1 | Industrial Survey

”Given one hour to save the planet, I would spend 59 minutes understanding the problem and one minute resolving it.” - Albert Einstein

1 IoT applications requirements bregell_hardware_2015

1.1 Summary and discussion

Use cases	Resources	Mobility	Heterogeneity	Scalability	QoS	Data	standardization	Safety	Security
Logistics									
Health-care									
Smart environment									
Personal and social									
Futuristic									
Smart Mobility									
Smart semaphores control									
Smart Red Swarm									
Smart panels									
Smart bus scheduling									
Smart EV management									
Smart surface parking									
Smart signs									
Smart energy systems									
Smart lighting									
Smart water jet systems									
Smart gathering									
Smart building									
Smart tourism									
Smart QRinfo									
Smart monitoring									
Smart hawk-eye									
Health Monitoring									
Water Distribution									
Electricity Distribution									
Smart Buildings									
Intelligent Transportation									
Surveillance									
Environmental Monitoring									

Table 1.1. Main IoT challenges**kouicem__internet__2018** + [1] **hancke__role__2012 alba__intelligent__2016**

2 IoT Wireless Networks (Norms & Standards)

Several different wireless communication protocols, such as Wireless LAN (WLAN), BLE, 6LoWPAN, and ZigBee may be suitable for IoT applications. They all operate in the 2.4GHz frequency band and this, together with the limited output power in this band, means that they all have a similar range. The main differences are located in the MAC, PHY, and network layer. WLAN is much too power hungry as seen in table 2.6 and is only listed as a reference for the comparisons.

2.1 SigFox

2.2 IETF

6LoWPAN

is a relatively new protocol that is maintained by the Internet Engineering Task Force (IETF) [7, 6]. The purpose of the protocol is to enable IPv6 traffic over a IEEE 802.15.4 network with as low overhead as possible; this is achieved by compressing the IPv6 and UDP header. A full size IPv6 + UDP header is 40+8 bytes which is tild 38% of a IEEE 802.15.4 frame, but with the header compression this overhead can be reduced to 7 bytes, thus reducing the overhead to tild 5%, as seen in figures 2.3 and 2.4.

2.3 3GPP

NB-IoT

EC-GSM

e-MTC

2.4 IEEE

IEEE 802.11

IEEE 802.15.4

At present days, there are several technology standards. Each one is designed for a specific need in the market. For the Wireless Sensor Networks, the aim is to transmit little information, in a small range, with a small power consumption and low cost. The IEEE 802.15.4 standard offers physical and media access control layers for low-cost, low-speed, low-power Wireless Personal Area Networks (WPANs)

Physical Layer The standard operates in 3 different frequency bands: - 16 channels in the 2.4GHz ISM band - 10 channels in the 915MHz ISM band - 1 channel in the European 868MHz band

Definitions Coordinator: A device that provides synchronization services through the transmission of beacons. PAN Coordinator: The central coordinator of the PAN. This device identifies its own network as well as its configurations. There is only one PAN Coordinator for each network. Full Function Device (FFD): A device that implements the complete protocol set, PAN coordinator capable , talks to any other device. This type of device is suitable for any topology. Reduced Function Device (RFD): A device with a reduced implementation of the protocol, cannot become a PAN Coordinator. This device is limited to leafs in some topologies.

Topologies Star topology: All nodes communicate via the central PAN coordinator , the leafs may be any combination of FFD and RFD devices. The PAN coordinator usually uses main power.

Peer to peer topology: Nodes can communicate via the central PAN coordinator and via additional point-to-point links . All devices are FFD to be able to communicate with each other.

Combined Topology: Star topology combined with peer-to-peer topology. Leafs connect to a network via coordinators (FFDs) . One of the coordinators serves as the PAN coordinator .

IEEE 802.15.4

The IEEE 802.15.4 standard defines the PHY and MAC layers for wireless communication [6]. It is designed to use as little transmission time as possible but still have a decent payload, while consuming as little power as possible. Each frame starts with a preamble and a start frame delimiter; it then continues with the MAC frame length and the MAC frame itself as seen in figure 2.2. The overhead for each PHY packet is only 4+1+1 133 tild 4.5%; when using the maximum transmission speed of 250kbit/s, each frame can be sent 133byte in 250kbit/s = 4.265ms. Furthermore, it can also operate in the 868MHz and 915MHz bands, maintaining the 250kbit/s transmission rate by using Offset quadrature phase-shift keying (O-QPSK).

Several network layer protocols are implemented on top of IEEE 802.15.4. The two that will be examined are 6LoWPAN and ZigBEE.

ZigBee

is a communication standard initially developed for home automation networks; it has several different protocols designed for specific areas such as lighting, remote control, or health care [27, 6]. Each of these protocols uses their own addressing with different overhead; however, there is also the possibility of direct IPv6 addressing. Then, the overhead is the same as for uncompressed 6LoWPAN, as seen in figure 2.5.

A new standard called ZigBee 3.0 aims to bring all these standards together under one roof to simplify the integration into IoT. The release date of this standard is set to Q4 2015.

2.5 LoRaWAN

LoRa (Long Range) is a proprietary spread spectrum modulation technique by Semtech. It is a derivative of Chirp Spread Spectrum (CSS). The LoRa physical layer may be used with any MAC layer; however, LoRaWAN is the currently proposed MAC which operates a network in a simple star topology.

As LoRa is capable to transmit over very long distances it was decided that LoRaWAN only needs to support a star topology. Nodes transmit directly to a gateway which is powered and connected

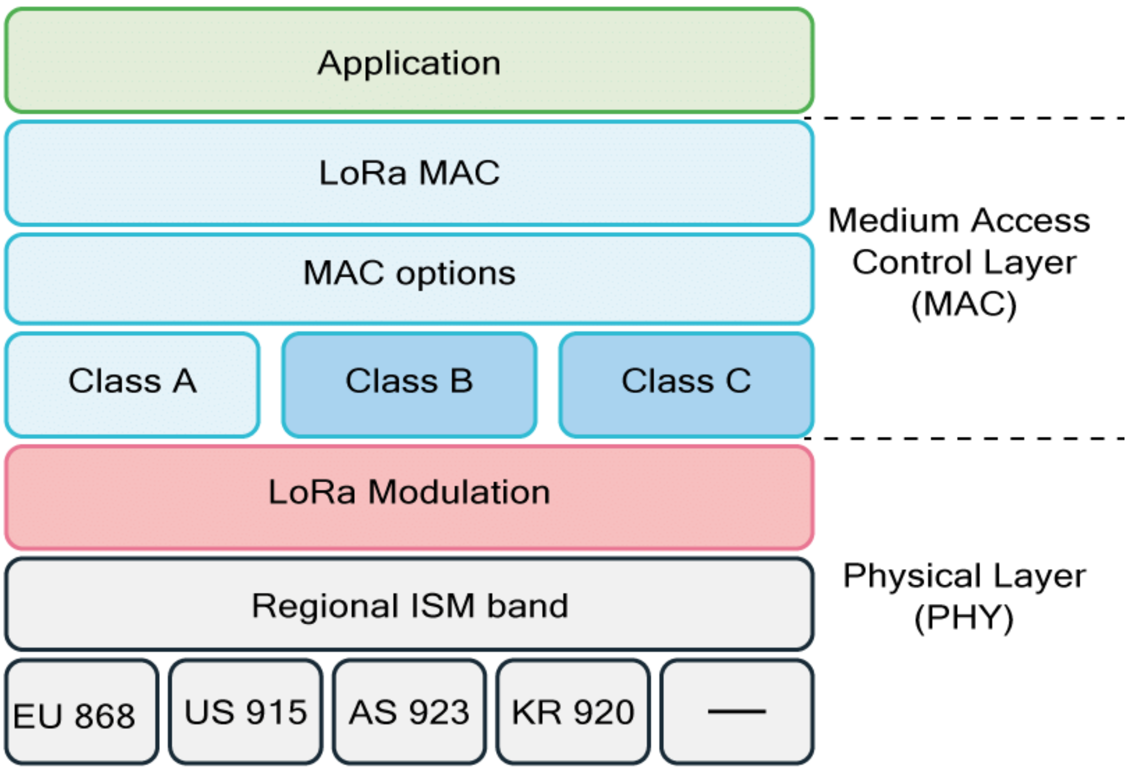

Figure 1. uhuhuh.

