1 | Survey plafforms

1 IoT end devices

1.1 Software platform

The operating system is the foundation of the IoT technology as it provides the functions for the connectivity between the nodes. However, different types of nodes need different levels of OS complexity; a passive node generally only needs the communication stack and is not in need of any threading capabilities, as the program can handle all logic in one function. Active nodes and border routers need to have a much more complex OS, as they need to be able to handle several running threads or processes, e.g. routing, data collection and interrupts. To qualify as an OS suitable for the IoT, it needs to meet the basic requirements: Low Random-access memory (RAM) footprint Low Read-only memory (ROM) footprint Multi-tasking Power management (PM) Soft real-time These requirements are directly bound to the type of hardware designed for the IoT. As this type of hardware in general needs to have a small form factor and a long battery life, the on-board memory is usually limited to keep down size and energy consumption. Also, because of the limited amount of memory, the implementation of threads is usually a challenging task, as context switching needs to store thread or process variables to memory. The size of the memory also directly affects the energy consumption, as memory in general is very power hungry during accesses. To be able reduce the energy consumption, the OS needs some kind of power management. The power management does not only let the OS turn on and off peripherals such as flash memory, I/O, and sensors, but also puts the MCU itself in different power modes. As the nodes can be used to control and monitor consumer devices, either a hard or soft real-time OS is required. Otherwise, actions requiring a close to instantaneous reaction might be indefinitely delayed. Hard real-time means that the OS scheduler can guarantee latency and execution time, whereas Soft real-time means that latency and execution time is seen as real-time but can not be guaranteed by the scheduler. Operating systems that meet the above requirements are compared in table 2.1 and 2.2.

Contiki

Contiki is a embedded operating system developed for IoT written in C [12]. It supports a broad range of MCUs and has drivers for various transceivers. The OS does not only support TCP/IPv4 and IPv6 with the uIP stack [9], but also has support for the 6LoWPAN stack and its own stack called RIME. It supports threading with a thread system called Photothreads [13]. The threads are stack-less and thus use only two bytes of memory per thread; however, each thread is bound to one function and it only has permission to control its own execution. Included in Contiki, there is a range

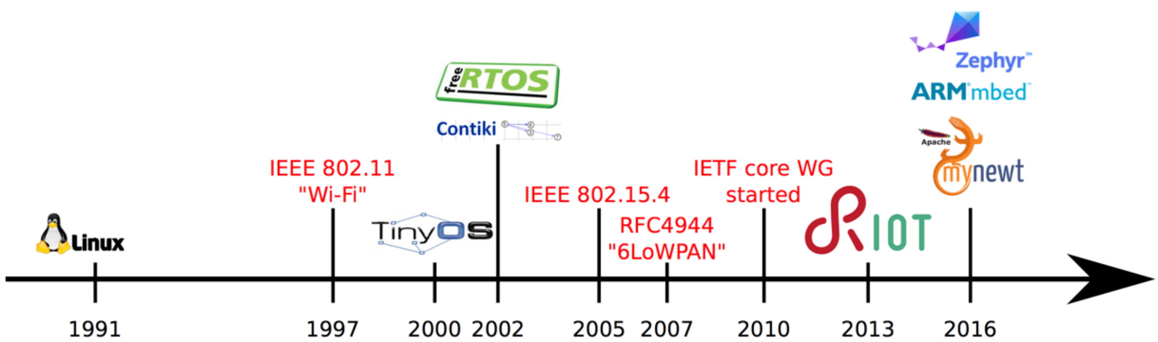

Figure 1. .

of applications such as a HTTP, Constrained Application Protocol (CoAP), FTP, and DHCP servers, as well as other useful programs and tools. These applications can be included in a project and can run simultaneously with the help of Photothreads. The limitations to what applications can be run is the amount of RAM and ROM the target MCU provides. A standard system with IPv6 networking needs about 10 kB RAM and 30 kB ROM but as applications are added the requirements tend to grow.

Contiki is an open source operating system for the Internet of Things. Contiki connects tiny low-cost, low-power micro-controllers to the Internet.

