Standardized Protocol Stack For The Internet Of (Important) Things

Maria Rita Palattella*, Nicola Accettura†, Xavier Vilajosana‡§¶, Thomas Watteyne§‖, Luigi Alfredo Grieco†, Gennaro Boggia† and Mischa Dohler**

* *SnT, University of Luxembourg, Luxembourg; contact email: maria-rita.palattella@uni.lu* † *Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Italy; contact email: {n.accettura,a.grieco,g.boggia}@poliba.it* ‡ *Universitat Oberta de Catalunya, Barcelona, Spain; contact email: xvilajosana@uoc.edu* § *BSAC, University of California, Berkeley; contact email: {xvilajosana,watteyne}@eecs.berkeley.edu* ¶ *Worldsensing, Spain; contact email: xvilajosana@worldsensing.com* ‖ *Dust Networks/Linear Technology; contact email: twatteyne@linear.com* ** *Centre Tecnologic de Telecomunicacions de Catalunya (CTTC), Spain; contact email: mischa.dohler@cttc.es*

Abstract

We have witnessed the Fixed Internet emerging with virtually every computer being connected today; we are currently witnessing the emergence of the Mobile Internet with the exponential explosion of smart phones, tablets and net-books. However, both will be dwarfed by the anticipated emergence of the Internet of Things (IoT), in which everyday objects are able to connect to the Internet, tweet or be queried. Whilst the impact onto economies and societies around the world is undisputed, the technologies facilitating such a ubiquitous connectivity have struggled so far and only recently commenced to take shape.

To this end, this paper introduces in a timely manner the cornerstones of a technically and commercially viable IoT which includes a detailed discussion on the particular standard of choice at each protocol layer. This stack is shown to meet the important criteria of power-efficiency, reliability and Internet connectivity. Industrial applications have been the early adopters of this stack, which has become the de-facto standard, thereby bootstraping early IoT developments.

Corroborated throughout this paper and by emerging industry alliances, we believe that a standardized approach, using latest developments in the IEEE 802.15.4 and IETF working groups, is the only way forward. We introduce and relate key embodiments of the power-efficient IEEE 802.15.4-2006 PHY layer, the power-saving and reliable IEEE 802.15.4e MAC layer, the IETF 6LoWPAN adaptation layer enabling universal Internet connectivity, the IETF ROLL routing protocol enabling availability, and finally the IETF CoAP enabling seamless transport and support of Internet applications.

The protocol stack proposed in the present work converges towards the standardized notations of the ISO/OSI and TCP/IP stacks. What thus seemed impossible some years back, i.e., building a clearly defined, standards-compliant and Internet-compliant stack given the extreme restrictions of IoT networks, is commencing to become reality.

I. Introduction

In early 2000's, Kevin Ashton from the MIT Auto-ID Center [1] proposed the term "Internet of Things", making reference to the binding of Radio Frequency Identifiers (RFID) information to the Internet. Soon, the interest for an Internet of connected objects raised the attention of governments and leading IT companies that recognized the

concept as one of their key axes for future economic growth and sustainability. The concept of Internet of Things was adopted by the European Union in the Commission Communication on RFID, published in March 2007 [2].

The Council's conclusions of November 2008 on Future Networks and the Internet, recognized that *"the Internet of Things is poised to develop and to give rise to important possibilities for developing new services but that it also represents risks in terms of the protection of individual privacy"* [3]. In 2008, the U.S. National Intelligence Council (NIC) reported that *"By 2025 Internet nodes may reside in everyday things, food packages, furniture, paper documents, and more. Today's developments point to future opportunities and risks that will arise when people can remotely control, locate, and monitor even the most mundane devices and articles. Popular demand combined with technology advances could drive widespread diffusion of an Internet of Things (IoT) that could, like the present Internet, contribute invaluably to economic development and military capability"* [4].

Although the IoT is a widely used term, its definition is still fuzzy due to the large amount of concepts it includes, and the ambiguity and opposed meaning of the two terms that compose the name "Internet of Things" [5]. Several definitions overlap in the literature, for example in [6] the IoT stands for a *"world-wide network of interconnected objects uniquely addressable, based on standard communication protocols"*. Whilst Atzoria et al. [7] considers IoT as much more than uniquely addressable objects; it envisages the existence of services that may interface Things having identities and virtual personalities operating in smart spaces and using intelligent interfaces to connect and communicate within social, environmental, and user contexts. Further conceptual designs, visions and application spaces of the IoT are exposed in [8]–[14] – all of which converge to the view that simple embedded sensor networking is now evolving to the much needed standards and Internet enabled communication infrastructure between objects.

Structurally, the IoT requires software architectures that are able to deal with a large amounts of information, queries, and computation, making use of new data processing paradigms, stream processing, filtering, aggregation and data mining, all of this sustained by communication standards such as HyperText Transfer Protocol (HTTP) [15] and Internet Protocol (IP) [16]. In contrast, due to the nature of IoT objects, very low power consumptions are required so any object can plug into the Internet while being powered by batteries or through energy-harvesting. Energy is wasted by transmission of unneeded data, protocol overhead, and non-optimized communication patterns; these need to be taken into account when plugging objects into the Internet. Existing Internet protocols such as HTTP and Transmission Control Protocol (TCP) [17] are not optimized for very low-power communications, due to both verbose meta-data and headers, and the requirements for reliability through packet acknowledgment at higher layers, which hinders the adaptation of existing protocols to run over that type of networks.

The objects conforming to the IoT have a wide range of connotations and understandings, including RFID [5], Wireless Sensor Networks (WSNs), Machine-to-Machine (M2M) [18], [19], among others. However, in the actual sense of its name, the IoT pertains to the ability to interconnect as well as Internet-connect objects, things, machines, etc. There are three core requirements related to this ability:

- **A Low Power Communication Stack.** The majority of objects will not be able to draw power from the mains, and have batteries at best. This means that finding enough energy to power processing and communication is

a major challenge. Whilst we are ready to recharge our mobile phones on a daily basis, changing batteries in millions of objects is impractical, at best. Any stack must therefore exhibit a low average power consumption. Indeed, the stack discussed in this paper obeys precisely this requirement.

- **A Highly Reliable Communication Stack.** Although the Internet is a best-effort transport medium, protocols incorporate error detection, retransmissions and flow control. These techniques are applied at various protocol layers concurrently, which leads to a reliable end-to-end experience, albeit in a rather inefficient way. For the IoT to merge seamlessly into the Internet, it needs to offer the same reliability we are used to on the Internet – with the additional requirement that said reliability is achieved at highest possible efficiency.
- **An Internet-Enabled Communication Stack.** Enabling another dialect of the Internet has profound implications on the protocol design. The Internet is exhibiting emergent behavior today because communication is bi-directional; it is hence of utmost importance to ensure that communication from objects but also towards objects is facilitated. Furthermore, the explosion of the Internet can arguably be attributed to the ability of any machine around the world to talk to any other machine, all this facilitated by one universal language, IP; it is hence of paramount importance that the IoT is IP enabled [13]. This in turn calls for standardized approaches, which is core to the exposition in this paper.

Identifying the requirement above has taken some time, as shown in Table I. Early conceptual designs can be traced back to the emergence of the Distributed Sensor Networks program [21] at the MIT Lincoln Labs [22]. It aimed at developing and extending target surveillance and tracking technology in systems that employ multiple spatially distributed sensors and processing resources. The WSN field really took off with the concept of "Smart Dust" and the eponymous DARPA project [23] - [24], lead by Prof. Kristofer Pister from the University of California, Berkeley. In the following years, the concept of a ubiquitous WSN has not only been demonstrated but also awaken commercial interest. A variety of pioneering companies thus emerged, such as Dust Networks, Ember, Millennial, Sensicast, Moteiv and Arch Rock. All of these companies had in one form or another their proprietary hardware and communication stack. It was quickly recognized however that having a multitude of proprietary systems connected to the Internet does impede the much hoped-for scalability and explosion of the IoT.

Starting in 2003, various IEEE and IETF standardization bodies commenced putting a framework to the communication protocols of the emerging systems.

The standard with the longest-standing impact is IEEE 802.15.4 [25], which defines a low-power Physical (PHY) layer, and upon which most IoT technologies have built. It also defines a Medium Access Control (MAC), which has been the foundation of ZigBee 1.0 and ZigBee 2006 [26]. It became soon clear that the single-channel nature of this MAC protocol caused its reliability to be unpredictable, especially in multi-hop settings. An alternative which uses channel-hopping to combat multipath fading and external interference was developed and commercialize by Dust Networks [27]. This protocol, called Time Synchronized Mesh Protocol (TSMP) [28], became the de-facto standard for reliable low-power wireless in industrial application, and was turned into the WirelessHART standard [29] in 2008. In 2011, it was integrated into the IEEE 802.15.4 standard through the IEEE 802.15.4e working group, and will thus become a default MAC protocol in the next revision of the IEEE 802.15.4 standard. With a

TABLE I
EVENTS WHICH HAVE HELPED SHAPING THE WORLD OF THE IOT.

Year	Event
1967	REMBASS Remotely Monitored Battlefield Sensor System
1978	Distributed Sensor Networks for Aircraft Detection Lincoln Labs - Lacoss
1992	RAND Workshop - Future Technology Driven Revolutions in Military Conflict. Concepts behind Smart Dust emerge.
1993-1996	DARPA ISAT studies - many WSN ideas and applications discussed. Deborah Estrin leads one of the studies.
1994	LWIM - Low Power Wireless Integrated Microsensors - Bill Kaiser (UCLA)
1997	Smart Dust proposal written, Kris Pister (Berkeley)
1998	Seth Hollar makes wireless mouse collars
1999	Endeavour project proposed by Randy Katz, David Culler (Berkeley) PicoRadio project started by Jan Rabaey (Berkeley)
2000	Crossbow begins selling 'Berkeley motes'
2001	Multiple demos proving viability
2002	Dust, Ember, Millennial, Sensicast founded
2003	IEEE 802.15.4-2003 standard Moteiv (now Sentilla) founded
2004	ZigBee 1.0 standard ratified TSMP 1.1 shipping
2005	Arch Rock founded
2006	ZigBee 2006 standard ratified IEEE 802.15.4-2006 standard
2007	WirelessHART standard ratified IETF 6LoWPAN's RFC4944 published WirelessHART shown to achieve 99.999% reliability [20]
2008-2009	IETF workgroup Routing Over Low-power Lossy links (ROLL) created IEEE 802.15.4e work group created
2010-2011	IEEE 802.15.4e's MAC protocol ratified IEFT 6LoWPAN's RFC4944 updated IETF ROLL's RPL routing protocol ratified

power-efficient and reliable link layer , it was now possible to connect these networks to the Internet. This was the trigger for the birth of various IETF WGs, notably 6LoWPAN [30] as a convergence layer, ROLL RPL [31] as a routing protocol, and CoAP [32] facilitating native support of current Internet applications.

