Internet of Things

Lecture 8 - IoT Operating Systems

IoT Demands

- Low Power
- Low Cost
- Low memory footprint
- IPv6, 6LoWPAN Adaptation Layer
- Routing protocol
- Packet loss - retransmission
- Lightweight application layer protocols (CoAP, MQTT)

Low-end IoT Devices

- Resource constrained devices
- IETF classification:
- Class 0
  - <<10 kB of RAM and <<100 kB Flash
  - Sensor node from WSN
- Class 1
  - ~10 kB of RAM and ~100 kB Flash
  - rich applications, advanced features, routing and secure protocols
- Class 2
  - more resources

Low-end IoT Devices

- Class 0
  - Not suitable to run an OS
  - Bare metal and hardware-specific software
- Class 1 and above
  - Router, host, server
  - Networking stack, reprogrammable applications
- Software primitives enabling easy hardware independent code production
- APIs for large-scale software development, deployment, and maintenance
- Software primitives provided by OSes

IoT OS Requirements

- Small memory footprint
  - optimized libraries, efficient data structures
  - tradeoff between: performance, a convenient API, and a small OS memory footprint
- Support for Heterogeneous Hardware
  - 8-bit, 16-bit, 32-bit architectures
  - different amounts of RAM & ROM

IoT OS Requirements

- Network Connectivity
  - various communication technologies
  - communication between devices and over the Internet
  - network stacks based on IP protocols
- Energy Efficiency
  - make use of energy saving options
  - use hardware cryptographic operations
  - provide energy saving options to upper layers
  - duty cycling, minimize number of tasks executed periodically

IoT OS Requirements

- Real-time capabilities
  - precise timing and timely execution
  - Real-time OS (RTOS)
  - guarantee worst-case execution times and interrupt latencies
  - kernel functions operate with a deterministic run time
- Security
  - high security and privacy standards
  - cryptographic libraries and security protocols
  - security updates
  - open source software

OS Design for IoT - Architecture

General Architecture and Modularity

- Exokernel
  - few abstractions between the application and the hardware
  - avoiding resource conflicts and checking access levels
- Microkernel
  - minimalistic set of features
  - device drivers run in user space
  - flexibility and robustness
    - an error in a driver does not affect the whole system

OS Design for IoT - Architecture

General Architecture and Modularity

- Monolithic
  - all components are developed together
  - device drivers run in kernel space
    - an error in a driver may affect the whole system
  - OS size may be too big for some IoT devices
- Hybrid
  - compromise between microkernel and monolithic

OS Design for IoT - Scheduling

Scheduling Model

- Scheduler impacts:
  - energy consumption
  - real-time capabilities
  - programming model
- 2 types: preemptive & cooperative
- Different schedulers available - select at build time
- Cooperative scheduler
  - each thread is responsible to yield control

OS Design for IoT - Scheduling

Scheduling Model

- Preemptive scheduler
  - can interrupt any task to allow another task to be executed
  - time slices
  - periodic timer tick
    - prevents the IoT device to enter the deepest power-save mode

OS Design for IoT - Memory

Memory Allocation

- Static allocation
  - over-provisioning
  - makes the system less flexible
- Dynamic allocation
  - Complicated design
  - Malloc is time-wise nondeterministic => breaks real time guarantees
  - handle out-of-memory situations at runtime
  - memory fragmentation

OS Design for IoT - Network Stack

Network Buffer Management

- chunks of memory shared between layers
- copy memory or pass pointers
- central memory manager

OS Design for IoT - Programming Model

Programming Model

- event-driven systems
  - every task has to be triggered by an event
  - event loop, shared stack
  - more efficient use of memory
- multithreaded systems
  - run each task in its own thread context
  - communicate using IPC
  - allow easy development of applications

OS Design for IoT - Drivers

Driver Model and Hardware Abstraction Layer

- sensors, actuators, ADC, SPI, I2C, CAN bus, serial lines, GPIOs
- driver interface
- Hardware Abstraction layer
  - well-defined interface to CPU, memory, interrupt handling
  - easy porting
- Overhead