2k RAM, 60k ROM; 10k RAM, 48K ROM Portable to tiny low-power micro-controllers I386 based, ARM, AVR, MSP430, ... Implements uIP stack IPv6 protocol for Wireless Sensor Networks (WSN) Uses the protothreads abstraction to run multiple process in an event based kernel. Emulates concurrency Contiki has an event based kernel (1 stack) Calls a process when an event happens

Contiki size One of the main aspect of the system, is the modularity of the code. Besides the system core, each program builds only the necessary modules to be able to run, not the entire system image. This way, the memory used from the system, can be reduced to the strictly necessary. This methodology makes more practical any change in any module, if it is needed. The code size of Contiki is larger than that of TinyOS, but smaller than that of the Mantis system. Contiki's event kernel is significantly larger than that of TinyOS because of the different services provided. While the TinyOS event kernel only provides a FIFO event queue scheduler , the Contiki kernel supports both FIFO events and poll handlers with priorities. Furthermore, the flexibility in Contiki requires more run-time code than for a system like TinyOS, where compile time optimization can be done to a larger extent.

The documentation in the doc folder can be compiled, in order to get the html wiki of all the code. It needs doxygen installed, and to run the command make html. This will create a new folder, doc/hmtl, and in the index.html file, the wiki can be opened.

Contiki Hardware Contiki can be run in a number of platforms, each one with a different CPU. Tab.7 shows the hardware platforms currently defined in the Contiki code tree. All these platforms are in the platform folder of the code.

Kernel structure

RIOT

RIOT is a open source embedded operating system supported by Freie Universität Berlin, INIRA, and Hamburg University of Applied Sciences [14]. The kernel is written in C but the upper layers support C++ as well. As the project originates from a project with real-time and reliability requirements, the kernel supports hard real-time multi-tasking scheduling. One of the goals of the project is to make the OS completely POSIX compliant. Overhead for multi-threading is minimal with less than 25 bytes per thread. Both IPv6 and 6LoWPAN is supported together with UDP, TCP, and IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL); and CoAP and Concise Binary Object Representation (CBOR) are available as application level communication protocols.

TinyOS

TinyOS is written in Network Embedded Systems C (nesC) which is a variant of C [15]. nesC does not have any dynamic memory allocation and all program paths are available at compile-time. This is manageable thanks to the structure of the language; it uses modules and interfaces instead of functions [16]. The modules use and provide interfaces and are interconnected with configurations; this procedure makes up the structure of the program. Multitasking is implemented in two ways: trough tasks and events. Tasks, which focus on computation, are non-preemptive, and run until completion. In contrast, events which focus on external events i.e. interrupts, are preemptive, and have separate start and stop functions. The OS has full support for both 6LoWPAN and RPL, and also have libraries for CoAP.

freeRTOS

One of the more popular and widely known operating systems is freeRTOS [17]. Written in C with only a few source files, it is a simple but powerful OS, easy to overview and extend. It features two modes of scheduling, pre-emptive and co-operative, which may be selected according to the requirements of the application. Two types of multitasking are featured: one is a lightweight Co-routine type, which has a shared stack for lower RAM usage and is thus aimed to be used on very small devices; the other is simply called Task, has its own stack and can therefore be fully pre-empted. Tasks also support priorities which are used together with the pre-emptive scheduler. The communication methods supported out-of-the-box are TCP and UDP.

Summary and conclusion

	LiteOS	Nano-RK	MANTIS	Contiki
Architecture	Monolithic	Layered	Modular	Modular
Scheduling Memory	Round Robin	Monotonic harmonized	Priority classes	Interrupts execute w.r.t.
Network	File	Socket abstraction	At Kernel COMM layer	uIP, Rime
Virtualization and Completion	Synchronization primitives	Serialized access semaphores	Semaphores	Serialized, Access
Multi threading	✓	✓	✗	✓
Dynamic protection	✓	✗	✓	✓
Memory Stack	✓	✗	✗	✗

Table 1.1. Common operating systems used in IoT environment **al-fuqaha__internet__24**