to a backbone infrastructure. Gateways are powerful devices with powerful radios capable to receive and decode multiple concurrent transmissions (up to 50). Three classes of node devices are defined: (1) Class A enddevices: The node transmits to the gateway when needed. After transmission the node opens a receive window to obtain queued messages from the gateway. (2) Class B enddevices with scheduled receive slots: The node behaves like a Class A node with additional receive windows at scheduled times. Gateway beacons are used for time synchronisation of end-devices. (3) Class C end-devices with maximal receive slots: these nodes are continuous listening which makes them unsuitable for battery powered operations.

ALIANCE

Class-A

Uplink LoRa Server supports Class-A devices. In Class-A a device is always in sleep mode, unless it has something to transmit. Only after an uplink transmission by the device, LoRa Server is able to schedule a downlink transmission. Received frames are de-duplicated (in case it has been received by multiple gateways), after which the mac-layer is handled by LoRa Server and the encrypted application-playload is forwarded to the application server.

Downlink LoRa Server persists a downlink device-queue for to which the application-server can enqueue downlink payloads. Once a receive window occurs, LoRa Server will transmit the first downlink payload to the device.

Confirmed data LoRa Server sends an acknowledgement to the application-server as soon one is received from the device. When the next uplink transmission does not contain an acknowledgement, a nACK is sent to the application-server.

Note: After a device (re)activation the device-queue is flushed.

Class-B LoRa Server supports Class-B devices. A Class-B device synchronizes its internal clock using Class-B beacons emitted by the gateway, this process is also called a “beacon lock”. Once in the state of a beacon lock, the device negotioates its ping-interval. LoRa Server is then able to schedule downlink transmissions on each occuring ping-interval.

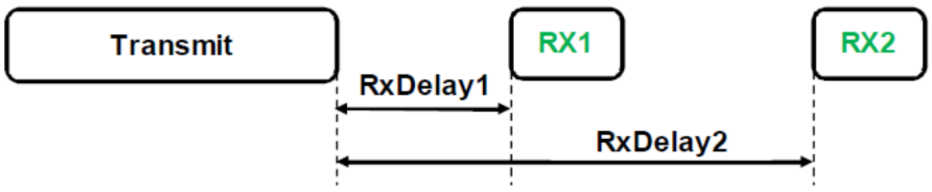

Figure 2. Class A.

Downlink LoRa Server persists all downlink payloads in its device-queue. When the device has acquired a beacon lock, it will schedule the payload for the next free ping-slot in the queue. When adding payloads to the queue when a beacon lock has not yet been acquired, LoRa Server will update all device-queue to be scheduled on the next free ping-slot once the device has acquired the beacon lock.

Confirmed data LoRa Server sends an acknowledgement to the application-server as soon one is received from the device. Until the frame has timed out, LoRa Server will wait with the transmission of the next downlink Class-B payload.

Note: The timeout of a confirmed Class-B downlink can be configured through the device-profile. This should be set to a value less than the interval between two ping-slots.

Requirements

Device The device must be able to operate in Class-B mode. This feature has been tested against the develop branch of the Semtech LoRaMac-node source.

Gateway The gateway must have a GNSS based time-source and must use at least the Semtech packet-forwarder version 4.0.1 or higher. It also requires LoRa Gateway Bridge 2.2.0 or higher.

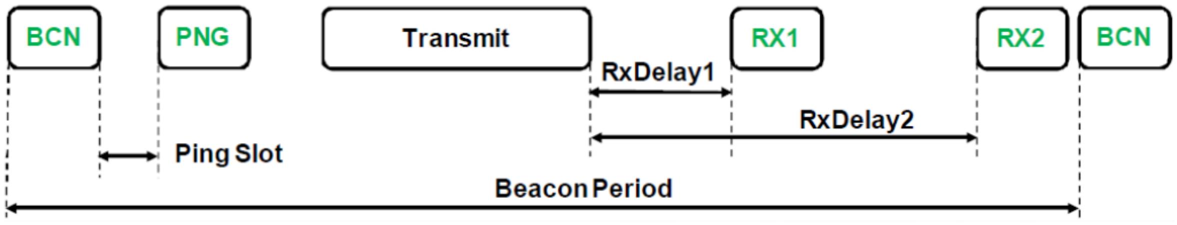

Figure 3. Class B.

Class-C

Downlink LoRa Server supports Class-C devices and uses the same Class-A downlink device-queue for Class-C downlink transmissions. The application-server can enqueue one or multiple downlink payloads and LoRa Server will transmit these (semi) immediately to the device, making sure no overlap exists in case of multiple Class-C transmissions.

Confirmed data LoRa Server sends an acknowledgement to the application-server as soon one is received from the device. Until the frame has timed out, LoRa Server will wait with the transmission of the next downlink Class-C payload.

Note: The timeout of a confirmed Class-C downlink can be configured through the device-profile.

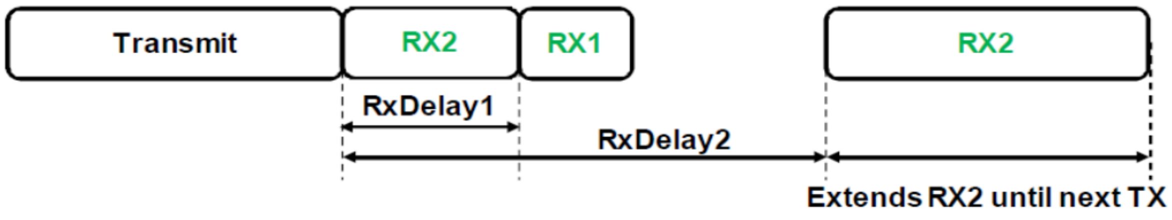

Figure 4. Class C.