These layers and their tight interaction are hence seen as instrumental in making the IoT happen from a technology point of view , and are thus surveyed and described in great details in subsequent sections. With respect to prior art, the core contributions of this paper can be summarized as follows:

- For the first time, a detailed survey on the new IEEE 802.15.4e MAC and IETF ROLL RPL routing standards is provided; indeed, these standards have been discussed in various sources at different technical depths but such a detailed exposure with explanations is unprecedented.
- The introduction of a clear vision on a workable communication stack for the IoT using proven standards and

the accumulated expertise of the authors, is also unprecedented.

- Finally, this paper shun away from using high-level approaches to defining the IoT but rather introduced concrete protocols as well as the reasoning why they will succeed and last.

The rest of the paper is organized as follows. Section II introduces the IEEE 802.15.4-2006 PHY, along with the main features of low-power radio hardware suitable for the IoT protocol stack. Section III then details the IEEE 802.15.4e Time Synchronized Channel Hopping (TSCH) MAC protocol, showing how it is able to provide both high reliability and energy-efficiency. IETF 6LoWPAN (IPv6 addressing) and IETF ROLL RPL (routing) are covered in Section IV and Section V, respectively. We then discuss the novelty introduced by the CoAP protocol at the application layer in Section VI. Section VII finally concludes this article and presents possible paths for future research.

II. Low-Power PHY Layer – IEEE 802.15.4-2006

The PHY layer determines the physical radio frequency at which the radio operates, the radio modulation, and the encoding of the signal. In this Section we briefly review the importance of a low-power hardware and PHY layer which, together with an energy-efficient MAC layer, cater for lower power connectivity among smart objects in the future IoT.

A. Low-Power Radio Hardware

A radio translates bytes of digital information into an electromagnetic signal for transmission over the air. When transmitting, the radio uses a modulation scheme to encode bytes of data into an analog signal. This signal is then amplified by a Power Amplifier (PA) before being sent over the antenna. A low-power radio typically outputs $0\ dBm$, or $1\ mW$. Fading and shadowing causes the amplitude of that signal to weaken as it travels through the air. When it is picked up by the receiver antenna, the signal is too weak to be demodulated directly, and the radio uses a Low-Noise Amplifier (LNA) to bring it to a level the demodulator can handle. The power of the weakest signal that the radio can successfully receive is called its sensitivity. A radio with a $-90\ dBm$ sensitivity (a typical number for low-power radios) can successfully demodulate signals as weak as $1\ pW$.

When on, the modulator, demodulator, PA and LNA all consume substantial amounts of current, making the radio the most power-hungry component of most designs. The radio, however, does not consume any energy when off. The challenge of a good communication stack is to enable reliable transmission of data, while keeping the radio off most of the time. Radio duty cycle is the portion of time the radio is on, either transmitting or receiving. It is a good indicator of the power consumption of the mote. An energy-efficient communication stack has a duty cycle (far) lower than $1\ \%$.

Table II lists data sheet numbers for commercially-available low-power radios from different vendors. Radios can change the power they transmit at, usually over a range going from -50 dBm to +5 dBm. In detail, 0 dBm (i.e, 1 mW) is the default transmit power for most radios; we therefore choose to show the transmit current at that power. The receive current is the current drawn by the radio when in receive mode; radio draw the same current when

TABLE II
COMPARISON OF DIFFERENT 802.15.4-COMPLIANT LOW-POWER RADIOS.

vendor	name	sensitivity	transmit current @ 0 dBm	receive current
Atmel	AT86RF231	-101 dBm	14.0 mA	12.3 mA
Dust Networks	DN6000	-91 dBm	5.4 mA	4.5 mA
Ember	EM357	-100 dBm	27.5 mA	25.0 mA
Freescale	MC13233	-94 dBm	26.6 mA	34.2 mA
Microchip	MRF24J40	-95 dBm	23.0 mA	19.0 mA
NXP/Jennic	JN5148	-95 dBm	15.0 mA (1.8 dBm)	17.5 mA
Texas Instruments	CC2520	-98 dBm	25.8 mA	18.8 mA

idle listening (without receiving a packet), or actively receiving bytes. A common misconception is to consider a system is energy-efficient when the motes transmit few packets. Table II shows that radios draw about the same current in transmit and receive mode. Optimizing the power consumption of a system therefore consists in lowering the duty cycle of the radio as a whole, i.e. having the radio off most of the time.

In most designs, motes are battery-powered, and it is impractical to change batteries. The goal of an energy-efficient solution is to increase the lifetime of a mote by decreasing its average current consumption. This can be done by choosing a radio which draws little current, and by using a protocol which runs the radio at a low duty cycle. Let's take some examples by assuming the mote is powered by a pair of AA batteries, holding 3000 mAh of charge. If we use the AT86RF231 (which draws ≈ 13 mA when on) and a protocol which left the radio on all the time (i.e. radio duty cycle is 100 %), the batteries will be depleated in 3000 $mAh/13$ $mA = 230$ h, or 10 days. If, on the same hardware, we use a protocol with a 1 % radio duty, the lifetime increases by a factor of 100, to 32 months. If, on top of that, we replace the radio by the lower-power DN6000 (which draws ≈ 5 mA when on), the lifetime increases to 3000 $mAh/(5\ mA \times 1\%) = 60000$ h, or 7 years.

B. PHY of IEEE 802.15.4-2006

The most prominent standard in low-power radio technology is IEEE 802.15.4 [25]. It defines both the PHY layer (e.g., the modulation scheme used) and the MAC layer (e.g., in a network, which mote talks when, on which channel). The first revision of the standard was published in 2003, with a revision in 2006. Several working groups are currently working on improving the standard in preparation for its next revision. These groups are identified by a letter, e.g. IEEE 802.15.4e to be discussed below.

The IEEE 802.15.4 PHY is a healthy trade-off between energy-efficiency, range, and data rate targeted at building-sized networks. While the current standard defines multiple PHY layers, the most widely used is the one operating in the $2.4 - 2.485$ GHz frequency band, a worldwide and unlicensed band.

In this band, the IEEE 802.15.4 PHY layer uses Offset-Quadrature Phase-Shift Keying (O-QPSK) modulation with a 2 $Mbps$ physical data rate. Internally to the radio, every group of 4 bits of data sent for transmission are encoded as 32 chips ('physical bits') by following some simple lookup table. From a user's perspective, the bitrate appears to be 250 $kbps$, although internally 8 more chips are sent over a 2 $Mcps$ link. This technique is referred

to as Direct Sequence Spread Spectrum (DSSS) and known to yield extra robustness.

IEEE 802.15.4 defines 16 frequency channels, located every 5 MHz between 2.405 GHz and 2.480 GHz. The channels themselves are only 2 MHz wide, so channel i does not interfere with channel $i-1$ or $i+1$; channels are said to be orthogonal. The radio can arbitrarily send and receive on any of those channels, and every compliant radio is able to switch channels in no more than 192 us.

When a radio sends a packet, it starts by transmitting a physical preamble for 128 us to allow for the receiver to lock to its signal. It then sends a well-known Start of Frame Delimiter (SFD) to indicate the start of the physical payload. The first byte of the physical payload indicates the length (in bytes) of the payload itself. Its maximum value is 127, which limits the length of a packet to 128 bytes when including the length byte. A radio listening continuously demodulates what it hears. When no other mote is transmitting, it hears white noise and the stream of bits coming out of the demodulator is random. The circuitry in the receiver looks for a physical preamble to 'lock onto'. Once locked on, the receiver waits for the SFD, then for the length byte. It then fills a receive buffer with the number of bytes indicated in the length byte, after which it can switch off safely. After successfully receiving a packet, the radio indicate reception to the micro-controller. From a implementer's point of view, the only requirements are to send packets of at most 128 bytes, and to have the first byte indicate how many bytes follow. Although in a protocol stack the bytes following the length byte comply to different header formats (e.g. MAC, routing, transport), they can be arbitrary as far as the radio is concerned.

III. Power-Saving Link Layer – IEEE 802.15.4e

IEEE 802.15.4 also defines a MAC protocol, i.e. the layer interacting directly with the radio. It defines the format of the MAC header (with fields such as source and destination address) and how motes can communicate with each other. This MAC layer is geared toward star networks, in which all motes communicate directly with a special coordinator mote. It is ill-suited for low-power multi-hop networking mainly because of the following reasons:

- **Powered routers.** While it is possible to use the existing IEEE 802.15.4 MAC protocol for multi-hop routing, the motes which are relaying the data need to keep their radio on all the time (100 % duty cycle). In a mesh network, each mote also acts as a router. Using the existing IEEE 802.15.4 MAC protocol leads to a power hungry solution.
- **Single channel operation.** The wireless medium is unreliable in nature and propagation effects, such as shadowing and multi-path fading, as well as temporarily varying interference can cause wireless links to break. If the network operates on a single channel, this causes network instabilities which can lead to network collapse.

The IEEE 802.15.4e working group was created in 2008 to redesign the IEEE 802.15.4 MAC protocol. Through time synchronization and channel hopping, it enables high reliability while maintaining very low duty cycles, both highly recommended requirements for the emerging IoT. IEEE 802.15.4e [33] is only a MAC protocol change, which does require any changes to the hardware.

Time Synchronized Channel Hopping (TSCH), part of IEEE 802.15.4e draft standard since 2010, is the latest generation of highly reliable and low-power MAC protocol, and thus very suitable for a protocol stack for IoT. The initial concept emerged in the proprietary Time Synchronized Mesh Protocol (TSMP) [28] in 2006. Following its successful adoption by the industrial automation community, TSMP was standardized in ISA100.11a [34] in 2008 and WirelessHART [29] in 2009. While those different standards might have adopted different packet formats and higher-layer commands, the underlying technology is the same: synchronize the motes for energy efficiency, channel hop for reliability.

