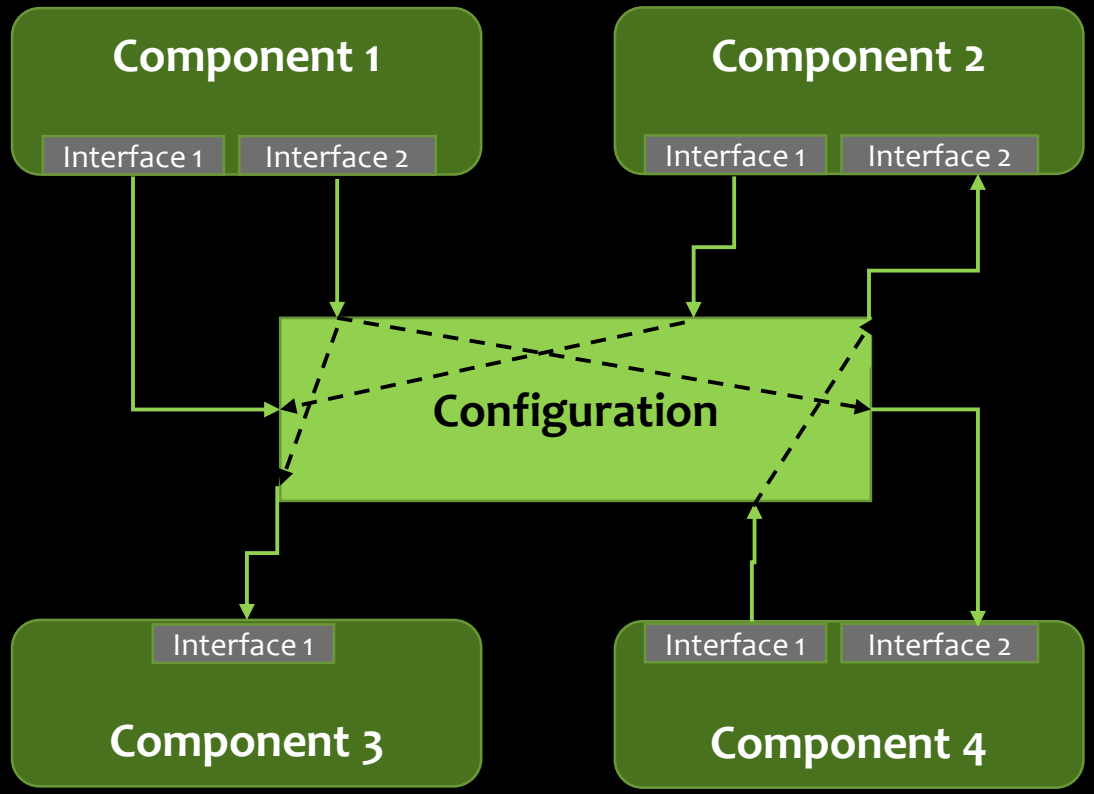
Written in NesC language

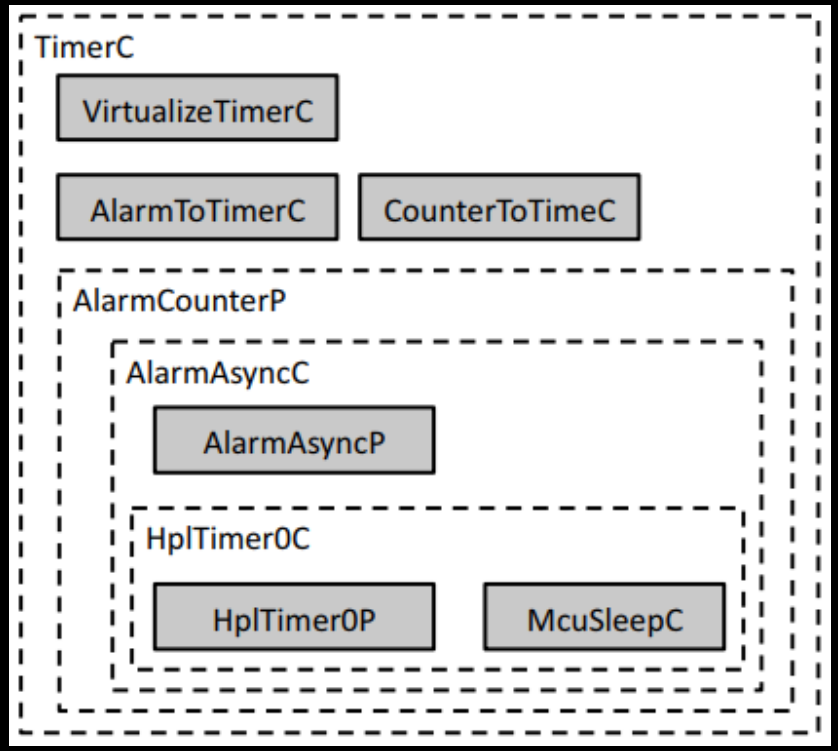
1. http://www.tinyos.net

[5] Levis (2012)