SEMTECH

LoRa has four configurable parameters:

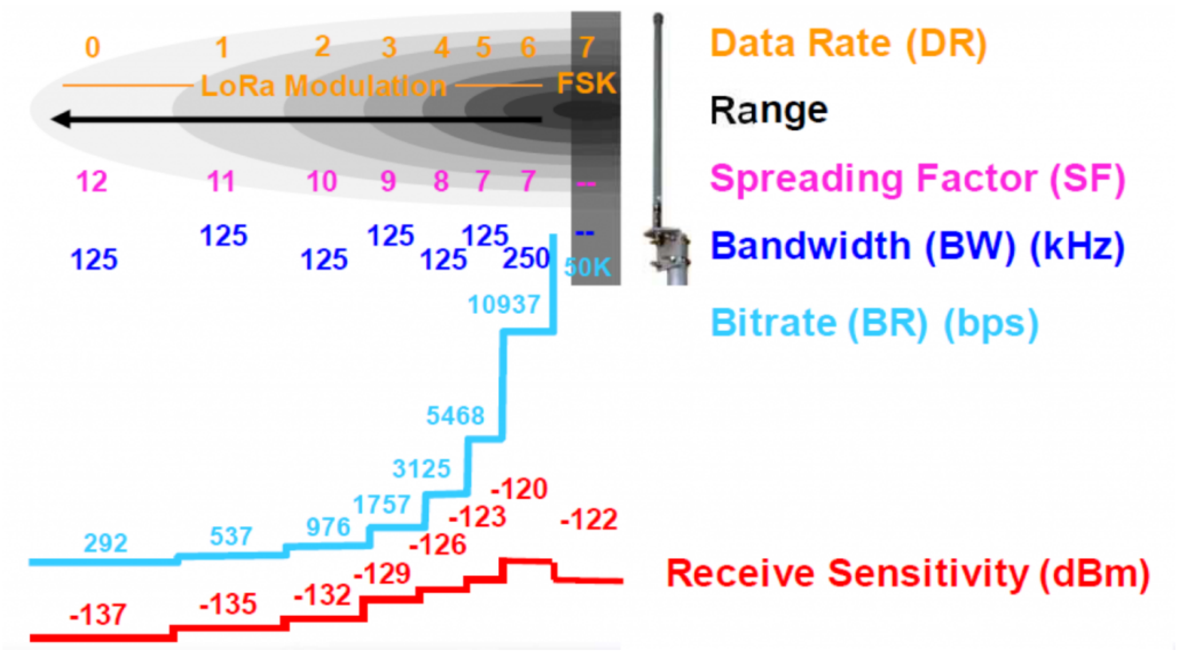

Figure 5. LoraWan Parameters.

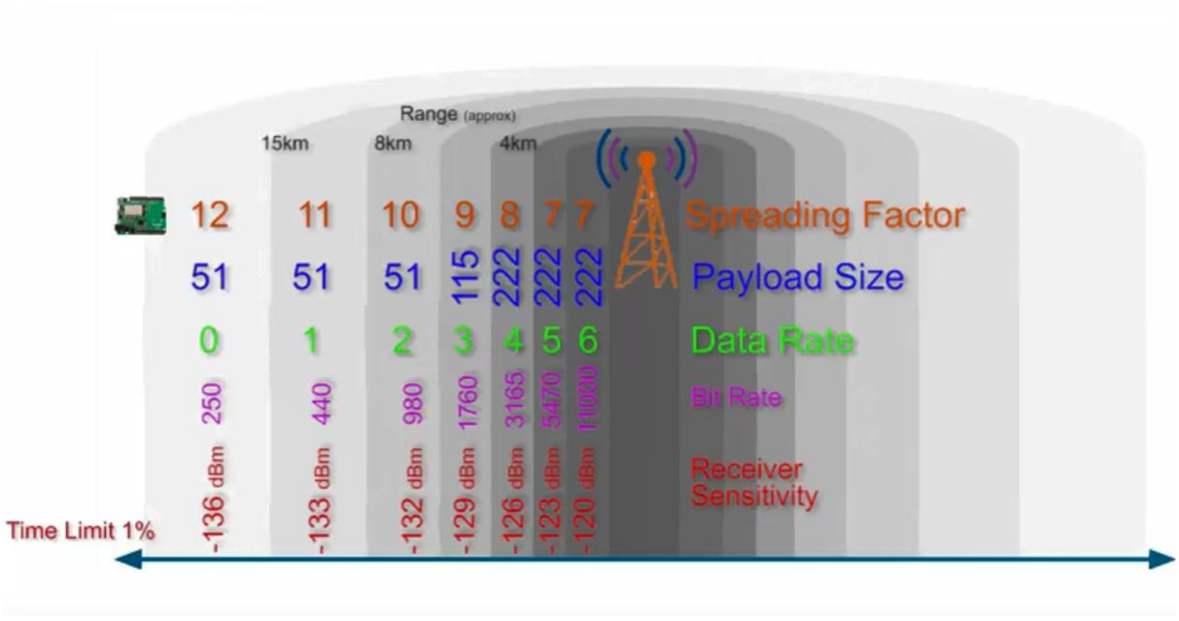

Figure 6. .

BW **Bandwidth:** Bandwidth (BW) is the range of frequencies in the transmission band. Higher BW gives a higher data rate (thus shorter time on air), but a lower sensitivity (due to integration of additional noise). A lower BW gives a higher sensitivity, but a lower data rate. Lower BW also requires more accurate crystals (less ppm). Data is send out at a chip rate equal to the bandwidth. So, a bandwidth of 125 kHz corresponds to a chip rate of 125 kcps. The SX1272 has three programmable bandwidth settings: 500 kHz, 250 kHz and 125 kHz. The Semtech SX1272 can be programmed in the range of 7.8 kHz to 500 kHz, though bandwidths lower than 62.5 kHz requires a temperature compensated crystal oscillator (TCXO).

CF **Carrier Frequency:** Carrier Frequency (CF) is the centre frequency used for the transmission band. For the SX1272 it is in the range of 860 MHz to 1020 MHz, programmable in steps of 61 Hz. The alternative radio chip Semtech SX1276 allows adjustment from 137 MHz to 1020 MHz.

CR **Coding Rate:** Coding Rate (CR) is the FEC rate used by the LoRa modem and offers protection against bursts of interference. A higher CR offers more protection, but increases time on air. Radios with different CR (and same CF/SF/BW), can still communicate with each other.

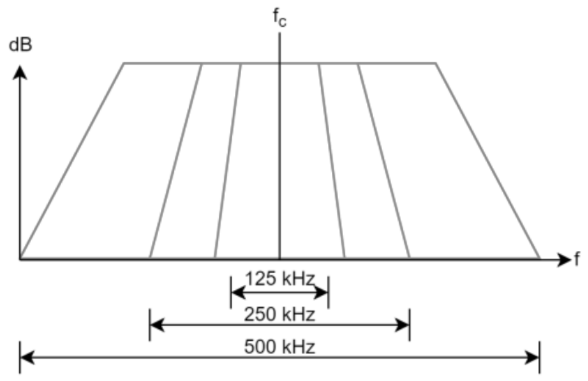

Figure 7. .

CR of the payload is stored in the header of the packet, which is always encoded at 4/8.

SF **Spreading Factor:** SF is the ratio between the symbol rate and chip rate. A higher spreading factor increases the Signal to Noise Ratio (SNR), and thus sensitivity and range, but also increases the air time of the packet. The number of chips per symbol is calculated as 2 sf . For example, with an SF of 12 (SF12) 4096 chips/symbol are used. Each increase in SF halves the transmission rate and, hence, doubles transmission duration and ultimately energy consumption. Spreading factor can be selected from 6 to 12. SF6, with the highest rate transmission, is a special case and requires special operations. For example, implicit headers are required. Radio communications with different SF are orthogonal to each other and network separation using different SF is possible.