OS	Contiki	MANTIS	Nano-RK	LiteOS
Architecture	Modular	Modular	Layered	Monolithic
Multi threading	✓	✗	✓	✓
Scheduling	Interrupts	Priority	Monotonic	Round Robin
	execute w.r.t.	classes	harmonized	
Dynamic Memory	✓	✓	✗	✓
Memory protection	✗	✗	✗	✓
Network Stack	uIP/Rime	At Kernel/COMM layer	Socket/abstraction	file
Virtualization and Completion	Serialized/Access	Semaphores	Serialized/semaphores	Synchronization/primitives

Table 1.2. Common operating systems used in IoT environment **al-fuqaha__internet__24**

1.2 Hardware platform

Processing Unit

Even though the hardware is in one sense the tool that the OS uses to make IoT possible, it is still very important to select a platform that is future-proof and extensible. To be regarded as an extensible platform, the hardware needs to have I/O connections that can be used by external peripherals. Amongst the candidate interfaces are Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I 2 C), and Controller Area Network (CAN). These interfaces allow developers to attach custom-made PCBs with sensors for monitoring or actuators for controlling the environment. The best practice is to implement an extension socket with a well-known form factor. A future-proof device is specified as a device that will be as attractive in the future as it is today. For hardware, this is very hard to achieve as there is constant development that follows Moores Law [4]; however, the most important aspects are: the age of the chip, its expected remaining lifetime, and number of current implementations i.e. its popularity. If a device is widely used by consumers, the lifetime of the product is likely to be extended. One last thing to take into consideration is the product family; if the chip belongs to a family with several members the transition to a newer chip is usually easier.

OpenMote OpenMote is based on the Ti CC2538 System on Chip (SoC), which combines an ARM Cortex-M3 with a IEEE 802.15.4 transceiver in one chip [18, 19]. The board follows the XBee form factor for easier extensibility, which is used to connect the core board to either the OpenBattery or OpenBase extension boards [20, 21]. It originates from the CC2538DK which was used by Thingsquare to demo their Mist IoT solution [22]. Hence, the board has full support for Contiki, which is the foundation of Thingsquare. It can run both as a battery-powered sensor board and as a border router, depending on what extension board it is attached to, e.g OpenBattery or OpenBase. Furthermore, the board has limited support but ongoing development for RIOT and also full support for freeRTOS.

MSB430-H The Modular Sensor Board 430-H from Freie Universität Berlin was designed for their ScatterWeb project [23]. As the university also hosts the RIOT project, the decision to support RIOT was natural. The main board has a Ti MSP430F1612 MCU [24], a **Ti CC1100 transceiver**, and a battery slot for dual AA batteries; it also includes a SHT11 temperature and humidity sensor and a MMA7260Q accelerometer to speed up early development. All GPIO pins and buses are connected to external pins for extensibility. Other modules with new peripherals can then be added by making a PCB that matches the external pin layout.

Zolertia As many other Wireless Sensor Network (WSN) products, the Zolertia Z1 builds upon the MSP430 MCU [25, 26]. The communication is managed by the Ti CC2420 which operates in the 2.4 GHz band. The platform includes two sensors: the SHT11 temperature and humidity sensor and the MMA7600Q accelerometer. Extensibility is ensured with: two connections designed especially for external sensors, an external connector with USB, Universal asynchronous **receiver/transmitter (UART)**, SPI, and I 2 C.

Radio Unit

Lora Tranceiver To limit the complexity of the radio unit:

- limiting message size: maximum application payload size between 51 and 222 bytes, depending on the spreading factor
- using simple channel codes: Hamming code
- supporting only half-duplex operation
- using one transmit-and-receive antenna
- on-chip integrating power amplifier (since transmit power is limited)

limiting message size: maximum application payload size between 51 and 222 bytes, depending on the spreading factor using simple channel codes: Hamming code supporting only half-duplex operation using one transmit-and-receive antenna on-chip integrating power amplifier (since transmit power is limited)