[6] Reusing (2012)

TinyOS

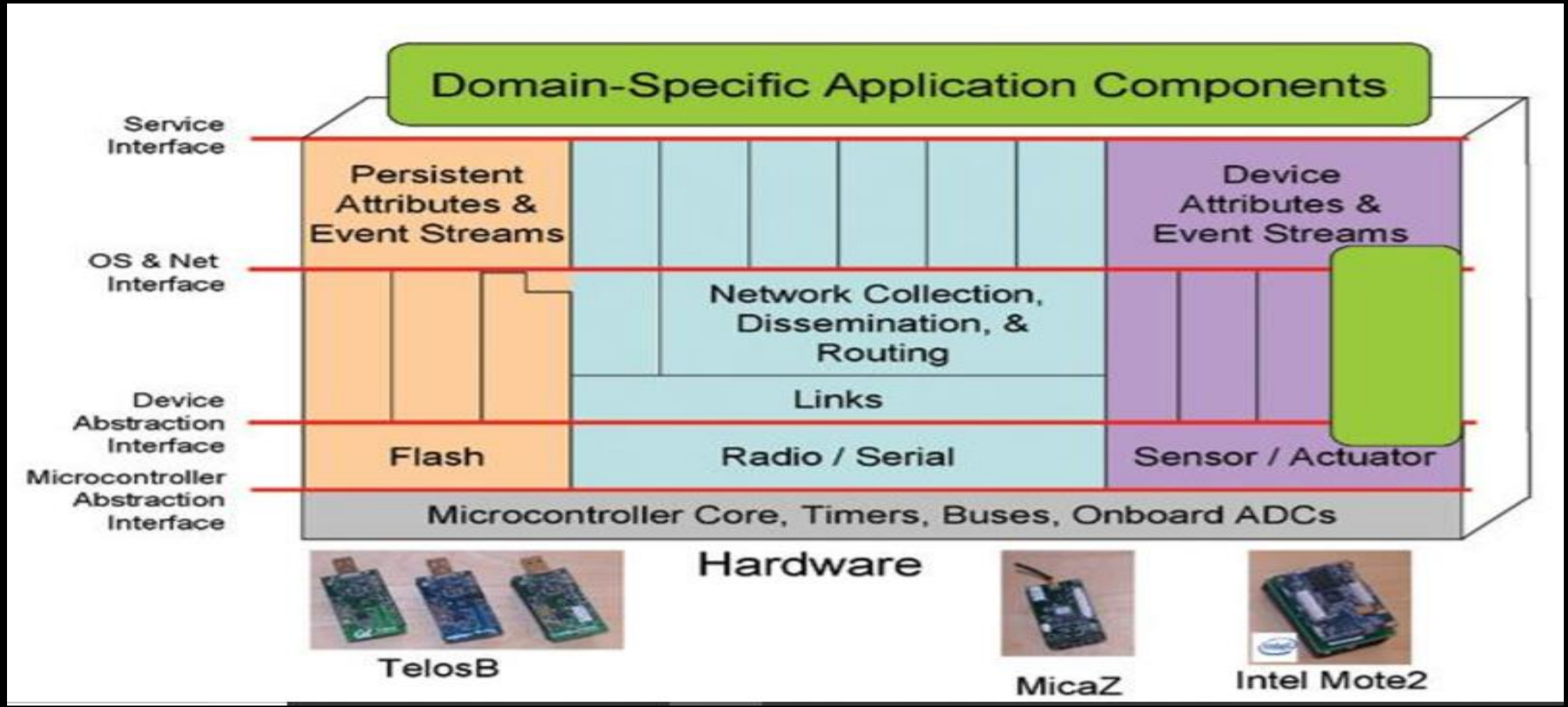
Components: **Commands, events, fixed-size frames, and tasks**