Tx **Transmition power:**

Payload **Payload legth:**

SF		07	08	09	10	11	12	07	08	09	10	11	12	07	08	09	10	11	12
	BW	125						250						500					
07	125	x								x								x	
08			x								x								x
09				x								x							
10					x								x						
11						x													
12							x												
07	250							x								x			
08									x								x		
09		x								x								x	
10			x								x								x
11				x								x							
12					x								x						
07	500													x					
08															x				
09								x								x			
10									x								x		
11		x								x								x	
12			x								x								x

Table 1.2. uyuyuy

	SF_7	SF_B	SF_9	SF_{10}	SF_{11}	SF_{12}
SF_7	- 6	16	18	19	19	20
SF_8	24	- 6	20	22	22	22
SF_9	27	27	- 6	23	25	25
SF_{10}	30	30	30	- 6	26	28
SF_{11}	33	33	33	33	- 6	29
SF_{12}	36	36	36	36	36	- 6

Module	SX1261/62/68	SX1272/73	SX1276/77/78/79
Modem	LoRa & FSK	LoRa	LoRa
Link budget	170dB	157dB	168dB
Power amplifier	/61: +15dBm 62/68:+22dBm	+14 dBm	+14dBm
Rx current	4.6 mA	10 mA	10 mA
Bit rate	62.5kbps-LoRa 300kbps-FSK	300 kbps	300 kbps
Sensitivity	-148 dBm	-137 dBm	-148 dBm
Blocking immunity	88 dB	89 dB	Excellent
Frequency	150-960 MHz	860-1000 MHz	137-1020 MHz
RSSI		127 dB	127 dB

Table 1.3. **gaddam__comparative__2018**

2.6 Divers

IPLC

BACnet

Z-WAze

Bluetooth LE

BLE is developed to be backwards compatible with Bluetooth, but with lower data rate and power consumption [28]. Featuring a data rate of 1Mbit/s with a peak current consumption less than 15mA, it is a very efficient protocol for small amounts of data. Each frame can be transmitted 47bytes in 1Mbit/s = 376Mus; thanks to the short transmission time, the transceivers consumes less power as the transceiver can be in receive mode or completely off most of the time. BLE uses its own addressing methods and as the MAC frame size (figure 2.6) is only 39bytes, thus IPv6 addressing is not possible.

Starting from Bluetooth version 4.2, there is support for IPv6 addressing with the Internet Protocol Support Profile; the new version allows the BLE frame to be variable between 2 257 bytes. The network set-up is controlled by the standard Bluetooth methods, whereas IPv6 addressing is handled by 6LoWPAN as specified in IPv6 over Bluetooth Low Energy [29].

2.7 Summary and discussion

3 IoT Protocols

Application protocol	DDS	CoAP	AMQP	MQTT	MQTT-SN	XMPP	HTTP
Service discovery		mDNS				DNS-SD	
Transport				UDP/TCP			
Network		IPv6 RPL			IPv4/IPv6		
		6LowPan		RFC 2464			RFC 5072
MAC		IEEE 802.15.4	IEEE 802.11 (Wi-Fi)		IEEE 802.3 (Ethernet)		2G, 3G, LTE
		2.4GHz, 915, 868MHz	2.4, 5GHz				
		DSS, FSK, OFDM	CSMA/CA		CUTP, FO		

Table 1.4. Standardization efforts that support the IoT

3.1 Application

LwM2M

CBOR

DTLS

OSCOAP

CoAP

- **Ver:** is the version of CoAP
- **T:** is the type of Transaction
- **TKL:** Token length
- **Code:** represents the request method (1-10) or response code (40-255).
    - Ex: the code for GET, POST, PUT, and DELETE is 1, 2, 3, and 4, respectively.
- **Message ID:** is a unique identifier for matching the response.
- **Token:** Optional response matching token.

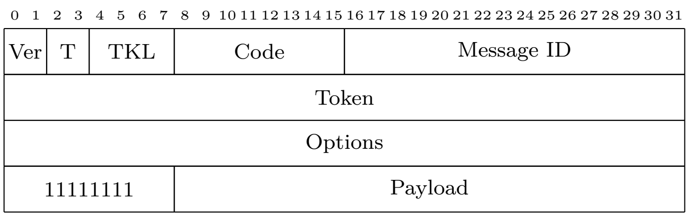

Figure 8. CoAP Header.

MQTT

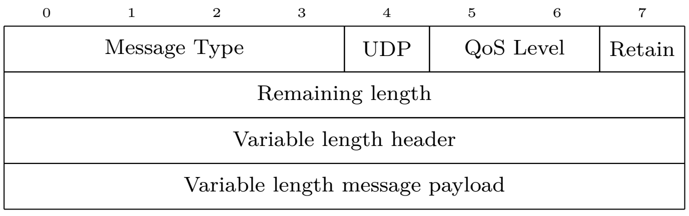

Figure 9. CoAP Header.

- **Message type:** CONNECT (1), CONNACK (2), PUBLISH (3), SUBSCRIBE (8) and so on
- **DUP flag:** indicates that the massage is duplicated
- **QoS Level:** identify the three levels of QoS for delivery assurance of Publish messages
- **Retain field:** retain the last received Publish message and submit it to new subscribers as a first message

XMPP

- Extensible Messaging and Presence Protocol
- Developed by the Jabber open source community
- An IETF instant messaging standard used for:
  - multi-party chatting, voice and telepresence
- Connects a client to a server using a XML stanzas
- An XML stanza is divided into 3 components:
  - message: fills the subject and body fields
  - presence: notifies customers of status updates
  - iq (info/query): pairs message senders and receivers
- Message stanzas identify:
  - the source (from) and destination (to) addresses
  - types, and IDs of XMPP entities

AMQP

- **Size:** the frame size.
- **DOFF:** the position of the body inside the frame.
- **Type:** the format and purpose of the frame.
  - Ex: 0x00 show that the frame is an AMQP frame
  - Ex: 0x01 represents a SASL frame.

DDS

- Data Distribution Service
- Developed by Object Management Group (OMG)
- Supports 23 QoS policies:
  - like security, urgency, priority, durability, reliability, etc
- Relies on a broker-less architecture
  - uses multicasting to bring excellent Quality of Service
  - real-time constraints
- DDS architecture defines two layers:
  - **DLRL:** Data-Local Reconstruction Layer
    - serves as the interface to the DCPS functionalities

  - **DCPS:** Data-Centric Publish/Subscribe
    - delivering the information to the subscribers
- 5 entities are involved with the data flow in the DCPS layer:
  - Publisher:disseminates data
  - DataWriter: used by app to interact with the publisher
  - Subscriber: receives published data and delivers them to app
  - DataReader: employed by Subscriber to access received data
  - Topic: relate DataWriters to DataReaders
- No need for manual reconfiguration or extra administration
- It is able to run without infrastructure
- It is able to continue working if failure happens.
- It inquires names by sending an IP multicast message to all the nodes in the local domain
  - Clients asks devices that have the given name to reply back
  - the target machine receives its name and multicasts its IP @
  - Devices update their cache with the given name and IP @

mDNS

- Requires zero configuration aids to connect machine
- It uses mDNS to send DNS packets to specific multicast addresses through UDP
- There are two main steps to process Service Discovery:
  - finding host names of required services such as printers
  - pairing IP addresses with their host names using mDNS
- Advantages
  - IoT needs an architecture without dependency on a configuration mechanism
  - smart devices can join the platform or leave it without affecting the behavior of the whole system
- Drawbacks
  - Need for caching DNS entries

COAP (COnstrained Application Protocol)

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things. More detailed information about the protocol is given in the Contiki OS CoAP section.