A. Slotframe Structure

In TSCH, motes synchronize on a slotframe structure. A slotframe is a group of slots which repeat over time. Each mote follows a schedule which tells it what to do in each slot. In a given slot, a mote can either transmit, receive, or sleep. In a sleeping slot, the mote does not turn on its radio. For each active slots, the schedule indicates with which neighbor to transmit or receive, and on which channel offset (detailed in Sec. III-C).

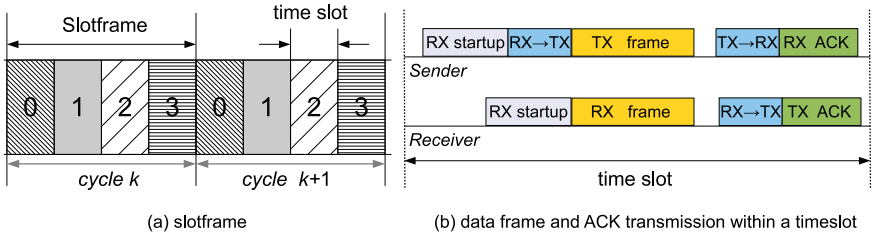

Fig. 1. A 4-slot slotframe and timeslot diagram of an acknowledged transmission.

As shown in Fig. 1, a single slot is long enough for the transmitter to send a maximum length packet, and for the receiver to send back an acknowledgment indicating good reception. While the duration of a slot is implement-specific, 10 ms is a possible value suggested in the draft [33].

When an upper layer generates a packet, it sends it to the MAC layer which stores the packet in a transmit queue. At each transmission slot, the MAC layer checks whether it has a packet in its queue destined to the neighbor associated with that slot. If not, it goes back to sleep without turning its radio on. If yes, it transmits the packet and waits for the ACK. If an ACK is received, it removes the packet from the queue. Otherwise, it keeps the packet in the queue for future re-transmission. A number of retransmissions is kept for every packet to avoid staleness.

At each reception slot, a mote turns on its radio right before the time it expects to receive the packet. If it receives a packet destined for it, it sends an acknowledgment, turns off its radio, and forwards the packet to the upper layer for processing. If it does not receive anything after some timeout, it returns to sleep. This means that either the transmitter had nothing to say, or that the packet got lost due to interference or fading.

Fig. 2 shows an example topology and the associated schedule. Here, the slotframe is only 5 slots long, and there are 6 channel offsets (the concept of channel offset is presented in Sec. III-D). Each mote in the network only cares about the cells it participates in. For example, when G has a packet to send to D, it waits for slot 3, and sends it

on channel offset 0. It will stay off for the other cells. If a packet needs to go from G to A, it will first be sent from G to D, be buffered at D, then sent from D to A in the next frame. Note that, as depicted in Fig. 2, while most cells are dedicated, some can be shared between different links (e.g. $D \rightarrow A$ and $C \rightarrow A$). IEEE 802.15.4e defines a simple backoff scheme for shared cells in case a collision occurs.

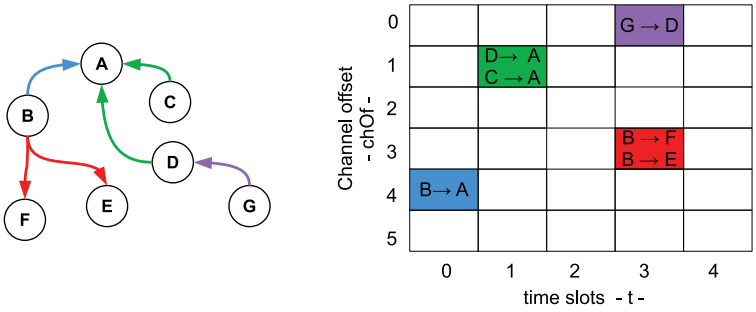

Fig. 2. Dedicated and shared links.

B. Scheduling

IEEE 802.15.4e defines how the MAC layer *executes* a schedule (as described in Sec. III-A). It does not specify how such as schedule is built. A schedule needs to be built carefully so that, when mote A has a transmit slot to mote B, B is actually listening for packets from A. Similarly, if A is no longer a neighbor of B (e.g. it moved or switched off), B should not be listening anymore for packets from A. While all these rules are intuitive, they also illustrate that the network's schedule needs to be both carefully built, and constantly refreshed as links come and go in the network. Scheduling can either happen in a centralized or a distributed way:

- In a **centralized** approach, a specific "manager" mote is responsible for building and maintaining the network schedule. Every mote in the network regularly updates the manager with the list of other motes it can hear, and the amount of data it is generating. From the neighbor information, the manager draws the connectivity graph. From the data generation demands, it assigns slots to the different links in the connectivity graph. Once this schedule is built, the manager then informs each mote about the links in the schedule it is participating in. The motes then simply follow these instructions. When there is a change in the connectivity (i.e., a mote lost a neighbor), the manager updates its schedule and informs the affected motes.

  In practice, networks often have a gateway mote which connect it to the Internet (see Sec. IV). In a centralized approach, that mote often also manages the IEEE 802.15.4 schedule. This type of approach has been commercially available since TSMP, and thousands such networks deployed.

  A centralized approach builds very efficient schedules. Since the manager knows exactly what the network looks like, it can apply centralized scheduling techniques. Yet, such an approach does not cope well with very dynamic network (e.g. mobile robots), since there is some delay between a link failing and the schedule being adapted.

- In a **distributed** approach, motes decide locally on which links to schedule with which neighbors. Once can imagine opting for this approach in mobile networks, or when the network has many gateway motes. The simplest solution is for each node to schedule a link to each neighbor. This is the approach adopted by Tinka et al. [35], which evaluates this approach by simulation and experimentally. A more complex problem is when multiple motes generate data at a constant rate, which requires links to be scheduled along the multi-hop route this data travels over. Internet-like reservation protocols such as the Resource Reservation Protocol (RSVP) [36] and the Multi Protocol Label Switching (MPLS) protocol [37] could be applied, but this is remains a very open issue.

C. Synchronization

Device-to-device synchronization is necessary to maintain connectivity with neighbors in a slotframe-based network. As shown in Fig. 1, no beacon is transmitted in a IEEE 802.15.4e network for TSCH applications.

Two methods are defined for allowing a device to synchronize to the network: (I) *Acknowledgment-Based* synchronization and (II) *Frame-Based* synchronization. The former involves the receiver calculating the delta between the expected time of frame arrival and its actual arrival, and providing that information to the sender mote in its acknowledgment. This allows a sender mote to synchronize to the clock of the receiver. The latter involves the receiver calculating the delta between the expected time of frame arrival and its actual arrival, and adjusting its own clock by the difference. This allows a receiver mote to synchronize to the clock of the sender.

When there is traffic in the network, motes which are communicating will implicitly re-synchronize using the data frames they exchange. If they have not been communicating for some time (typically 30 s), motes will exchange an empty data frame (called keep-alive messages) simply to re-synchronize.

In a typical IEEE 802.15.4e TSCH network, time propagates outwards from the PAN coordinator. It is very important to maintain unidirectional time propagation and avoid timing loops. Each device periodically synchronizes its network clock to at least one other device, and it also provides its network time to its neighbor motes. Each mote determines whether to follow a neighbors clock based on the presence of a *ClockSource* flag in the corresponding neighbors record (configured by the network manager in a centralized system). It has to be specified that the direction of time propagation is independent of data flow in the network.

D. Channel Hopping

The IEEE 802.15.4e TSCH MAC adds channel hopping to time slotted access. Channel hopping implies frequency diversity that mitigates the effects of interference and multipath fading. Moreover, the use of several frequencies increases the network capacity, because more motes can transmit their frames at the same time, using different channel offsets. Channel hopping combined with slot access improves also reliability. The advantages derived from the channel hopping have been already tested and analyzed [38], [39].

Let $(t, chOf)$ be the slot and the channel offset, respectively, assigned to a given link. The channel offset, *chOf*,

is translated to a frequency f (i.e., a real channel) using the following translation function

$$f = F\{(ASN + chOf) \bmod n_{ch}\}, \tag{1}$$

where *ASN* is the *Absolute Slot Number*, i.e., the total number of slots that elapsed since the network was deployed. The ASN is incremented at each slot and shared by all the devices in the network. In detail, $ASN = (k \cdot S + t)$, where k is the slotframe cycle, as shown in Fig. 1. The function F is realized with a look-up-table, containing the set of available channels. The value n_{ch} (i.e., the number of available physical frequencies) is the size of such a look-up-table. Moreover, the following constraints on t and $chOf$ hold: $0 \leq t \leq S - 1$, and $0 \leq chOf \leq n_{ch} - 1$. In an IEEE 802.15.4e network, 16 channels are available. Furthermore, a blacklist can be used to restrict the set of allowed channels for coexistence purposes. If the slotframe size, S, and the number of channels, n_{ch}, are relatively prime, the translation function assures that each link rotates through k available channels over k slotframe cycles. In other words, successive frames over a same link are sent over different physical frequencies in successive slotframe cycles k.

E. Network Formation

IEEE 802.15.4e TSCH networks support the same two classes of devices considered in the IEEE 802.15.4 standard [25]: the Full Function Device (FFD), which can act as a simple mote or as a network coordinator, and the Reduced Function Device (RFD), which cannot become a network coordinator and hence only talks to a network coordinator. A network topology is composed by a combination of FFD and RFD devices.

The network formation procedure for TSCH applications includes two components: *advertising* and *joining*. As a part of advertising, only FFD devices that are already members of the network can send command frames announcing the presence of the network. A new device trying to join the network, instead, listens for the *Advertisement* command frames. When at least one of these frame is received, the new mote can attempt to join the network sending a *Join Request* command frame to an advertising device. In order to fulfill this task, two consecutive slots in the slotframe are required: the first one for broadcasting *Advertisement* frames and the second one for receiving *Join Request* frames. In fact, given that a single packet can be transmitted in a single slot, a new device that listens to the *Advertisement* message in a given slot, will send the *Join Request* frame in the next slot.

In a centralized management system, *Join Request* frames are routed to the PAN coordinator. In a distributed management system, they can be processed locally. When a new mote is accepted into the network, the advertiser activates the mote by setting up slotframes and links between the new mote and other existing ones. These slotframes and links can also be deleted and/or modified after a mote has joined the network.

1) Network Ramp-Up: In order to aid the understanding of the network formation procedure, Fig. 3 shows the messages exchanged during the network build up phase in a simple scenario with a PAN coordinator (mote A), a FFD (mote B) and a RFD (mote C), using a 7 slots slotframe. For each exchanged message we specify the slot ASN used for that transmission. Moreover, we mark the beginning of each new slotframe occurrence, specifying

the value of the slotframe cycle, k. As it can be seen in Fig. 3, several messages are sent in consecutive slotframe cycles.