Most popular OSes

- TinyOS
- Contiki OS
- RIOT OS
- FreeRTOS
- NuttX

TinyOS

- From 2000
- Developed for WSN (8-bit, 16-bit arch) - class 0 devices
- Monolithic kernel
- Cooperative scheduler
- Event driven
- Components virtually wired, as described by configurations
- It provides algorithms, protocols, device drivers, file systems and a shell

TinyOS

- Dialect of the C programming language called nesC
- BLIP networking stack includes 6LoWPAN
- Memory efficiency by reducing linked code to a minimum
- Lack of a large developer community
- BSD license

Contiki

- Developed by Adam Dunkels in 2003
- Supports a wide range of resource constrained devices
  - 8-bit AVR, 16- and 20-bit MSP430, 32-bit ARM Cortex M3
- Monolithic architecture
  - core system + processes => a single image
  - all processes share the same memory space and privileges with the core system
- Cooperative scheduling

Contiki

- Memory allocation
  - static allocation
  - third-party modules for dynamic allocation, malloc
- 2 network stacks: uIP & Rime
- uIP (micro IP)
  - 6LoWPAN, IPv4, IPv6, IPv6 neighbor discovery, IPv6 multicasting, RPL, TCP, UDP
  - Queuebuf allocates packet buffers from a static pool of memory
- Protothreads - lightweight, cooperative

Contiki

- C programming language
- Hardware Abstraction Layer (HAL)
  - hardware-specific functionality is put in separate components
  - common API for using that hardware
- Cooja simulator
  - network simulation
  - cycle-accurate emulation
  - debugging features: setting breakpoints, read/write to memory addresses, single-stepping through instructions

Contiki

- Extra features:
  - shell, file system, database management system
  - runtime dynamic linking, cryptography libraries
  - fine-grained power tracing tool
- Real-world deployments
- Commercial IoT products
- Academic research
- Large community of developers
- GitHub repo