Overview Like HTTP, CoAP is a document transfer protocol. Unlike HTTP, CoAP is designed for the needs of constrained devices. The packets are much smaller than HTTP TCP flows. Packets are simple to generate and can be parsed in place without consuming extra RAM in constrained devices. CoAP runs over UDP, not TCP. Clients and servers communicate through connectionless datagrams. Retries and reordering are implemented in the application stack. It follows a client/server model. Clients make requests to servers, servers send back responses. Clients may GET, PUT, POST and DELETE resources. CoAP implements the REST model from HTTP, with the primitives GET, POST, PUT and DELETE.

Coap Methods CoAP extends the HTTP request model with the ability to observe a resource. When the observe flag is set on a CoAP GET request, the server may continue to reply after the initial document has been transferred. This allows servers to stream state changes to clients as they occur. Either end may cancel the observation. CoAP defines a standard mechanism for resource discovery. Servers provide a list of their resources (along with metadata about them) at /.well-known/core. These links are in the application/link-format media type and allow a client to discover what resources are provided and what media types they are.

Coap Transactions

Coap Messages The CoAP message structure is designed to be simpler than HTTP, for reduced transmission data. Each field responds to a specific purpose.

- Constrained Application Protocol
- The IETF Constrained RESTful Environments
- CoAP is bound to UDP
- CoAP can be divided into two sub-layers
  - messaging sub-layer
  - request/response sub-layer
    a) Confirmable.
    b) Non-confirmable.

   c) Piggybacked responses.
   d) Separate response
- CoAP, as in HTTP, uses methods such as:
  - GET, PUT, POST and DELETE to
  - Achieve, Create, Retrieve, Update and Delete
  - Ex: the GET method can be used by a server to inquire the client's temperature

MQTT

- Message Queue Telemetry Transport
- Andy Stanford-Clark of IBM and Arlen Nipper of Arcom
  - Standardized in 2013 at OASIS
- MQTT uses the publish/subscribe pattern to provide transition flexibility and simplicity of implementation
- MQTT is built on top of the TCP protocol
- MQTT delivers messages through three levels of QoS
- Specifications
  - MQTT v3.1 and MQTT-SN (MQTT-S or V1.2)
  - MQTT v3.1 adds broker support for indexing topic names
- The publisher acts as a generator of interesting data.

XMPP

- Extensible Messaging and Presence Protocol
- Developed by the Jabber open source community
- An IETF instant messaging standard used for:
  - multi-party chatting, voice and telepresence
- Connects a client to a server using a XML stanzas
- An XML stanza is divided into 3 components:
  - message: fills the subject and body fields
  - presence: notifies customers of status updates
  - iq (info/query): pairs message senders and receivers
- Message stanzas identify:
  - the source (from) and destination (to) addresses
  - types, and IDs of XMPP entities

AMQP

- Advanced Message Queuing Protocol
- Communications are handled by two main components
  - exchanges: route the messages to appropriate queues.
  - message queues: Messages can be stored in message queues and then be sent to receivers
- It also supports the publish/subscribe communications.
- It defines a layer of messaging on top of its transport layer.
- AMQP defines two types of messages
  - bare massages: supplied by the sender
  - annotated messages: seen at the receiver
- The header in this format conveys the delivery parameters:
  - durability, priority, time to live, first acquirer & delivery count.
- AMQP frame format

Size	the frame size.
DOFF	the position of the body inside the frame.
Type	the format and purpose of the frame. ✳ Ex: 0x00 show that the frame is an AMQP frame ✳ Ex: 0x01 represents a SASL frame.

DDS

- Data Distribution Service
- Developed by Object Management Group (OMG)
- Supports 23 QoS policies:
  - like security, urgency, priority, durability, reliability, etc
- Relies on a broker-less architecture
  - uses multicasting to bring excellent Quality of Service

Application protocol	Rest-Full	Trans-port	Publish/Sub-scribe	Request/Re-sponse	Security	QoS	Header size (Byte)
COAP	✓	UDP	✓	✓	DTLS	✓	4
MQTT	✗	TCP	✓	✗	SSL	✓	2
MQTT-SN	✗	TCP	✓	✗	SSL	✓	2
XMPP	✗	TCP	✓	✓	SSL	✗	-
AMQP	✗	TCP	✓	✗	SSL	✓	8
DDS	✗	UDP TCP	✓	✗	SSL DTLS	✓	-
HTTP	✓	TCP	✗	✓	SSL	✗	-

Table 1.5. Application protocols comparison

    - real-time constraints
- DDS architecture defines two layers:

DLRL Data-Local Reconstruction Layer

        - serves as the interface to the DCPS functionalities

DCPS Data-Centric Publish/Subscribe

        - delivering the information to the subscribers

- 5 entities are involved with the data flow in the DCPS layer:
    - Publisher:disseminates data
    - DataWriter: used by app to interact with the publisher
    - Subscriber: receives published data and delivers them to app
    - DataReader: employed by Subscriber to access received data
    - Topic: relate DataWriters to DataReaders
- No need for manual reconfiguration or extra administration
- It is able to run without infrastructure
- It is able to continue working if failure happens.
- It inquires names by sending an IP multicast message to all the nodes in the local domain
    - Clients asks devices that have the given name to reply back
    - the target machine receives its name and multicasts its IP @
    - Devices update their cache with the given name and IP @

mDNS

- Requires zero configuration aids to connect machine
- It uses mDNS to send DNS packets to specific multicast addresses through UDP
- There are two main steps to process Service Discovery:
    - finding host names of required services such as printers
    - pairing IP addresses with their host names using mDNS
- Advantages
    - IoT needs an architecture without dependency on a configuration mechanism
    - smart devices can join the platform or leave it without affecting the behavior of the whole system
- Drawbacks
    - Need for caching DNS entries

3.2 Network

6TiSCH

OLSRv2

AODVv2

LoRaWAN

ROHC

IPHC

SCHC

NHC

ROLL

RPL

RPL is a Distance Vector IPv6 routing protocol for LLNs that specifies how to build a Destination Oriented Directed Acyclic Graph (DODAG) using an objective function and a set of metrics/constraints. The objective function operates on a combination of metrics and constraints to compute the 'best' path.