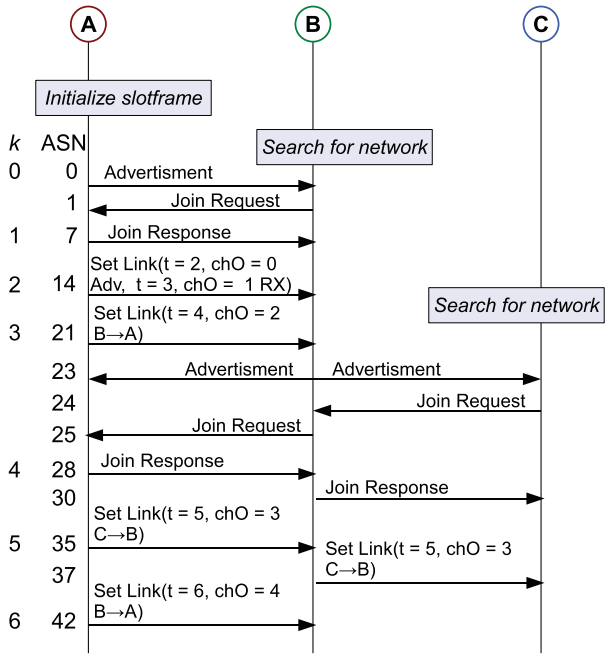

Fig. 3. Messages exchanged during the network build up phase among the motes in a simple network with a PAN coordinator, a FFD and a RFD.

We suppose that the TSCH network is based on a centralized management system. In other words, the PAN coordinator works as master node and, during the *network build up* phase, it defines the links for allowing broadcast and dedicated communications in the network. Being the first mote in the network, the PAN coordinator starts one slotframe, to which other motes may later synchronize. It reserves the first two slots within the slotframe to itself for broadcasting the *Advertisement* command frames and receiving the *Join Request* frames, respectively. As soon as other devices join the network, the PAN coordinator assigns slots and channel offsets to each of them.

Fig. 4 shows the links scheduled within the 7 slots slotframe, during the network build up phase. In this example the PAN coordinator assigns the same channel offset value, *chOf* = 0, to all the broadcast links, and *chOf* = 1 to all the dedicated links used for receiving Join Request frames.

As already specified, the use of the same *chOf* does not imply the use of the same channel. In fact, in the frequency translation function, given by Eq. (1), the value of *chOf* is the same, but the value of t is different, and therefore it results in a different channel to be used.

In the considered network scenario, mote C is a simple RFD device that cannot allow other motes to join the network. For this reason, the links used for advertising and joining the network are not set for it. In order to allow frames generated by mote C to reach the PAN coordinator, a slot is reserved for each dedicated link along the path that goes from mote C to the root mote (i.e., $C \rightarrow B$, $B \rightarrow A$).

Once all the motes have joined the network, i.e., the PAN coordinator has not received any *Join Request* frames

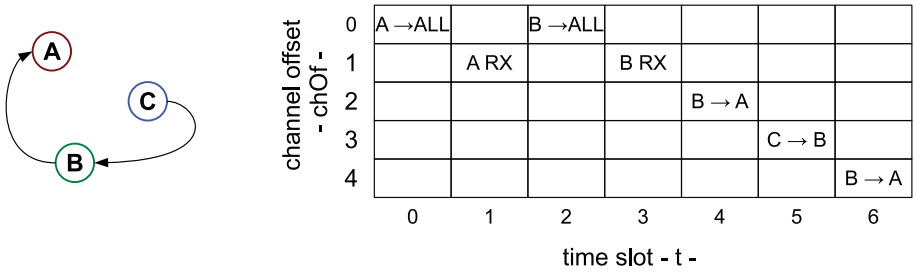

Fig. 4. Time pattern within the 7 slots slotframe defined during the network build up phase in Fig. 3.

during a timeout period, the Advertising procedure can be disabled. Afterwords, the procedure could be activated again, with a given frequency, in order to check if there are new devices that wait to join the network.

IV. Connecting to the Internet – IETF 6LoWPAN

As well known, the Internet is composed of many networks, a number of which are typically traversed by a packet on its way from source to destination. Thus, for each type of link layer technology which the network is based on, there needs to be an "IP-over-X" specification that defines how to transport IP packets. In many cases, to map the services required by the IP layer onto the services provided by the lower layer (i.e, the link layer), the "IP-over-X" specification can amount to a (sub)layer of its own, often called *adaptation layer* [40]. In the process of shaping the IoT world, in 2007 the IETF IPv6 over Low power WPAN (6LoWPAN) working group has started to work on specifications for transmitting IPv6 over IEEE 802.15.4 networks.

Typically, Low power WPANs are characterized by: small packet sizes[1] , support for addresses with different lengths, low bandwidth, star and mesh topologies, battery supplied devices, low cost, large number of devices, unknown node positions, high unreliability, and long idle periods during which communications interfaces are turned off to save energy [30], [41] – [44].

Given the aforementioned features, it is clear that the adoption of IPv6 on top of a Low power WPAN is not straightforward, but poses strong requirements for optimization of this adaptation layer. For instance, due to the IPv6 default minimum MTU size of 1280 bytes, a no-fragmented IPv6 packet would be too large to fit in an IEEE 802.15.4 frame; moreover, the overhead due to the 40 bytes long IPv6 header would waste the scarce bandwidth available at the PHY layer.

For these reasons, the 6LoWPAN working group has devoted huge efforts for defining an effective adaptation layer in [45]- [46]. Further issues encompass the auto-configuration of IPv6 addresses [47], the compliance with the recommendation on supporting link-layer subnet broadcast in shared networks [48], the reduction of routing and management overhead, the adoption of lightweight application protocols (or novel data encoding techniques),

[1]In an IEEE 802.15.4 WPAN, the maximum length for a packet to be transmitted on physical layer is 127 bytes.

and the support for security mechanisms (i.e., confidentiality and integrity protection, device bootstrapping, key establishment and management).

A. 6LoWPAN frame format

To solve the problem of manage IPv6 packets, allowing link-layer forwarding and fragmentation, 6LoWPAN uses an intermediate adaptation layer between IPv6 and IEEE 802.15.4 MAC levels [45]. Moreover, 6LoWPAN may compress IPv6 header and *Next Headers*, by suppressing redundant information that can be inferred from other layers in the communication stack [46].

Specifically, all 6LoWPAN encapsulated datagrams (that should be transported over IEEE 802.15.4 MAC) are prefixed by astack of headers, each one identified by a type field. In particular, the header types can be logically grouped in four categories, depending on the function they play in the 6LoWPAN adaptation strategy, as shown in Table III and summarized below:

- a *NO 6LoWPAN Header* is used for specifying that the received packet is not compliant to 6LoWPAN specifications and therefore it has to be discarded (in this way allowing the coexistence with other no-6LoWPAN nodes in the same network).
- A *Dispatch Header* is used to compress an IPv6 header or to manage link-layer multicast/broadcast.
- A *Mesh Addressing Header* allows IEEE 802.15.4 frames to be forwarded at link-layer, extending single-hop WSNs in multi-hop ones.
- A *Fragmentation Header* is used when a datagram does not fit within a single IEEE 802.15.4 frame.

It is worth to note that each header may be present or not, depending on the needs. Moreover, headers should appear in a precise order, as described in the sequel.

TABLE III
6LOWPAN HEADER TYPES.

First 2 bits		**Following bit combinations**	
NO 6LoWPAN	00	xxxxxx	Any combination
Dispatch	01	000000	Additional Dispatch byte follows
		000001	Uncompressed Ipv6 Addresses
		000010	LOWPAN_HC1 compressed IPv6
		010000	LOWPAN_BC0 broadcast
		1xxxxx	LOWPAN_IPHC compressed IPv6
Mesh Addressing	10	xxxxxx	Any combination
Fragmentation	11	000xxx	First Fragmentation Header
		100xxx	Subsequent Fragmentation Header

The IEEE 802.15.4 standard does not define any routing capability and relies on functionalities of upper layers to do this. At this aim, a routing protocol that can be used for populating the routing table will be described in Sec. V. Anyway, two devices do not require direct reachability in order to communicate because an *Originator* device

may use other intermediate devices as forwarders toward the *Final Destination* device. To realize the frame delivery using a unicast communication, a *Mesh Addressing Header* is used prior to any other headers of the 6LoWPAN encapsulation. For each forwarder node, it includes the link-layer addresses of the considered forwarding node and of the next-hop node, in addition to the the link-layer addresses of the *Originator* and of the *Final Destination*.

When some form of multicast/broadcast communication at link layer is needed for controlled flooding mechanisms (e.g., the one described in Sec.V) or for topology discovery, a *Broadcast Header* immediately follows the *Mesh Addressing Header* (if present). It is a kind of *Dispatch Header* (see below) and it includes a 1-byte long *Sequence Number* for detecting and, thus, suppressing duplicate packets.

The *Fragmentation Header* can be used for fragmentation purposes and it must follow *Mesh Addressing* and *Broadcast* headers, if present. Such a header includes: the *Datagram Size*, that is the dimension of the entire IP packet before link layer fragmentation (it shall be the same for all link layer fragments of an IP packet); the *Datagram Tag*, which identifies univocally the original fragmented IP packet; and the *Datagram Offset* that specifies the offset of the fragment from the beginning of the payload of the datagram (obviously it is present only in the second and subsequent fragments.)

Finally, the *Dispatch Header* category includes several kinds of headers, used for encapsulating and, optionally, compressing an IPv6 packet. Therefore, a *Dispatch Header*, except the *Broadcast* one, must follow all the other ones described till now. The 6LoWPAN specifications consider the *LOWPAN_IPHC* encoding scheme for compressing IPv6 header, as defined in IETF RFC 6282 [46]. It substitutes the original scheme suggested in the IETF RFC 4944 [45]. However, new implementations of 6LoWPAN should support *LOWPAN_HC1* decompression for backward compatibility issues. Therefore, both compression schemes are identified by different *Dispatch Types* (see Table III).

B. Header compression

Within the same WPAN, many IPv6 header fields are expected to be common and/or easy to derive without requiring their explicit indication by the sender. As an example, the *Payload Length* can be inferred either from the *MAC Frame Length* or from the *Datagram Size* field in the fragment header (if present); *Hop Limit* will be set to a well-known value by the source; addresses assigned to 6LoWPAN interfaces are formed with an *Interface Identificator* derived directly from MAC addresses.