Contiki

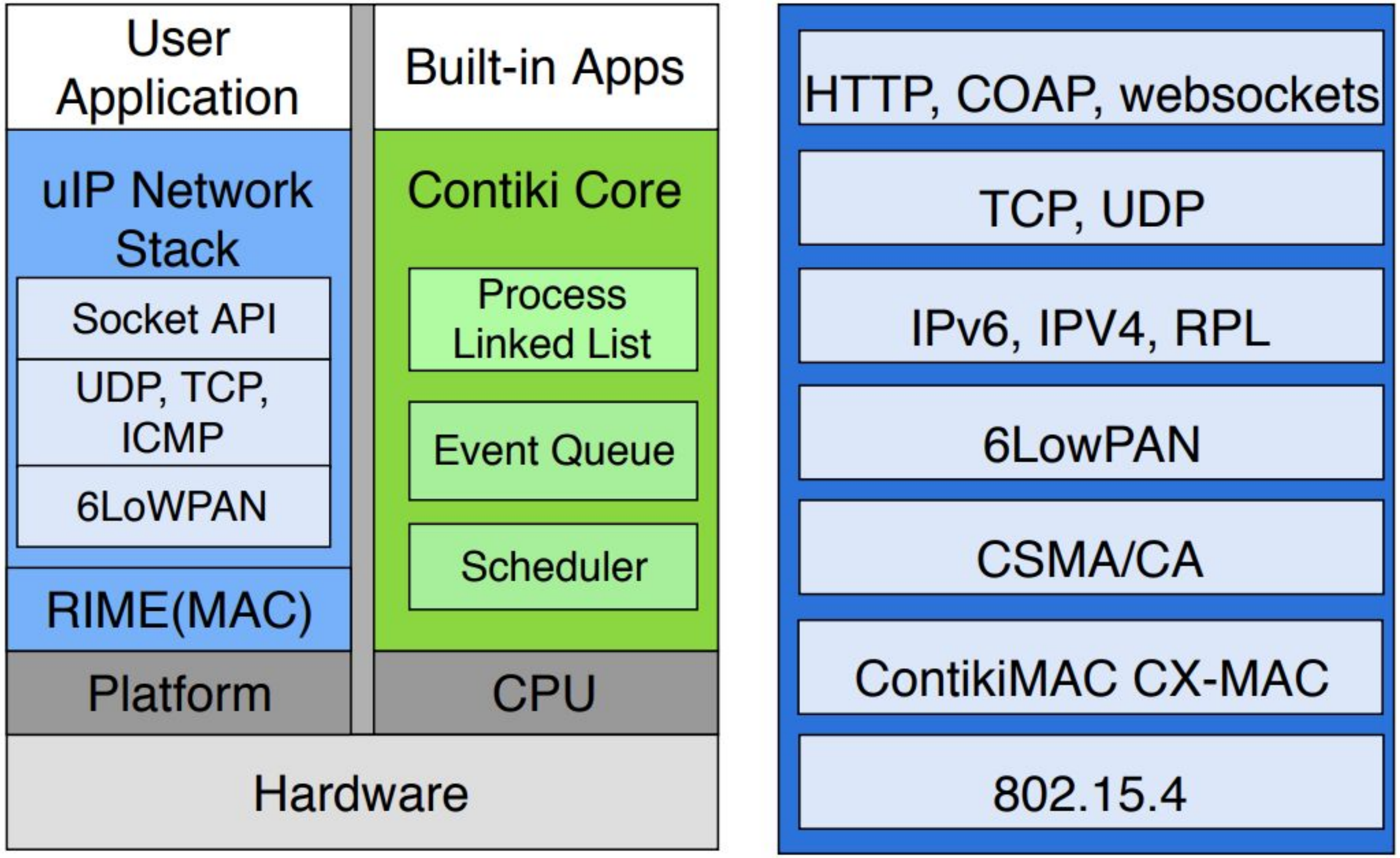


Figure 2.1: Contiki-OS architecture stack and supported IoT stack.

Source: Ângelo André Oliveira Ribeiro, "Deploying RIOT Operating System on a Reconfigurable Internet of Things End-device"

RIOT

- From 2013
- Microkernel architecture
- For real-time WSNs
- 8-bit, 16-bit, 32-bit MCUs
- full multithreading (~Linux)
    - memory overhead
    - efficient context switch - small number of CPU cycles
    - reduced TCB
    - memory-passing IPC between threads
    - multi-threading may be removed if necessary

RIOT

- Tickless scheduler (for energy efficiency)
  - switch to idle thread
  - deepest sleep mode
  - external or kernel-generated interrupts wake up the system
- Preemptive scheduler based on fixed priorities
- Memory allocation
  - both static & dynamic
  - kernel uses only static allocation
  - enforcing constant periods for kernel tasks
  - dynamic allocation only in userspace