An RPL Instance consists of multiple Destination Oriented Directed Acyclic Graphs (DODAGs). Traffic moves either up towards the DODAG root or down towards the DODAG leafs. The graph building process starts at the root or LBR (LowPAN Border Router). There could be multiple roots configured in the system. The RPL routing protocol specifies a set of ICMPv6 control messages to exchange graph related information. These messages are called DIS (DODAG Information Solicitation), DIO (DODAG Information Object) and DAO (DODAG Destination Advertisement Object). The root starts advertising the information about the graph using the DIO message. The nodes in the listening vicinity (neighbouring nodes) of the root will receive and process DIO messages potentially from multiple nodes and makes a decision based on certain rules (according to the objective function, DAG characteristics, advertised path cost and potentially local policy) whether to join the graph or not. Once the node has joined a graph it has a route toward the graph (DODAG) root. The graph root is termed as the 'parent' of the node. The node computes the 'rank' of itself within the graph, which indicates the "coordinates" of the node in the graph hierarchy. If configured to act as a router, it starts advertising the graph information with the new information to its neighbouring peers. If the node is a "leaf node", it simply joins the graph and does not send any DIO message. The neighbouring peers will repeat this process and do parent selection, route addition and graph information advertisement using DIO messages. This rippling effect builds the graph edges out from the root to the leaf nodes where the process terminates. In this formation each node of the graph has a routing entry towards its parent (or multiple parents depending on the objective function) in a hop-by-hop fashion and the leaf nodes can send a data packet all the way to root of the graph by just forwarding the packet to its immediate parent. This model represents a MP2P (Multipoint-to-point) forwarding model where each node of the graph has reach-ability toward the graph root. This is also referred to as UPWARD routing. Each node in the graph has a 'rank' that is relative and represents an increasing coordinate of the relative position of the node with respect to the root in graph topology. The notion of "rank" is used by RPL for various purposes including loop avoidance. The MP2P flow of traffic is called the 'up' direction in the DODAG.

The DIS message is used by the nodes to proactively solicit graph information (via DIO) from the neighbouring nodes should it become active in a stable graph environment using the 'poll' or 'pull' model of retrieving graph information or in other conditions. Similar to MP2P or 'up' direction of traffic, which flows from the leaf towards the root there is a need for traffic to flow in the opposite or 'down' direction. This traffic may originate from outside the LLN network, at the root or at any intermediate nodes and destined to a (leaf) node. This requires a routing state to be built at every node and a mechanism to populate these routes. This is accomplished by the DAO (Destination Advertisement Object) message. DAO messages are used to advertise prefix reachability towards the leaf nodes in support of the 'down' traffic. These messages carry prefix information, valid lifetime and other information about the distance of the prefix. As each node joins the graph it will send DAO message to its parent set. Alternately, a node or root can poll the sub-dag for DAO message through an indication in the DIO message. As each node receives the DAO message, it processes the prefix information and adds a routing entry in the routing table. It optionally aggregates the prefix information received from various nodes in the subdag and sends a DAO message to its parent set. This process continues until the prefix information reaches the root and a complete path to the prefix is setup. Note that this mode is called the "storing" mode of operation where intermediate nodes have available memory to store routing tables. RPL also supports another mode called "non-storing" mode where intermediate node do not store any routes.

6LowPAN

6LoWPAN is a networking technology or adaptation layer that allows IPv6 packets to be carried efficiently within a small link layer frame, over IEEE 802.15.4 based networks. As the full name implies, "IPv6 over Low-Power Wireless Personal Area Networks", it is a protocol for connecting wireless low power networks using IPv6.

As the full name implies, "IPv6 over Low-Power Wireless Personal Area Networks", it is a protocol for connecting wireless low power networks using IPv6.

Characteristics

- Compression of IPv6 and UDP/ICMP headers
- Fragmentation / reassembly of IPv6 packets
- Mesh addressing
- Stateless auto configuration
-

Encapsulation Header format All LowPAN encapsulated datagrams are prefixed by an encapsulation header stack. Each header in the stack starts with a header type field followed by zero or more header fields.

Fragment Header The fragment header is used when the payload is too large to fit in a single IEEE 802.15.4 frame. The Fragment header is analogous to the IEEE 1394 Fragment header and includes three fields: Datagram Size, Datagram Tag, and Datagram Offset. Datagram Size identifies the total size of the unfragmented payload and is included with every fragment to simplify buffer allocation at the receiver when fragments arrive out-oforder. Datagram Tag identifies the set of fragments that correspond to a given payload and is used to match up fragments of the same payload. Datagram Offset identifies the fragment's offset within the unfragmented payload and is in units of 8-byte chunks.

Mesh addressing header The Mesh Addressing header is used to forward 6LoWPAN payloads over multiple radio hops and support layer-two forwarding. The mesh addressing header includes three fields: Hop Limit, Source Address, and Destination Address. The Hop Limit field is analogous to the IPv6 Hop Limit and limits the number of hops for forwarding. The Hop Limit field is decremented by each forwarding node, and if decremented to zero the frame is dropped. The source and destination addresses indicate the end-points of an IP hop. Both addresses are IEEE 802.15.4 link addresses and may carry either a short or extended address.

Header compression (RFC4944) RFC 4944 defines HC1, a stateless compression scheme optimized for link-local IPv6 communication. HC1 is identified by an encoding byte following the Compressed IPv6 dispatch header, and it operates on fields in the upper-layer headers. 6LoWPAN elides some fields by assuming commonly used values. For example, it compresses the 64-bit network prefix for both source and destination addresses to a single bit each when they carry the well-known link-local prefix. 6LoWPAN compresses the Next Header field to two bits whenever the packet uses UDP, TCP, or ICMPv6. Furthermore, 6LoWPAN compresses Traffic Class and Flow Label to a single bit when their values are both zero. Each compressed form has reserved values that indicate that the fields are carried inline for use when they don't match the elided case. 6LoWPAN elides other fields by exploiting cross-layer redundancy. It can derive Payload Length − which is always elided − from the 802.15.4 frame or 6LoWPAN fragmentation header. The 64-bit interface identifier (IID) for both source and destination addresses are elided if the destination can derive them from the corresponding link-layer address in the 802.15.4 or mesh addressing header. Finally, 6LoWPAN always elides Version by communicating via IPv6.

The HC1 encoding is shown in Figure 11. The first byte is the dispatch byte and indicates the use of HC1. Following the dispatch byte are 8 bits that identify how the IPv6 fields are compressed. For each address, one bit is used to indicate if the IPv6 prefix is linklocal and elided and one bit is used to indicate if the IID can be derived from the IEEE 802.15.4 link address. The TF bit indicates whether Traffic Class and Flow Label are both zero and elided. The two Next Header bits indicate if the IPv6 Next Header value is 7UDP, TCP, or ICMP and compressed or carried inline. The HC2 bit indicates if the next header is compressed using HC2. Fully compressed, the HC1 encoding reduces the IPv6 header to three bytes, including the dispatch header. Hops Left is the only field always carried inline.

RFC 4944 uses stateless compression techniques to reduce the overhead of UDP headers. When the HC2 bit is set in the HC1 encoding, an additional 8-bits is included immediately following the HC1 encoding bits that specify how the UDP header is compressed. To effectively compress UDP ports, 6LoWPAN introduces a range of wellknown ports (61616 – 61631). When ports fall in the well-known range, the upper 12 bits may be elided. If both ports fall within range, both Source and Destination ports are compressed down to a single byte. HC2 also allows elision of the UDP Length, as it can be derived from the IPv6 Payload Length field.

The best-case compression efficiency occurs with link-local unicast communication, where HC1 and HC2 can compress a UDP/IPv6 header down to 7 bytes. The Version, Traffic Class, Flow Label, Payload Length, Next Header, and linklocal prefixes for the IPv6 Source and Destination addresses are all elided. The suffix for both IPv6 source and destination addresses are derived from the IEEE 802.15.4 header.

However, RFC 4944 does not efficiently compress headers when communicating outside of link-local scope or when using multicast. Any prefix other than the linklocal prefix must be carried inline. Any suffix must be at least 64 bits when carried inline even if derived from a short 802.15.4 address. As shown in Figure 8, HC1/HC2 can compress a link-local multicast UDP/IPv6 header down to 23 bytes in the best case. When communicating with nodes outside the LoWPAN, the IPv6 Source Address prefix and full IPv6 Destination Address must be carried inline.