The *LOWPAN_IPHC* encoding scheme performs effective compression of unique local, global, and multicast IPv6 addresses, based on shared states. To this end, a 13-bit *LOWPAN_IPHC* encoding field is appended to the first 3 bits of the *Dispatch Type*. If some of the IPv6 header fields have to be carried in clear, they follow the *LOWPAN_IPHC* encoding. In the best case, the *LOWPAN_IPHC* can compress the IPv6 header down to 2 bytes in an IPv6 link-local communication (i.e., a direct single-hop communication). When a packet is routed through multiple hops, *LOWPAN_IPHC* can compress the IPv6 header down to 7 bytes.

6LoWPAN provides also a technique to compress IPv6 next-headers, namely the *LOWPAN_NHC* encoding. Compression formats for different next-headers are identified by a variable-length bit-pattern which immediately

follows the *LOWPAN_IPHC* compressed header. Each next-header in the original IPv6 packet will be present in the compressed one in the same order and it will encoded with the appropriate *LOWPAN_NHC* format.

Finally,the RFC 6282 allows a compression format for UDP headers using *LOWPAN_NHC*. The UDP Length field is always elided, as it can be inferred from lower layers using the 6LoWPAN *Fragmentation Header* or the IEEE 802.15.4 header. The Checksum field can be also elided if authorized by upper layers. Source and destination ports can be compressed if they match some common cases and, hence, the compression result is carried in-line after the *LOWPAN_NHC* encoding field; the length of the compression result can range from a minimum of 8 bits (i.e., 4 bits for each port) to 32 bits (i.e., both ports are not compressed). Not compressed or partially compressed fields are carried in-line, appearing in the same order as they do in the original UDP header. In the best case, an UDP header can be compressed to only 2 bytes, i.e., one byte for the *LOWPAN_NHC* encoding field and the other one for the compressed ports.

V. Routing – IETF ROLL

Routing issues are very challenging for 6LoWPAN, given the low-power and lossy radio-links, the battery supplied nodes, the multi-hop mesh topologies, and the frequent topology changes due to mobility. Successful solutions should take into account the specific application requirements, along with IPv6 behavior and 6LoWPAN mechanisms [40]. An effective solution is being developed by the IETF "Routing Over Low power and Lossy (ROLL) networks" working group. Recently, it has proposed the leading IPv6 Routing Protocol for Low-power and Lossy Networks (LLNs), RPL, based on a gradient-based approach [31], [43], [44], [49].

RPL can support a wide variety of different link layers, including ones that are constrained, potentially lossy, or typically utilized in conjunction with host or router devices with very limited resources, as in building/home automation, industrial environments, and urban applications [50]- [53]. It is able to quickly build up network routes, to distribute routing knowledge among nodes, and to adapt the topology in a very efficient way; thus, it is suitable also for smart grid communications [54].

The information dissemination mechanism of RPL is regulated by the so called *trickle timer* [55], that has to be properly tuned in order to assure a small signaling overhead and fast path repair operations [56] [57].

In the most typical setting entailed by RPL, the nodes of the network are connected through multi-hop paths to a small set of root devices, which are usually responsible for data collection and coordination duties. For each of them, a Destination Oriented Directed Acyclic Graph (DODAG) is created by accounting for link costs, node attributes/status information, and an Objective Function, which maps the optimization requirements of the target scenario.

Each DODAG is identified with a *DODAGID*, which is set upon its creation, and with a *DODAGVersionNumber*, which is updated each time the graph is rebuilt. The topology is set-up based on the *Rank* metric, which encodes the distance of each node with respect to its reference root, as specified by the Objective Function. Regardless the way it is computed (see Sec. V-D for more details), the *Rank* should monotonically decrease as the DODAG is followed towards the DODAG destination, in accordance to the gradient-based approach.

The identification of the different kinds of traffic encompassed by RPL is a basic preliminary step needed to understand the main facets of this protocol. In fact, signaling information exchanged among nodes and ancillary data structures used in support of routing operations are strictly related to the requirements of the considered data flows.

The *Multipoint-to-Point* (*MP2P*) is the dominant traffic in many LLN applications. It is usually routed towards nodes with some application relevance, such as the LLN gateway to the larger Internet or to the core of private IP networks. In general, these destinations are the DODAG roots and they act mainly as data collection points for distributed monitoring applications. Contrariwise, *Point-to-Multipoint* (*P2MP*) data streams can be used for actuation purposes, by means of messages sent from DODAG roots to destination nodes. Finally, *Point-to-Point* (*P2P*) traffic is necessary to allow communications between two devices belonging to the same LLN, e.g., a sensor and an actuator. In this case, a packet will flow from the source towards the common ancestor of those two communicating devices; then, downward towards the destination. In some constrained scenarios (e.g., when nodes cannot store routes), the common ancestor may be a DODAG root, needing some form of IP source routing from the DODAG root towards the destination.

As an obvious consequence, RPL has to discover both upward routes (i.e., from nodes to DODAG roots) in order to enable MP2P and P2P flows, and downward routes (i.e., from DODAG roots to nodes) to support P2MP and P2P traffic.

A. RPL topology formation

The simplest RPL topology is made by a single DODAG with just one root, that is, it is built as a DAG containing only one DODAG. For example, this is the case of a WSN monitoring a small size area.

A more complex scenario encompassed by RPL is composed of multiple uncoordinated DODAGs with independent roots. This topological choice is a way to split the LLN in several partitions depending on the needs of the application context. Note that, in this kind of scenario, all DODAGs belong to the same DAG and each node can join only one of those DODAGs.

A more sophisticated and flexible configuration could contain a single DODAG with a virtual root that coordinates several LLN root nodes . The main advantage in this case, with respect to the previous one, is the absence of limitations on the parent set selection, given that all nodes belong to the same virtual DODAG , although a stronger coordination is needed among the root nodes.

Depending on the application requirements, it is also possible to combine the three examples presented so far in more complex topology.

Moreover, RPL provides a way to manage several monitoring applications on the same network. Multiple instances of RPL may run concurrently on the network devices and each instance has specific routing optimization objectives, such as the minimization of delay and energy consumption. Such an instance is strictly linked to a single DAG (composed by one or more DODAGs). To this aim, a *RPLInstanceID* is also employed to identify one of the possible RPL instances running on the same network.

The formation of all these possible kinds of topologies relies on the RPL information dissemination mechanism, which enables a minimal configuration in the nodes and allows them to operate mostly autonomously. In this sense, a key role is played by DODAG Information Option (DIO) messages, containing information about the *Rank*, the Objective Function, the IDs, and so on. They are multicasted (periodically and link-locally) by each node to create the DODAG, thus establishing paths towards the roots.

In detail, according to RPL specifications, in order to implement network formation and management operations, all nodes execute several operations: they send and receive DIOs; they compute their own *Rank*, based on the information included in the received DIOs; they join a DODAG and select a set of possible parents in that DODAG among all nodes in the neighborhood; they select the preferred parent among the possible ones. More specifically, when a node joins the LLN, it waits for a DIO message in order to discover possible parents. Optionally, a new node can multicast a DODAG Informational Solicitation (DIS) to ask for a DIO.

A node receiving a DIO message uses its information to join a new DODAG, or to maintain an existing one, according to the Objective Functions and the *Ranks* of their neighbors. It can also detect possible routing loops. To reach these goals, the following function is used to compare node *Ranks* and to create a DODAG:

$$DAG_{Rank(N)} = \lfloor Rank(N)/MinHopRankIncrease \rfloor \tag{2}$$

where N is the node identifier, $Rank(N)$ is the *Rank* of node N, $\lfloor x \rfloor$ is the greatest integer less than or equal to x; and *MinHopRankIncrease* is the implementation-dependent minimum hop rank increase value, representing the minimum difference between the *Rank* of a node and the *Ranks* of its possible parents.

Upon a DIO message is received from a neighbor, a node setups its own *Rank* to a value that is a function of both the neighbor *Rank* and the cost to reach the DODAG root through it. The considered node lets that its set of possible parents contain only that neighbor, if one of the following conditions is true: (i) the node *Rank* was not already setup; (ii) the old value, A, of the node *Rank* and the computed one, B, verify the relation $DAG_{Rank(A)} > DAG_{Rank(B)}$. Instead, if $DAG_{Rank(A)} = DAG_{Rank(B)}$, the neighbor is added to the set of possible parents. In other cases, the DIO is not further considered. Finally, each node can select its preferred parent within its set of candidate parents based on several possible rules, such as Objective Function, path cost, *Rank*, and so on.

On the other hand, a node advertises its presence, the affiliation with a DODAG, the routing cost, and the related metrics by sending DIO messages to nodes in its neighborhood, only if it has already computed its own *Rank*. An exception is allowed to the DODAG root, which is configured to get its own *Rank* equal to the value *MinHopRankIncrease*, and to send it with the *DODAGID*, the routing cost, and the related metrics into DIO messages. In this way, a DODAG is constructed in a widening-wave fashion, starting from the DODAG root.

A DODAG root can issue a global repair operation by creating a newer version of the DODAG. Nodes in the new DODAG can choose a new *Rank* regardless their positions within the old DODAG Version. RPL also supports mechanisms which may be used for local repair within the same DODAG version, e.g., upon loop detection.

It is worth to remark that these procedures are useful to establish upward routes only. Therefore, in presence of P2MP and P2P traffics, an additional mechanism is required to create downward paths. To this end, RPL uses

Destination Advertisement Object (DAO) messages to back-propagate routing information from leaf nodes to the roots. They are triggered by the reception of a DIO message, or in global and local repair operations. After receiving a DAO message, each node forwards it to its parent at the expiration of a timer, which is implementation-dependent [31].

To avoid redundancies and to control the signaling overhead, the *trickle* algorithm [55] triggers, for each node, a new DIO message only when the overall amount of control packets already sent in the neighborhood of that node is small enough. With this algorithm, the time is split in an endless sequence of intervals with size I. A node can transmit a new DIO message at a random instant t in the second half of each interval if, since the beginning of that interval, the number of signaling messages (which have been heard) is smaller than a given threshold, δ_{RC}, i.e., the *redundancy* constant. The size I is not fixed, but it is varied along the time in the range $[I_m, 2^M \cdot I_m]$, where I_m is the minimum size of I. In particular, note that, starting from its minimum value, I is doubled at the end of each interval, up to a maximum number, M, of times. When an *inconsistent* state is detected (e.g., there is the detection of a loop or the join to a new DODAG version), the trickle timer is reset, i.e., I is set again equal to I_m.