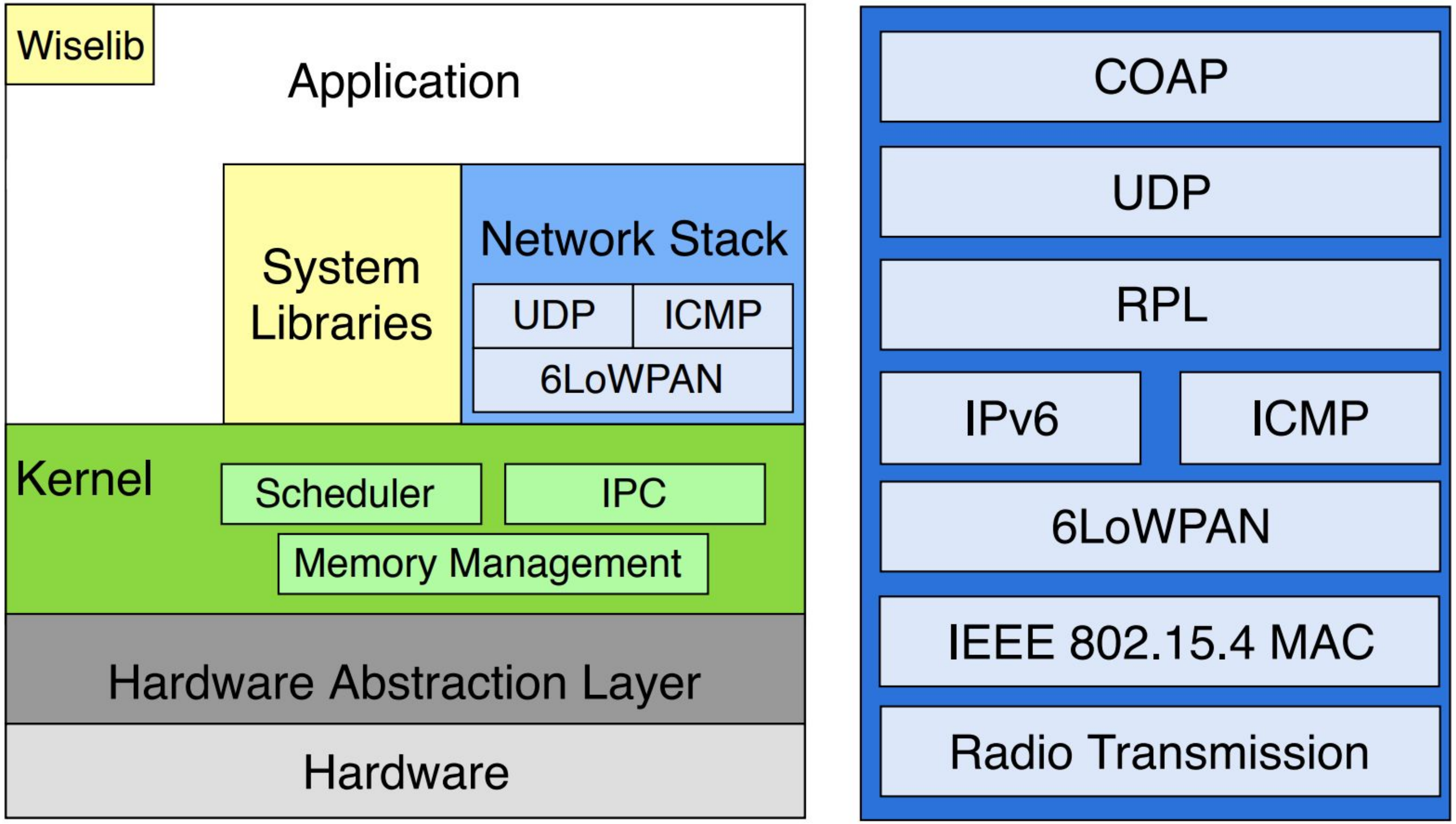
RIOT

- Multiple network stacks
- **GNRC** network stack with IP protocols
  - 6LoWPAN, IPv6, RPL, UDP, CoAP
  - centralized network buffer structure
  - pointers passed between layers
- Extra features: shell, crypto libraries, data structures
- Kernel written in ANSI C language, assembly
- Apps & libraries written in C or C++

RIOT

- Real-time guarantees
  - zero latency interrupt handlers, minimum context switching times
- Hardware abstraction layer
- Standard debugging tools (GDB and Valgrind)
- Run OS as process over Linux & Mac OS
- Cooja simulator
- POSIX standard interfaces
- Open source community
- LGPL license

RIOT



Source: Ângelo André Oliveira Ribeiro, "Deploying RIOT Operating System on a Reconfigurable Internet of Things End-device"

Figure 2.3: RIOT-OS architecture stack and supported IoT stack.

FreeRTOS

- Developed by Richard Barry in 2002
- GPL license
- Real-time OS
- Preemptive microkernel
- Support for multithreading
- Small, simple, portable, easy to use
- More a threading library than a full-fledged OS
- Thread handling, mutexes, semaphores, software timers

FreeRTOS

- Preemptive, priority based round-robin scheduler
  - periodic timer tick interrupt
  - tickless mode
- Real-time guarantees
  - only deterministic operations in critical section and interrupt
- Message queues as IPC
- Does not provide a networking stack
  - Various libraries for networking

FreeRTOS

- 5 memory allocation schemes
    1. allocate only
    2. simple and fast allocate & free
    3. malloc() and free() from C library
    4. more complex allocate & free (memory coalescence)
    5. even more complex - allows heap span over memory sections
- OS implemented in C language, C++ for apps
- Does not define a portable driver model
    - works with vendor BSPs