Header compression Improved (draft-hui-6lowpan-hc-01) To provide better compression over a broader range of scenarios, the 6LoWPAN working group is standardizing an improved header compression encoding format, called HC. The format defines a new encoding for compressing IPv6 header, called IPHC. The new format allows Traffic Class and Flow Label to be individually compressed, Hop Limit compression when common values (E.g., 1 or 255) are used, makes use of shared-context to elide the prefix from IPv6 addresses, and supports multicast addresses most often used for IPv6 ND and SLAAC. Contexts act as shared state for all nodes within the LoWPAN. A single context holds a single prefix. IPHC identifies the context using a 4-bit index, allowing IPHC to support up to 16 contexts simultaneously within the LoWPAN. When an IPv6 address matches a context's stored prefix, IPHC compresses the prefix to the context's 4-bit identifier. Note that contexts are not limited to prefixes assigned to the LoWPAN but can contain any arbitrary prefix. As a result, share contexts can be configured such that LoWPAN nodes can compress the prefix in both Source and Destination addresses even when communicating with nodes outside the LoWPAN.

The improved header compression encoding is shown in Figure 8. The first three bits (011) form the header type and indicate the use of IPHC. The TF bits indicate whether the Traffic Class and/or Flow Label fields are compressed. The HLIM bits indicate whether the Hop Limit takes the value 1 or 255 and compressed, or carried inline.

Bits 8-15 of the IPHC encoding indicate the compression methods used for the IPv6 Source and Destination Addresses. When the Context Identifier (CID) bit is zero, the default context may be used to compress Source and/or Destination Addresses. This mode is typically when both Source and Destination Addresses are assigned to nodes in the same LoWPAN. When the CID bit is one, two additional 4-bit fields follow the IPHC encoding to indicate which one of 16 contexts is in use for the source and destination addresses. The Source Address Compression (SAC) indicates whether stateless compression is used (typically for link-local communication) or stateful context-based compression is used (typically for global communication). The Source Address Mode (SAM) indicates whether the full Source Address is carried inline, upper 16 or 64-bits are elided, or the full Source Address is elided. When SAC is set and the Source Addresses' prefix is elided, the identified context is used to restore those bits. The Multicast (M) field indicates whether the Destination Address is a unicast or multicast address. When the Destination Address is a unicast address, the DAC and DAM bits are analogous to the SAC and SAM bits. When the Destination Address is a multicast address, the DAM bits indicate different forms of multicast compression. HC also defines a new framework for compressing arbitrary next headers, called NHC. HC2 in RFC 4944 is only capable of compressing UDP, TCP, and ICMPv6 headers, the latter two are not yet defined. Instead, the NHC header defines a new variable length Next Header identifier, allowing for future definition of arbitrary next header compression encodings. HC initially defines a compression encoding for UDP headers, similar to that defined in RFC 4944. Like RFC 4944, HC utilizes the same well-known port range (61616-61631) to effectively compress UDP ports down to 4-bits each in the best case. However, HC no longer provides an option to carry the Payload Length in line, as it can always be derived from the IPv6 header. Finally, HC allows elision of the UDP Checksum whenever an 10upper layer message integrity check covers the same information and has at least the same strength. Such a scenario is typical when transportor application-layer security is used. As a result, the UDP header can be compressed down to two bytes in the best case.

Routing protocol	Control Cost	Link Cost	Node Cost
OSPF/IS-IS	✗	✓	✗
OLSRv2	?	✓	✓
RIP	✓	?	✗
DSR	✓	✗	✗
RPL	✓	✓	✓

Table 1.6. Routing protocols comparison **__rpl2__**

- Routing over low-power and lossy links (ROLL)
- Support minimal routing requirements.
  - like multipoint-to-point, point-to-multipoint and point-to-point.
- A Destination Oriented Directed Acyclic Graph (DODAG)
  - Directed acyclic graph with a single root.
  - Each node is aware of ts parents
  - but not about related children
- RPL uses four types of control messages
  - DODAG Information Object (DIO)
  - Destination Advertisement Object (DAO)

- DODAG Information Solicitation (DIS)
- DAO Acknowledgment (DAO-ACk)

- Standard topologies to form IEEE 802.15.4e networks are

Star contains at least one FFD and some RFDs

Mesh contains a PAN coordinator and other nodes communicate with each other

Cluster consists of a PAN coordinator, a cluster head and normal nodes.

- The IEEE 802.15.4e standard supports 2 types of network nodes

FFD Full function device: serve as a coordinator
- ✳ It is responsible for creation, control and maintenance of the net
- ✳ It store a routing table in their memory and implement a full MAC

RFD Reduced function devices: simple nodes with restricted resources
- ✳ They can only communicate with a coordinator
- ✳ They are limited to a star topology

Routing protocol	**Control Cost**	**Link Cost**	**Node Cost**
OSPF/IS-IS	✗	✓	✗
OLSRv2	?	✓	✓
RIP	✓	?	✗
DSR	✓	✗	✗
RPL	✓	✓	✓

Table 1.7. Routing protocols comparison **__rpl2__**

3.3 MAC

Channel based	FDMA	OFDMA WDMA SC-FDMA		
	TDMA	MF-TDMA STDMA		
	CDMA	W-CDMA TD-CDMA TD-SCDMA DS-CDMA FH-CDMA MC-CDMA		
	SDMA	HC-SDMA		
Packet-based	Collision recovery	ALOHA Slotted ALOHA R-ALOHA AX.25 CSMA/CD		
	Collision avoidance	MACA MACAW CSMA CSMA/CA DCF PCF HCF CSMA/CARP		
	Collision-free	Token ring Token bus MS-ALOHA		
Duplexing methods	Delay and disruption tolerant	MANET VANET DTN Dynamic Source Routing		

Table 1.8

Sharing the channel

TDMA, FDMA, CDMA, TSMA

Transmitting information

TFDM, TDSSS, TFHSS

3.4 Radio

Digital modulation

ASK, APSK, CPM, FSK, MFSK, MSK, OOK, PPM, PSK, QAM, SC-FDE, TCM WDM

Hierarchical modulation

QAM, WDM

Spread spectrum

SS, DSSS, FHSS, THSS

Radio performance

Power Level (dB) The dB measures the power of a signal as a function of its ratio to another standardized value. The abbreviation dB is often combined with other abbreviations to represent the values that are compared. Here are two examples:

- dBm—The dB value is compared to 1 mW.
- dBw—The dB value is compared to 1 W.

$$Power(indB) = 10 * log(10)(Signal/Reference) \tag{1}$$

Where:

- log(10) is logarithm base 10.
- Signal is the power of the signal.
- Reference is the reference power.

For example, if you want to calculate the power in dB of 50 mW:
Power (in dB) = 10 * log(10) (50/1) = 10 * 1.7 = 17 dBm

Receive Signal Strength Indicator RSSI Receiver sensitivity is defined as the minimum signal power level with an acceptable Bit Error Rate (in dBm or mW) that is necessary for the receiver to accurately decode a given signal. This is usually expressed in a negative number depending on the data rate. For example an Access Point may require an RSSI of at least negative -91 dBm at 1 MB and an even higher strength RF power -79 dBm to decode 54 MB.