[5] Levis (2012)
[6] Reusing (2012)

TinyOS

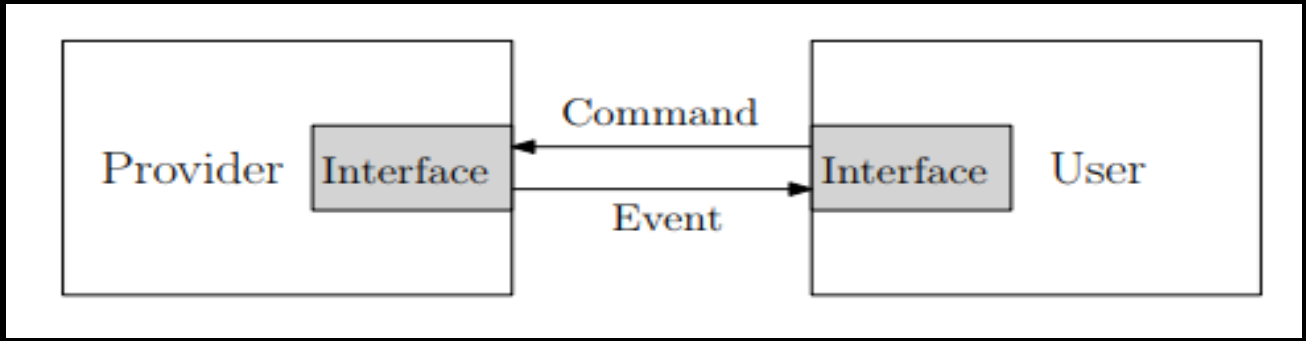


[3] Farroq & Kunz (2011)

TinyOS

Commands are non-blocking requests made to the low level components

Split-phase execution model



Event-handlers like tasks can store information in its frame, assign tasks, issue high-level events, or call low-level commands.

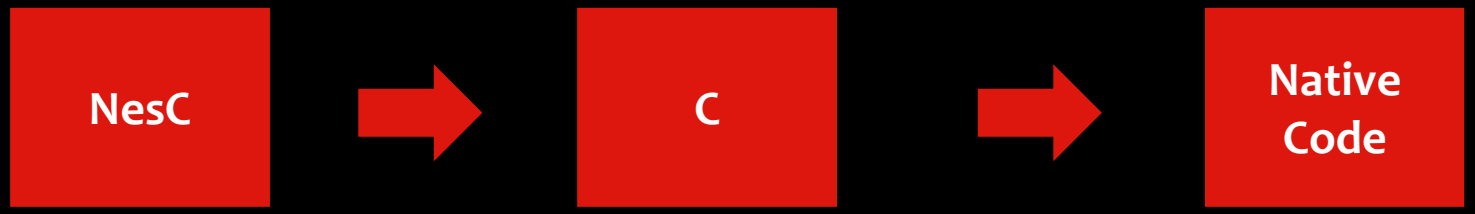
Concurrency is achieved with tasks

[6] Reusing (2012)
[8] Chien et al (2011)

TinyOS

- FIFO scheduling algorithm
- Implementation of Earliest Deadline First scheduling for RT applications
- Support for threads – TOS Threads
- Shared-stack model
- No separation between kernel and user space