B. RPL control messages

In the previous section we have explained how the DIS, DIO, and DAO objects are used to create and maintain a LLN topology. Herein, we explain how they are exchanged in multi-hop network. Specifically, they are encapsulated as RPL control messages into ICMPv6 packets, according to [58]. The structure of a message is reported in Fig. 5.

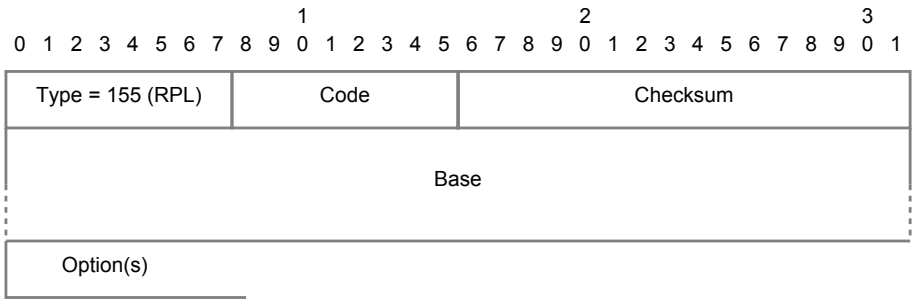

Fig. 5. RPL encapsulation in an ICMPv6 packet.

The requested Type code for RPL messages is 155 (such a value has to be confirmed by the Internet Assigned Numbers Authority, IANA). The Code field indicates which kind of control message, i.e., DIS, DIO, or DAO, is present after the Checksum . The Base field is the RPL message header and it contains only the basic information related to the functionalities of the carried object. Instead, the Option(s) field is the body of such messages and, depending of the needs, it may be composed of any combination of optional functionalities (padding, metric containers, route information, DODAG configuration, RPL target, and so on).

A DIS message (code field $= 0x00$) is composed by a 2 bytes long header, which is initialized to zero by the sender and it is ignored by the receiver. In general, it includes in its body a solicited information option, which expresses possible preferences on the RPL instance, the DODAG or the version that nodes want to join.

A DIO message (code field $= 0x01$) provides a richer set of information with respect to DIS packets. In fact, besides containing *RPLInstanceID*, *DODAGID*, *DODAGVersionNumber*, and *Rank*, it also includes several flags. The more significant ones specify: (i) if a repair operation is ongoing; (ii) if RPL can support downward routes and, in case, if nodes in the DODAG should maintain them in their routing tables; (iii) a preference index for the DODAG, called the *DODAGpreference*.

Finally, DAO messages (code field $= 0x02$) transport the *RPLInstanceID* in their header, in order to construct downward routes in the related DAG. Among all the features provisioned by RPL, its local instances can be declared in support of a future on-demand routing solution. In that case, the *DODAGID* should be also included in the DAO header. Other notable fields are: (i) a settable flag for asking an acknowledgment (i.e., the RPL DAO-ACK message, with code field $= 0x03$) for the correct reception of a DAO message; (ii) a DAOSequence field to distinguish between consecutive DAO message receptions.

Besides, each RPL message has a secure variant providing integrity and protection as well as optional confidentiality and delay features .

C. Metrics and constraints

Possible metrics and constraints (which can be fruitfully exploited for timely adapting the topology to changing network conditions) are [59]: node energy, hop count, link throughput, latency, link reliability, and link color. In particular, with the term "colors" RPL refers to specific properties of links, so that the link color is used to include or exclude such links from the paths.

This richness of information, from on side, makes RPL highly adaptable to different operating conditions. On the other hand, it is necessary to keep under control the adaptation rate of routing metrics in order to avoid path instabilities, which would severely impair LLN performance and scalability.

All the available metrics can be advertised in DIO or DAO messages, using the DAG Metric Container object (see Fig. 6). In particular, a Metric Container Option can be carried into the option field of a RPL control message and it can include several Metric Container Objects.

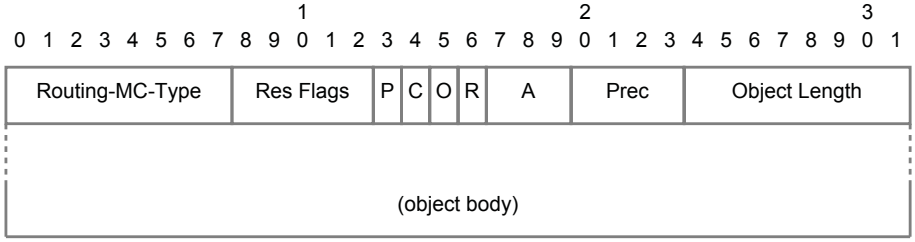

Fig. 6. Metric Container Object format.

The Routing Metric/Constraint Type field identifies each Routing Metric/Constraint object, while the length field reports the length of the object body (expressed in bytes). The flag *C* is used to discriminate whether the content is a routing constraint or a metric. If the content is a routing constraint, the *O* flag expresses if it is optional. Otherwise, if the content is a routing metric *P* indicates if all nodes on the path record metrics; *R*, instead, specifies if metric are recorded or aggregated along the path; *A* discriminates among additive, multiplicative, maximizing or minimizing metrics. Finally, the *Prec* field sets the precedence of the object relative to other ones in the container.

D. The Objective Function

In RPL, the Objective Function translates key metrics and constraints into a *Rank*, which models the node distance from a DODAG root, in order to optimize the network topology in a very flexible way. Furthermore, the Objective Function allows the selection of a DODAG to join and the identification of a number of peers in that DODAG as parents. Generally speaking, the parent selection at a node could be triggered in response to several events, such as the reception of a DIO message, a timer elapse, all DODAG parents become unavailable, or a trigger indicating that the state of a candidate neighbor has changed. After the Objective Function has scanned all the interfaces at a node to check whether they can be eligible for establishing a link in the topology, all candidate neighbors are examined to evaluate if they can act as RPL router. These preliminary operations are useful to exclude all those links and candidate nodes that do not match basic Objective Function compatibility rules, e.g., related to security issues, performance, an so on. Then, the node scans the list of the candidate parents that passed the preliminary tests. The *Rank* that would result from having each of them as parent is evaluated. The preferred parent is elected as the one that can grant the smallest *Rank*, provided that this *Rank* is smaller than the one currently held by the node itself. Obviously, these operations can be iterated when more than one parent has to be selected.

The Objective Function is identified by an Objective Code Point (OCP) within the DIO Configuration option, indicating the method that should be used to construct the DODAG. The Objective Functions proposed by IETF are described below.

1) Objective Function 0: It is identified by an OCP equal to zero [60], and it requires only the information in the RPL DIO header, such as the *Rank* and the *DODAGPreference*. A node *Rank* is obtained by adding a normalized scalar, *RankIncrease*, to the *Rank* of a selected preferred parent. The *RankIncrease* value is a multiple of $0x100$, so that *Rank* values can be stored in one octet. Given that in the RPL main specification [31] there is neither default Objective Function, nor default metric container, it might happen that two implementations, following different guidelines for a specific problem or environment, will not support a common Objective Function which they could interoperate with. Therefore, Objective Function 0 is designed as a common denominator among all the generic implementations. It ignores metric containers and it leaves to implementation the responsibility to compute how link properties are transformed into a *RankIncrease*.

2) Minimum Rank Objective Function with Hysteresis: It is designed to find the paths with the smallest path cost while preventing excessive churn in the network [61]. It is identified by an OCP equal to 1. A node switches

to the minimum cost path, *NewPathCost*, only if the following inequality is verified:

$$NewPathCost < CurrentPathCost - PARENT\_SWITCH\_THRESHOLD \quad (3)$$

where *CurrentPathCost* is the path cost of the current path, and *PARENT_SWITCH_THRESHOLD* is a given threshold, implementing hysteresis.

This Objective Function may be used with any additive metric listed in [59] as long the routing objective is to minimize the given routing metric. Besides, it employs a DODAG parent set with only one node. This node is automatically chosen as the preferred parent. As a consequence, any candidate neighbor may become the preferred parent.

VI. Transport Layer and Above – IETF CoAP

A LLN using IPv6 provides world-wide Internet integration given that nodes can be addressed and information can be routed through the network without requiring specialized NAT techniques at the gateways. However, for complete Internet compatibility, some features which are not addressed by the network layer are required. On one hand, it would be desirable that a node manage multiple non-interfering requests. This issue can be dealt by specialized application code running at each node or by multiplexing network layer through the use of the concept of port. Besides, end to end reliability cannot be guaranteed by network layer as its task need to be performed over the network routing structure. Usually both functionalities are addressed by upper-network layers such as transport and application in the TCP/IP network stack.

On the other hand, application layer protocols provide application independent semantics that facilitate content representation and inter-operability between different applications. Protocols, such as HTTP [15], enable applications to inter-operate in a client/server content/resource centric fashion. Internet of Things aims to enable LLNs to interoperate with existing applications without the need of specialized application oriented code, thus requiring LLNs to talk application layer protocols. As classical networks do not need to operate with energy restrictions, content tagging and metadata are not optimized for minimum packet overhead; this limits their integration in LLN applications. Thus, a set of techniques to compress application layer protocol metadata have been proposed without compromising application inter-operability.

A. Transport over LLNs

The Transport layer is responsible of providing end-to-end reliability over IP based networks. TCP [17] provides traffic control and congestion control through Automatic Repeat-Request (ARQ) techniques [62]. It sustains the traffic on the Internet and provides reliability thanks to the control overhead introduced for each transmitted packet. Reliable transport protocols over LLNs are being studied but the amount of information for traffic control and reliability are expensive in terms of number of transmitted packets and end to end packet confirmation which directly maps to energy consumption. Dunkels et al [63] presented a lightweight TCP implementation based on the use of caching that reduces the amount of control packets. Other approaches focus on the use of Selective Repeat

variant [64] which selectively acknowledge received packets with the caveat that acknowledgments are end-to-end. So that, they are required to cross the entire network and this is not energy efficient. Due to the expensive energy requirements imposed by end to end reliability and the lack of a clear proposal for reliable transport in LLNs, the use of User Datagram Protocol (UDP) [65] and retransmission control mechanisms at application layer are demonstrating a good tradeoff between energy cost and reliability.