Ref	**Module**	**Frequency MHz**	**Tx power**	**Rx power**	**Sensi-tivity**	**Chan-nels**	**Dis-tance**
libelium_waspmote_2017	**Semtech SX1272**	863-870 (EU) 902-928 (US)	14 dBm	dBm	-134 dBm	8 13	22+ km
libelium_waspmote_2017	**rn2483**						

Table 1.3

Sensing Unit

GPS

Humidity

Temperature

1.3 Summary and discussion

1.4 Lora

Preamble		Sync msg	PHY Header	PHDR-CRC	PHY Payload													
Modulation	length	Sync msg	PHY Header	PHDR-CRC	MAC Header			MAC Payload										
Modulation	length	Sync msg	PHY Header	PHDR-CRC	MType	RFU	Major	Frame Header									FPort	**Fr Pa**
Modulation	length	Sync msg	PHY Header	PHDR-CRC	MType	RFU	Major	Dev Address		FCtrl					FCnt	FOpts	FPort	**Fr Pa**
Modulation	length	Sync msg	PHY Header	PHDR-CRC	MType	RFU	Major	NwkID	NwkAddr	ADR	ADRACKReq	ACK	FPending /RFU	FOptsLen	FCnt	FOpts	FPort	**Fr Pa**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

0) **Modulation :**
   - Lora: 8 Symbols, 0x34 (Sync Word)
   - FSK: 5 Bytes, 0xC194C1 (Sync Word)
1. **Length :**
2. **Sync msg :**
3. **PHY Header :** It contains:
   - The Payload length (Bytes)
   - **The Code rate**
   - Optional 16bit CRC for payload
4. **Phy Header :** CRC It contains CRC of Physical Layer Header
5. **MType :** is the message type (uplink or a downlink)
   - whether or not it is a confirmed message (reqst ack)
   - 000 Join Request
   - 001 Join Accept
   - 010 Unconfirmed Data Up
   - 011 Unconfirmed Data Down
   - 100 Confirmed Data Up
   - 101 Confirmed Data Down
   - 110 RFU
   - 111 Proprietary
6. **RFU :** Reserved for Future Use
7. **Major :** is the LoRaWAN version; currently, only a value of zero is valid
   - 00 LoRaWAN R1
   - 01-11 RFU
8. **NwkID :** the short address of the device (Network ID): 31th to 25th
9. **NwkAddr :** the short address of the device (Network Address): 24th to 0th
10. **ADR :** Network server will change the data rate through appropriate MAC commands
    - 1 To change the data rate
    - 0 No change
11. **ADRACKReq :** (Adaptive Data Rate ACK Request): if network doesn't respont in 'ADR-ACK-DELAY' time, end-device switch to next lower data rate.
    - 1 if (ADR-ACK-CNT) >= (ADR-ACK-Limit)
    - 0 otherwise
12. **ACK :** (Message Acknowledgement): If end-device is the sender then gateway will send the ACK in next receive window else if gateway is the sender then end-device will send the ACK in next transmission.
    - 1 if confirmed data message
    - 0 otherwise
13. **FPending↓ /RFU ↑ :** (Only in downlink), if gateway has more data pending to be send then it asks end-device to open another receive window ASAP
    - 1 to ask for more receive windows
    - 0 otherwise
14. **FOptsLen :** is the length of the FOpts field in bytes ă 0000 to 1111
15. **FCnt :** 2 type of frame counters
    - FCntUp: counter for uplink data frame, MAX-FCNT-GAP
    - FCntDown: counter for downlink data frame, MAX-FCNY-GAP
16. **FOpts :** is used to piggyback MAC commands on a data message
17. **FPort :** a multiplexing port field
    - 0 the payload contains only MAC commands
    - 1 to 223 Application Specific
    - 224 & 225 RFU
18. **FRMPayload :** (Frame Payload) Encrypted (AES, 128 key length) Data
19. **MIC :** is a cryptographic message integrity code
    - computed over the fields MHDR, FHDR, FPort and the encrypted FRMPayload.
20. **CRC :** (only in uplink),
    - CCITT $x^{16} + x^{12} + x^5 + 1$
    - IBM $x^{16} + x^{15} + x^5 + 1$