NuttX

- Developed by Gregory Nutt in 2007
- Apache Software Foundation
- POSIX and ANSI compliance
- MCUs ranging from 8-bit to 32-bit architectures
- Built as microkernel or monolithic kernel
- Highly modular
- Real-time capabilities
- Tickless scheduler

NuttX

- Scheduler
  - Fully preemptible
  - Realtime scheduling (FIFO, RR, SPORADIC)
  - Tickless operation support (lower power consumption)
- Virtual File System (VFS)
- Loadable kernel modules
- Symmetric Multi-Processing (SMP)
- Pseudo-terminals (PTY) and I/O redirection
- On-demand paging

NuttX

- Inspiration from Linux/Unix:
  - VFS
  - MTD
  - PROCFS
  - NuttShell
- Very customizable
- Small memory footprint
- Network stack - IPv4, IPv6, UDP, 6LoWPAN
- Apache License 2.0

OS Comparison

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
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

Source: O. Hahm, E. Baccelli, H. Petersen and N. Tsiftes, "Operating Systems for Low-End Devices in the Internet of Things: A Survey," in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 720-734, Oct. 2016.

OS Comparison

Name	Category	MCU w/o MMU	< 32 kB RAM	6LoWPAN	RTOS scheduler	HAL	Energy-efficient MAC layers
Contiki	Event-driven	✓	✓	✓	✗	✓	✓
RIOT	Multithreading	✓	✓	✓	✓	✓	✗[a]
FreeRTOS	RTOS	✓	✓	✗[b]	✓	✗	✗[c]

Source: O. Hahm, E. Baccelli, H. Petersen and N. Tsiftes, "Operating Systems for Low-End Devices in the Internet of Things: A Survey," in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 720-734, Oct. 2016.

OS Comparison

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading	MCU w/o MMU	Modularity	Real-Time
Contiki	<2kB	<30kB	○	✘	○	✓	○	○
Tiny OS	<1kB	<4kB	✘	✘	○	✓	✘	✘
Linux	~1MB	~1MB	✓	✓	✓	✘	○	○
RIOT	~1.5kB	~5kB	✓	✓	✓	✓	✓	✓

TABLE I

KEY CHARACTERISTICS OF CONTIKI, TINYOS, LINUX, AND RIOT. (✓) FULL SUPPORT, (○) PARTIAL SUPPORT, (✘) NO SUPPORT. THE TABLE COMPARES THE OS IN MINIMUM REQUIREMENTS IN TERMS OF RAM AND ROM USAGE FOR A BASIC APPLICATION, SUPPORT FOR PROGRAMMING LANGUAGES, MULTI-THREADING, MCUS WITHOUT MEMORY MANAGEMENT UNIT (MMU), MODULARITY, AND REAL-TIME BEHAVIOR.

Source: E. Baccelli, O. Hahm, M. Günes, M. Wählisch and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2013, pp. 79-80.

Bibliography

- O. Hahm, E. Baccelli, H. Petersen and N. Tsiftes, "Operating Systems for Low-End Devices in the Internet of Things: A Survey," in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 720-734, Oct. 2016. (link)
- Levis, P. et al. (2005). TinyOS: An Operating System for Sensor Networks. In: Weber, W., Rabaey, J.M., Aarts, E. (eds) Ambient Intelligence. Springer (link)
- A. Dunkels, B. Gronvall and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," 29th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 455-462 (link)
- E. Baccelli, O. Hahm, M. Günes, M. Wählisch and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WOKSHOPS), 2013, pp. 79-80. (link)

Bibliography

- Ângelo André Oliveira Ribeiro, “Deploying RIOT Operating System on a Reconfigurable Internet of Things End-device”, Master Thesis, 2017. (link)
- Mastering the FreeRTOS™ Real Time Kernel (link)
- http://www.tinyos.net/
- http://www.contiki-os.org/
- https://www.riot-os.org/
- https://www.freertos.org/
- https://nuttx.apache.org/docs/latest/
- https://nuttx.apache.org/docs/latest/introduction/about.html