UDP is a datagram oriented protocol that provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism and overhead. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed (i.e. UDP neither provides guarantees to the upper layer protocols for message delivery nor retains state of UDP messages once sent). Like TCP, UDP provides application multiplexing through the concept of port.

As stated in previous sections, 6LoWPAN removes a number of fields in the IPv6 and UDP headers because they take well-known values, or because they can be inferred from fields in the IEEE 802.15.4 header.

B. Application Layer

In the application scenarios addressed by LLNs, we will find a range of devices involved with very different capabilities, from full servers to constrained devices consisting of 8-bit or 16-bit microcontrollers with wireless network interfaces such as IEEE 802.15.4 radios. In that kind of scenarios, there is the need to restrict the use of different protocols to a certain subset that can interoperate across device types, inclusively at application layer. Experience with protocol gateways translating between protocols providing similar services, tells us that such gateways can cause nasty operational problems since protocol semantics are often not 100% translatable in some corner cases. In addition, the use of web services on the Internet applications has become the de-facto standard which draws that application layer interoperability have to be in conformance with a representational state transfer architecture of the web [15].

Due to the restrictions imposed by LLNs, a straightforward implementation of RESTful architectures such as the client/server model defined by HTTP is not possible and an adaptation is required. While REST architectures make assumptions on efficient reliable transport and are not strictly constrained by payload size as expensive fragmentation is dealt at lower layers, a LLN has to carefully take into account several of these features as the services offered by lower layers are considerably more restrictive. 6LoWPAN for example supports the expensive IPv6 packet fragmentation into 127 bytes long packets, but an abuse of that makes the network inoperable. Thus a requirement for application layer is to limit the packet extension.

The IETF Constrained RESTful Environments (CORE) working group [66] has defined the Constrained Application Protocol (CoAP) [32] which easily translates to HTTP for integration with the web, while meeting specialized requirements such as: multicast support, very low overhead, and simplicity for constrained environments. CoAP has been designed as a generic protocol for LLNs taking into account the features of the underlying architecture [67]. The CORE working group, instead of blindly making a compression of HTTP [15], defined a subset of the RESTful specification, making it interoperable with HTTP but also specializing it for so constrained environments.

The summary of the main features addressed by CoAP are [32]:

- Constrained web protocol specialized to M2M requirements.
- Stateless HTTP mapping through the use of proxies or direct mapping of HTTP interfaces to CoAP.
- UDP transport with application layer reliable unicast and best-effort multicast support.
- Asynchronous message exchanges.
- Low header overhead and parsing complexity.
- URI and Content-type support.
- Simple proxy and caching capabilities.
- Optional resource discovery.

Unlike HTTP, CoAP is an asynchronous request/response protocol over a datagram oriented transport such as UDP. The client/server architecture of HTTP is slightly different in CoAP as end-points do not assume a so clear role. This is motivated by the nature of the underlying transport, which is asynchronous (i.e., datagram oriented), and both endpoints acting as clients and servers. The architecture of CoAP is divided in two layers, a *message layer* in charge of reliability and sequencing and a *request/response layer* in charge of mapping requests to responses and their semantics (see Fig. 7).

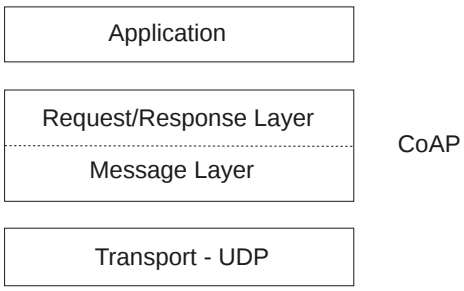

Fig. 7. CoAP architecture.

1) Message layer: The function of the CoAP message layer is to control message exchanges over UDP between two endpoints. Requests and Responses share a common message format. Messages are identified by an ID used to detect duplicates and for reliability. There are four types of messages which are specified in the header.

- *Confirmable:* messages that require a response, which can be piggybacked in an acknowledgement or sent asynchronously in another message if the response takes too time to be computed. Confirmable messages that cannot be processed are replied with a Reset message. Responses to confirmable messages are also confirmable messages that need to be acknowledged.
- *Non-Confirmable:* Messages that do not need to be neither acknowledged nor replyed.
- *Acknowledgement:* Messages that confirm the reception of a confirmable message. They can contain the piggybacked response to the confirmable message.
- *Reset:* In case a confirmable message cannot be processed.

In addition, multicast messages are supported being only possible for Non-Confirmable messages.

2) Request/Response layer: CoAP request and response semantics are carried in CoAP messages, which include either a method code or response code, respectively. Optional (or default) request and response information, such as the URI and payload content-type are carried as CoAP options. A Token Option is used to match responses to requests independently from the underlying messages. As CoAP is implemented over non-reliable transport, CoAP messages may arrive out of order, appear duplicated or be lost without notice. Thus, CoAP needs to implement a reliability mechanism with the following features:

- Simple stop-and-wait retransmission reliability with exponential back-off for confirmable messages.
- Duplicate detection for both confirmable and non-confirmable messages.
- Multicast support.

Reliability works over *Confirmable* messages. Upon reception of such message, receiver must acknowledge it or reject it by sending a *Reset* message. The sender retransmits the *Confirmable* message at exponentially increasing intervals, until it receives an *acknowledgment* (or *Reset* message), or runs out of attempts (controlled by a retransmission counter). *Non-confirmable* messages are never acknowledged nor rejected. In case they cannot be processed, they are ignored.

Request and Responses are mapped to each other thanks to the token embedded into the header. A Request consists of the method that should be applied to the resource, of the identifier of the resource, of a payload and an Internet media type (if any), and of an optional meta-data about the request. A Response is identified by the Code field in the CoAP header. Similar to the HTTP Status Code, the CoAP Response Code indicates the result of the attempt to understand and satisfy the request.

3) CoAP frame: CoAP messages are encoded in a simple binary format. As shown in Fig. 8, a message consists of a fixed-sized CoAP Header followed by options in Type-Length-Value (TLV) format and a payload. The number of options is determined by the header. The payload is made up of the bytes after the options, if any; its length is calculated from the datagram length. The main fields on the frame are the following:

- Version: 2 bits that show the CoAP version number. Set to 1 in the current version.
- Type: 2 bits that indicate the type of message. In particular, we can have the messages: (0) Confirmable, (1) Non-Confirmable, (2) Ack, (3) Reset.
- Option Count: 4 bits, indicate the number of options in the option header.
- Code: 8 bits that indicate if the message is a request or a response. In case of request, indicates the request method (GET, PUT, and so on). In case of a response, the Response code (2.01, 4.03, and so on).
- Message ID: 16 bits field with a unique ID to match *Confirmable* and *Acknowledgements* or *Reset* messages and to detect duplicates.
- Options: Option list in TLV format.
- Payload: The content of the message, usually a resource representation. Its type is defined by the Content-Type Option. Error responses include a human-readable description of the error such as “Bad Gateway”.

4) CoAP basic methods: CoAP offers the methods for a RESTful architecture.

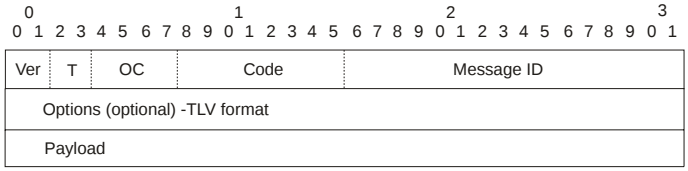

Fig. 8. CoAP frame format.

- **GET**: Idempotent and safe operation that retrieves a representation for the information corresponding to the resource identified by the request URI.
- **POST**: Requests the processing of the representation enclosed in the resource identified by the request URI. Normally it results in a new resource or the target resource being updated. The method is neither safe nor idempotent.
- **PUT**: Requests that the resource identified by the request URI be updated or created with the enclosed representation. The representation format is specified by the media type given in the Content-Type Option. PUT is not safe but idempotent.
- **DELETE**: The method requests that the resource identified by the request URI be deleted.

Responses are identified by Response codes analogous to HTTP Status codes. Due to space limitations only some of them are commented in this section. The full list of codes can be found at [32].

- **Success**: Codes 2.XX represent that the request has been received, understood and accepted. For example, code 2.01 is analogous to HTTP code 201 “Created” but only in response to POST and PUT requests.
- **Client Error**: Codes 4.XX are returned when a client incurred in some error. For example, code 4.04 is the same as HTTP code 404 “Not Found”.
- **Internal Server Error**: Code 5.xx returned when a server is not able to carry out the request. For example 5.02 analogous as HTTP 502 code “Bad Gateway”.

5) Caching and Proxying: The goal of caching in application layer protocols is to reduce the required network bandwidth thanks to the reuse of prior response messages to satisfy a specific current request. In some cases, a cached response can be used without requiring a network request, considering the constraints of LLNs, this extremely benefits the lifetime, latency and network round-trips. Contrarily to HTTP, CoAP responses are defined to be cacheable according to the Response Code in the header definition. For this purpose, a freshness mechanism based on a Max-Age option is used. Responses are tagged by the server (i.e., the end-point answering) with an explicit age that will maintain the response cached until its expiration. By default the Max-Age option is 60 s.

CoAP distinguishes between requests to an origin server and a request made through a proxy. A proxy is a CoAP end-point that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to serve the response using a cache in order to reduce response time and network bandwidth or energy consumption. CoAP requests to a proxies are made as *Confirmable* or *Non-Confirmable* requests to the proxy end-point, but setting the Proxy-Uri Option and splitting the request URI

to the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options. As in the architecture of Internet caching and proxying are fundamental to alleviate the traffic in LLNs.

6) CoAP URIs: CoAP URIs are very similar to HTTP URIs. The "coap" URI scheme has been identified for CoAP resources and for providing the means to locate the resources. As in RESTful architectures, resources are organized hierarchically and governed by a potential origin server listening for requests on a given port. The structure of the URI follows the model defined in [68]:

$$coap-URI \;=\; \text{“}coap:\text{”}\;\; \text{“}//\text{”}\; host\; [\; \text{“}:\text{”}\;\; port\;]\; path-abempty\; [\; \text{“}?\text{”}\; query\;]$$

The host can be provided either as an IP address, or a name which should be resolved using a resolution service such as DNS. The port is the UDP port where the end-point is listening and the path defines the resource in that host. Finally, as in RESTful resources, the query in the form of "key=value" pairs enables the parametrization of the resource.

As indicated in [32] application designers are encouraged to make use of short, but descriptive URIs. Since the environments addressed by CoAP are usually constrained for bandwidth and energy, the trade-off between these two qualities should lean towards the shortness, without ignoring descriptiveness.