2 Divers

$$
\begin{aligned}
SF &= log_2 \frac{R_c}{SR} && (1.1) \\
R_{c[chips/s]} &= BW && (1.2) \\
R_{S[symbols/s]} &= \frac{R_c}{2^{SF}} = \frac{BW}{2^{SF}} && (1.3) \\
R_m = DR_{\mathbf{[bps]}} &= SF.RS = SF * \frac{BW_{[Hz]}}{2^{SF}} * CR && (1.4) \\
SF &= log_2 \frac{R_c}{SR} && (1.5) \\
\mathbf{T_s} &= \frac{2^{SF}}{BW_{[Hz]}} && (1.6) \\
SR_{\mathbf{[sps]}} &= \frac{BW}{2^{SF}} && (1.7) \\
DR_{\mathbf{[bps]}} &= SF * \frac{BW_{[Hz]}}{2^{SF}} * CR && (1.8) \\
BR_{\mathbf{[bps]}} &= SF * \frac{\frac{4}{4+CR}}{\frac{2SF}{BW}} && (1.9) \\
Sen_{\mathbf{[dBm]}} &= -174 + 10\log_{10} BW + NF + SNR && (1.10) \\
SNR_{\mathbf{[dB]}} &= 20.log(\frac{S}{N}) && (1.11) \\
SNR_{\mathbf{[dB]}} &= 20.log(\frac{S}{N}) && (1.12) \\
BER_{\mathbf{[bps]}} &= \frac{8}{15}.\frac{1}{16}.\sum k = 216{-}1^k(\frac{16}{k})e^{20.SINR(\frac{1}{k}-1)} && (1.13) \\
BER_{\mathbf{[bps]}} &= 10^{\alpha.e^{\ \beta SNR}} && (1.14) \\
PER_{\mathbf{[pps]}} &= 1 - (1 - BER)^{n_{bits}} && (1.15) \\
PRR &= (1 - BER)^L && (1.16) \\
RSSI &= Tx_{power}.\frac{Rayleigh_{power}}{PL} && (1.17) \\
& && (1.18)
\end{aligned}
$$

fakhfakh_deep_2017

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(p_i - r_i)^2 \tag{1.19}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - r_i)^2} \tag{1.20}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - r_i| \tag{1.21}$$

$$Recall = \frac{TP}{TP + FN} \tag{1.22}$$

$$Precision = \frac{TP}{TP + FP} \tag{1.23}$$

$$F1\_Score = \frac{2 \times \text{ Precision } \times \text{ Recall}}{\text{precision } + \text{recall}} \tag{1.24}$$

$$TPR = \frac{TP}{TP + FN} \tag{1.25}$$

$$FPR = \frac{FP}{FP + TN} \tag{1.26}$$

$$ROC = (TPR, FPR) \tag{1.27}$$

$$Novelty = \sum_{i \in L} \frac{\log_2 P_i}{n} \quad \text{where} \quad P_i = \frac{n - rank_i}{n - 1} \tag{1.28}$$

$$Serendipity = \frac{1}{n}\sum_{i \in n} \max\left(P_{\text{user}} - P_U, 0\right) \times rel_i \tag{1.29}$$

$$diversity = \frac{a}{c}\sum_{i=1}^{c}\frac{1}{n}\sum_{j=1}^{n} i_j \tag{1.30}$$

$$Coverage = 100 \times \frac{u}{U} \tag{1.31}$$

$$Stability = \frac{1}{P_2}\sum_{i \in P_2}|P_{2,i} - P_{1,i}| \tag{1.32}$$

$$DCG = rel_1 + \sum_{i=2}^{\text{pos}} \frac{rel_i}{\log_2 i} \tag{1.33}$$

$$IDCG = rel_1 + \sum_{i=2}^{|h|-1} \frac{rel_i}{\log_2 i} \tag{1.34}$$

$$NDCG = \frac{DCG}{IDCG} \tag{1.35}$$

$$\tag{1.36}$$