Toolchain



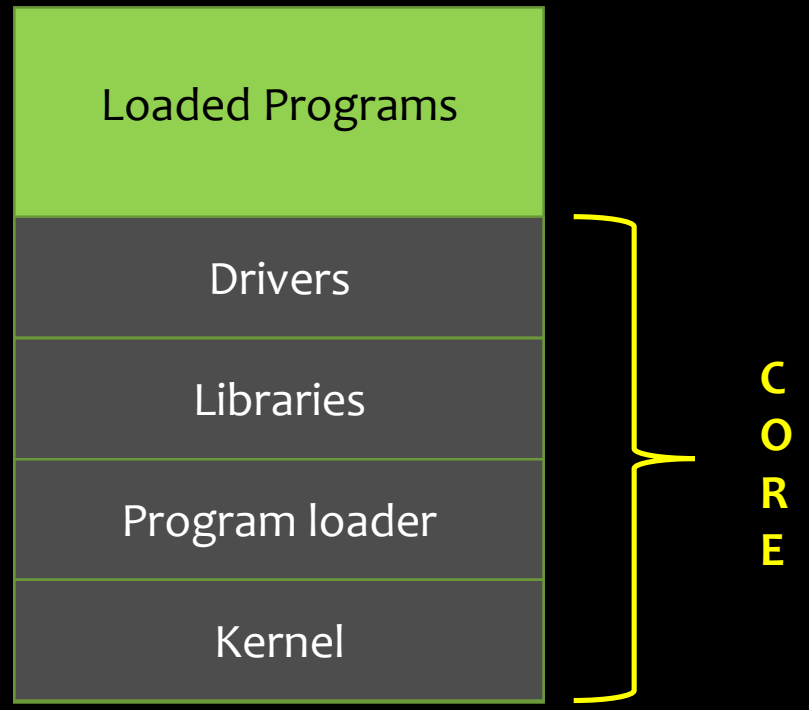
[5] Levis (2012)
[6] Reusing (2012)

Contiki

Swedish Institute of Computer Science

Current version 3.0 (25/08/2015)

- Dynamic loading and unloading of code at run-time
- Core distributed in a single image / loaded programs distributed independently
- Possibility of multi-threading atop of an event-driven kernel
- Static allocation of memory



[4] Hahm (2012)
[6] Reusing (2012)

Contiki

- Application programs and Services

  Process implements event handler and [poll handler function]