Signal to Noise Ratio SNR Noise is any signal that interferes with your signal. Noise can be due to other wireless devices such as cordless phones, microwave devices, etc. This value is measured in decibels from 0 (zero) to -120 (minus 120). Noise level is the amount of interference in your wireless signal, so lower is generally good for WLAN deployments. Typical environments range between -90dBm and -98dBm with little ambient noise. This value may be even higher if there is a lot of RF interference coming in from other non-802.11 devices on the same spectrum Signal to Noise Ratio or SNR is defined as the ratio of the transmitted power from the AP to the ambient (noise floor) energy present. To calculate the SNR value, we add the Signal Value to the Noise Value to get the SNR ratio. A positive value of the SNR ratio is always better. For example, say your Signal value is -55dBm and your Noise value is -95dBm. The difference of Signal (-55dBm) + Noise (-95dBm) = 40db—This means you have an SNR of 40. Note that in the above equation you are not merely adding two numbers, but are interested in the "difference" between the Signal and Noise values, which is usually a positive number. The lower the number, the lower the difference between noise and transmitted power, which in turn means lower quality of signal. The higher the difference between Signal and Noise means that the transmitted signal is of much higher power than the noise floor, thereby making it easier for a WLAN client to decode the signal.

Signal Attenuation Signal attenuation or signal loss occurs even as the signal passes through air. The loss of signal strength is more pronounced as the signal passes through different objects. A transmit power of 20 mW is equivalent to 13 dBm. Therefore if the transmitted power at the entry point of a plasterboard wall is at 13 dBm, the signal strength will be reduced to 10 dBm when exiting that wall. Some common examples are shown in Table 10-5.

3.5 Summary and discussion

4 IoT end devices

4.1 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

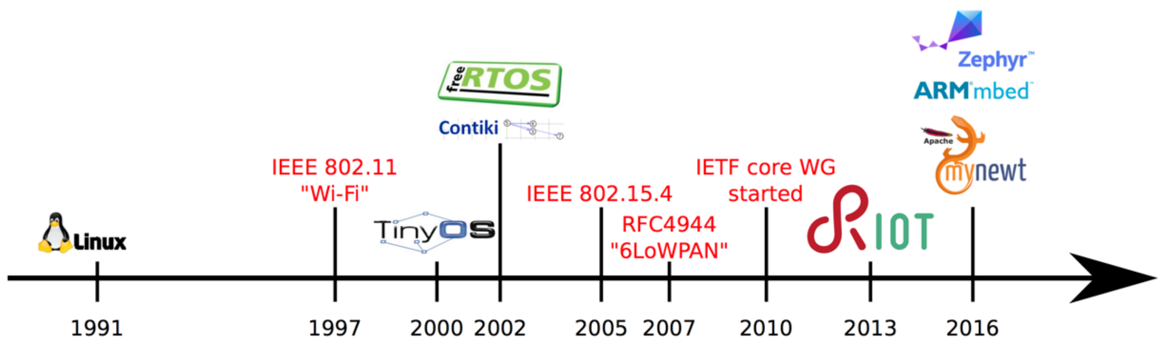

Figure 10. .

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 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 1.9. 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 1.10. Common operating systems used in IoT environment **al-fuqaha_internet_24**

4.2 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 Moore's 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	Sensitiv-ity	Chan-nels	Dis-tance
libelium__waspmote__2015	**Semtech SX1272**	863-870 (EU) 902-928 (US)	14 dBm	dBm	-134 dBm	8 13	22+ km
libelium__waspmote__2017	**rn2483**						

Table 1.11

Sensing Unit

GPS

Humidity

Temperature

4.3 Summary and discussion

5 SDN platforms

Plan de controle	Plan de gestion	Plan de doonées
Controle d'admission	Controle et supervision de QoS	Controle du trafic
Réservation de ressources	Gestion de contrats	Façonnage du trafic
Routage	QoS mapping	Controle de congestion
Signalisation	Politique de QoS	Classification de paquets
		Marquage de paquets
		Ordonnancements des paquets
		Gestion de files d'attente

Table 1.12. An example table.

qin_software_2014 Many studies have identified SDN as a potential solution to the WSN challenges, as well as a model for heterogeneous integration.

qin_software_2014 This shortfall can be resolved by using the SDN approach.

kobo_survey_2017 SDN also enhances better control of heterogeneous network infrastructures.

kobo_survey_2017 Anadiotis et al. define a SDN operating system for IoT that integrates SDN based WSN **(SDN-WISE)**. This experiment shows how heterogeneity between different kinds of SDN networks can be achieved.

kobo_survey_2017 In cellular networks, OpenRoads presents an approach of introducing SDN based heterogeneity in wireless networks for operators.

diaye_software_2017 There has been a plethora of (industrial) studies synergising SDN in IoT. The major characteristics of IoT are low latency,wireless access, mobility and heterogeneity.

diaye_software_2017 Thus a bottom-up approach application of SDN to the realisation of heterogeneous IoT is suggested.

diaye_software_2017 Perhaps a more complete IoT architecture is proposed, where the authors apply SDN principles in IoT heterogeneous networks.

oftwaredefined_2017 it provides the SDWSN with a proper model of network management, especially considering the potential of heterogeneity in SDWSN.

oftwaredefined_2017 We conjecture that the SDN paradigm is a good candidate to solve the heterogeneity in IoT.

Management architecture	**Management feature**	**Controller configuration**	**Traffic Control**	**Configuration and monitoring**	**Scapability and localization**	**Communication management**
luo_sensor_2012 Sensor Open Flow	SDN support protocol	Distributed	in/out-band	✓	✓	✓
costanzo_software_2012 SDWN	Duty sycling, aggregation, routing	Centralized	in-band	✓		
galluccio_sdnwise_2015 SDN-WISE	Programming simplicity and aggregation	Distributed	in-band		✓	
degante_smart_2014 Smart	Efficiency in resource allocation	Distributed	in-band		✓	
SDCSN	Network reliability and QoS	Distributed	in-band		✓	
TinySDN	In-band-traffic control	Distributed	in-band		✓	
Virtual Overlay	Network flexibility	Distributed	in-band		✓	
Context based	Network scalability and performance	Distributed	in-band		✓	
CRLB	Node localization	Centralized	in-band			
Multi-hope	Traffic and energy control	Centralized	in-band			✓
Tiny-SDN	Network task measurement	-	in-band			

Table 1.13. SDN-based network and topology management architectures. **ndiaye_software_2017**

6 Blockchain

6.1 Application

Blockchain Layers

- Transaction & contract layer
- Validation layer (forward validation request)
- Block Generation Layer (PoW,PoC, PoA PoS, PBFT)
- Distribution Layer

Consensus algorithms

- Proof of Work (PoW)

- Proof of Capacity (PoC)
- Proof of Authority (PoA)
- Proof of Stake (PoS)
- Proof of Bizantine Fault Tolerant (PBFT)

6.2 Summary and discussion

Bibliography

”A quote in a speech, article or book is like a gun in the hands of a soldier. It speaks with authority.”

Others

[1] V. P. Venkatesan, C. P. Devi, and M. Sivaranjani, Design of a Smart Gateway Solution Based on the Exploration of Specific Challenges in IoT, in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, Tamilnadu, India: IEEE, Feb. 2017.