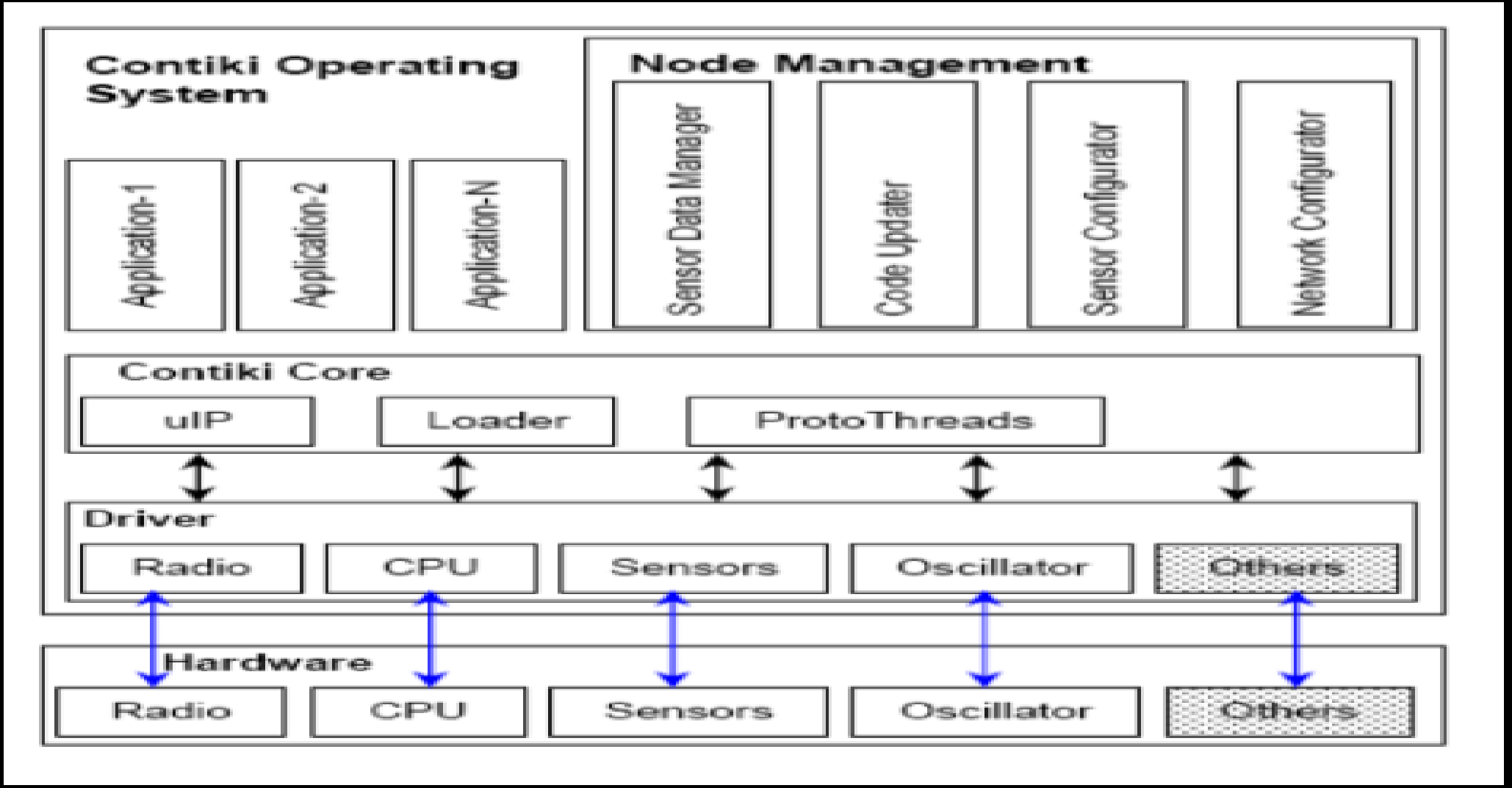
  Execution only through those handlers

  Process has to keep its state information between calls

- Scheduler can be configured to call the polling handlers of all processes that implement one – in periodic intervals
- Run to completion
- Asynchronous and Synchronous events
- Protothreads atop of kernel or a multithreading model explicitly linked if the application uses it.

[6] Reusing (2012)

Contiki


Contiki Operating System
Node Management
Application-1
Application-2
Application-N
Sensor Data Manager
Code Updater
Sensor Configurator
Network Configurator
Contiki Core
uIP
Loader
ProtoThreads
Driver
Radio
CPU
Sensors
Oscillator
Others
Hardware
Radio
CPU
Sensors
Oscillator
Others

Conclusion

One size fits all approach is not appropriate for WSN OSs

One size fits most approach is also very difficult to achieve imposed by the constrained environment of WSN

Both analyzed OSs are compliant to most of the given requirements even with significant differences in design.

Research fields

Real-time capabilities
Radio duty cycling and mote synchronization
Energy efficiency / Harvesting
Network connectivity
Security and safety
Small memory footprint
Heterogeneous device support
Intelligent IoT / Local Processing
Local storage
Programming tools / strategies for adoption

Bibliography

[1] Adam Dunkels. Poster Abstract: Rime — A Lightweight Layered Communication Stack for Sensor Networks.

[2] Farhana Javed, Muhamamd Khalil Afzal, Muhammad Sharif, Byung-Seo Kim. (2018) **Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review**. IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 20, NO. 3, THIRD QUARTER 2018

[3] Muhammad Omer Farooq and Thomas Kunz (2011) **Operating Systems for Wireless Sensor Networks: A Survey**. Sensors 11, 5900-5930; doi:10.3390/s110605900

[4] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes (2016) **Operating Systems for Low-End Devices in the Internet of Things: A Survey**. IEEE INTERNET OF THINGS JOURNAL, VOL. 3, NO. 5, October 2016

[5] Philip Levis (2012) **Experiences from a Decade of TinyOS Development**. 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)

[6] Tobias Reusing (2012) **Comparison of Operating Systems TinyOS and Contiki**. Seminar SN SS2012. Network Architectures and Services, August 2012

[7] Bormann C, Ersue M, Keranen, A. Ericsson (2014) RFC7228 Terminology for Constrained-Node Networks. https://tools.ietf.org/html/rfc7228

[8] Chien et al. (2011) A comparative study on operating system for wireless sensor networks. ICASIS International Conference on Advanced Computer Science and Information Systems, Proceedings 2011.