Resource discovery is related to how CoAP end-points are addressed and it is defined in [69]. Basically, the function of the discovery mechanism is to provide URIs ("links") for the resources offered, complemented by information describing the relationship between the resource description and each resource as well as other attributes.

7) Application layer protocols mapping: As CoAP implements a subset of the HTTP functionalities, there is a direct mapping between them. Besides CoAP can be easily mapped to the Session Initiation Protocol (SIP) [70] and the Extensible Messaging and Presence Protocol (XMPP) [71] as they have some similarity with HTTP.

- CoAP-HTTP Mapping enables CoAP clients to access resources on HTTP servers through an intermediary. This is initiated by including the Proxy-Uri Option with an "http" URI in a CoAP request to a CoAP-HTTP proxy, or by sending a CoAP request to a reverse proxy that maps CoAP to HTTP. The mapping is straightforward, requiring the translation of the HTTP Status codes to the Response Codes in CoAP.
- HTTP-CoAP Mapping enables HTTP clients to access resources on CoAP servers through an intermediary. This is initiated by specifying a "coap" URI in the Request-Line of an HTTP request to an HTTPCoAP proxy, or by sending an HTTP request to a reverse proxy that maps HTTP to CoAP. The mapping requires a filtering of those codes, options, and methods that are not supported by CoAP.

VII. Concluding Remarks

Frost & Sullivan confirm that the industrial segment is the fastest growing market for sensors, corroborating the notion of an Internet of Important Things. The world market over all industry segments is estimated at some 36 billion Euros, showing the importance of that market but also underlying the need of getting its design right. It has to be noticed that *Important* stands for not only economical terms, but also for the implications of enriching

Internet with the information of millions (or even billion in the coming future) of real world critical, unattended sensing and actuating objects. The latter are omnipresent and important *per se* [2] as they inter-actuate physically not only with other smart objects, but also with human beings.

The aim of this paper has hence been to outline a technically viable communication architecture able to support the stringent energy and connectivity needs of the emerging IoT. To this end, we have seconded the community's view in the need of a standardized architecture which replaces proprietary approaches by means of a transparent end-to-end architecture.

From a PHY perspective, we found that the current IEEE 802.15.4-2006 PHY layer(s) suffice in terms of energy efficiency. In the end, it is the actual hardware implementation which dictates the exact current draws and thus the energy needed to transmit a given information bit. Current hardware implementations by e.g., Dust Networks are already very close to the limit of possibilities when it comes to short-range and medium-rate communication. Given that a large amount of IoT applications however will require only a few bits to be send, it may be advisable to commence looking into a standardized PHY layer which allows ultra low rate transmissions over very narrow frequency bands, with the obvious advantage of enormous link budgets and thus significantly enhanced ranges.

From a MAC perspective, we found that the current IEEE 802.15.4-2006 MAC layer(s) do not suffice which essentially triggered the existence of the IEEE 802.15.4e working group. We have presented in great details the MAC protocol of this new family which is tailored to industrial multihop/mesh applications under extreme fading and interference conditions. Channel hopping, albeit not novel in the wireless communications community, has been successfully applied to this embedded MAC; in addition to a rigid slot structure allowing for enormous energy savings since transmitter and receiver only wake up when truly needed. Some open issues pertain also to this family in that no optimal centralized or decentralized scheduling protocols have been put forward; nor is it entirely clear which approach is to be preferred.

From a networking perspective, the introduction of the IETF 6LoWPAN protocol family has been instrumental in connecting the low power radios to the Internet and the work of IETF ROLL allowed suitable routing protocols to achieve universal connectivity. Indeed, both WGs enabled IPv6 connectivity which is a great asset in guaranteeing global reachability, true scalability, reliable security and, since IP-enabled networks have been successfully engineered and deployed for decades now, the same engineering skills maintaining and troubleshooting these type of emerging networks; this is an enormous advantage over proprietary solutions. Various open issues pertain to the networking layer, however; examples are a suitable choice of the embodiment of the objective function, inclusion of trust, ability to run over any link layer protocol and not only those which have regular beacons, etc.

From an application perspective, the introduction of the IETF CoAP protocol family has been instrumental in ensuring that application layers and applications themselves do not need to be re-engineered to run over low-power embedded networks. Indeed, the current approach allows for the same design principles as currently used in general Internet application designs, thus acting as a true enabler for the IoT.

[2] by themselves

The introduced stack, in one form or another, is currently being evangelized by various industrial alliances. Of importance to the development of an IoT are arguably the Zigbee and IPSO alliances. Whilst the Zigbee alliance has traditionally been proponent of the IEEE 802.15.4-2006 PHY/MAC embodiments and an alliance-proprietary protocol stack (referred to as profile) on top, it is lately adopting above IETF recommendations at networking layers. The IPSO alliances, on the other hand, is very actively pushing for IPv6 enabled solutions to be adopted across the industry with the ultimate aim to facilitate true connectivity.

This paper has shed light onto some of the most recent and emergent design paradigms related to the communications stack of a viable Internet of Things. A lot of tweaking and optimizing is still ahead of us but we believe that the major bulk of design work is accomplished and that the current stack will make a significant impact in the take-off of the IoT.

ACKNOWLEDGEMENTS

This publication is based in parts on work performed in the framework of the projects VITRO-257245, EXALTED-258512, CALIPSO-288879, OUTSMART-285038 and SWAP-251557, which are partially funded by the European Community. The Authors would like to acknowledge the contributions by the various colleagues from these projects. This work was supported in part also by the project ERMES PON 01_03113/3, funded by the Italian MIUR, the funding of which is gratefully acknowledged. Xavier Vilajosana is funded by the Spanish Ministry of Education under Fullbright-BE grant (INF-2010-0319).

REFERENCES

[1] Auto-ID Labs. Available online: http://www.autoidlabs.org.

[2] European Union. COM(2007) 96. *European Commission Communication on RFID*, Mar. 2007.

[3] Council ot The European Union. *Transport, Telecommunications and Energy*. Available online: http://www.internet-of-things-research.eu/documents.htm, 27 Nov. 2008.

[4] U.S. National Intelligence Council (NIC). *Global Trends 2025: A Transformed World*. NIC, Available online: www.dni.gov/nic/NI-2025-project.html, Nov. 2008.

[5] Y. Huang and G. Li. Descriptive Models for Internet of Things. In *Proc. of Int. Conf. on Intelligent Control anf Information Processing (ICICIP)*, Dalian, China, Aug. 2010.

[6] INFSO D.4 Networked Enterprise RFID INFSO G.2 Micro Nanosystems in Co-operationwith the Working Group RFID of the ETP EPOSS. Internet of Things in 2020, Roadmap for the Future, Version 1.1. Technical report, 27 May 2008.

[7] L. Atzoria, A. Ierab, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, Oct. 2010.

[8] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi. From Todays's INTRAnet of Things to a Future INTERnet of Things: A Wireless- and Mobility-Related View. *IEEE Wireless Communications*, 17(6):44 – 51, Dec. 2010.

[9] L. Coetzee and J. Eksteen. The Internet of Things - Promise for the Future? An Introduction. In *Proc. of IST-Africa Conf.*, Gaborone, Botswana, May 2011.

[10] E. Fleisch. *What is the Internet of Things? - An Economic Perspective*. Auto-ID Labs, 2010.

[11] European Research Cluster on Internet of Things (IERC). *Internet of Things - Pan European Research and Innovation Vision*. IERC, Available online: http://www.internet-of-things-research.eu/documents.htm, Oct. 2011.

[12] L. Mainetti, L. Patrono, and A. Vilei. Evolution of Wireless Sensor Networks towards the Internet of Things: A survey. In *Proc. of 19th Int. Conf. on Software, Telecommunications and Computer Networks (SoftCOM)*, Split, Dubrovnik, Sept. 2011.

[13] J.P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[14] O. Hersent, D. Boswarthick, and O. Elloumi. *The Internet of Things: Key Applications and Protocols*. Wiley, 2012.

[15] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter andP. Leach, and T. Berners-Lee. HyperText Transfer Protocol – HTTP/1.1. RFC 2616, IETF, June 1999.

[16] DARPA Internet Program. Internet Protocol. RFC791, DARPA, Sept. 1981.

[17] J. Postel. Transmission Control Protocol. RFC 793, IETF, Sep. 1981.

[18] G. Lawton. Machine-to-Machine Technology Gears up for growth. *Computer*, 37(9):12 – 15, 2004.

[19] ETSI TS 102 689 v1.1.1. Machine-to-Machine communications (M2M): M2M service requirements, Aug. 2010.

[20] D. Lance, L.William, and S. Jonathan. Channel-Specific Wireless Sensor Network Path Data. In *16th IEEE Int. Conf. on Computer Communications and Networks (ICCCN)*, Turtle Bay Resort, Honolulu, Hawaii, USA, Aug. 2007.

[21] R.T. Lacoss. *Distributed Sensor Networks*. MIT/LL (Massachusetts Institute of Technology/Lincoln Laboratory), 1983.

[22] MIT Lincoln Laboratory. Available online: http://www.ll.mit.edu/.

[23] J.M. Kahn, H. Katz, and K.S.J. Pister. Next century challenges: Mobile Networking for Smart Dust. In *Proc. of ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom)*, Seattle, WA, Aug. 1999.

[24] K.S.J. Pister, J.M. Kahn, and B.E. Boser. Smart Dust: Wireless Networks of Millimeter-Scale Sensor Nodes. *Highligh Article in Electronics Research Laboratory - Research Summary*, 1999.

[25] IEEE std. 802.15.4. *Part. 15.4: Wireless Medium Access Control (MAC) and PhysicalLayer (PHY) Specifications for Low-Rate Wireless PersonalArea Networks (LR-WPANs)*. IEEE Standard for Information Technology, Sep. 2006.

[26] ZigBee Alliance. Available online: www.zigbee.org.

[27] Dust Networks. Available online: www.dustnetworks.com.

[28] K. Pister and L. Doherty. TSMP: Time synchronized mesh prootocol. In *Proc. of Int. Symp. Distributed Sensor Networks, DSN*, Florida, USA, Nov. 2008.

[29] HART Communication Protocol and Fundation. Available online: http://www.hartcomm2.org.

[30] N. Kushalnagar, G. Montenegro, and C. Schumacher. *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview,Assumptions, Problem Statement, and Goals*. IETF, RFC 4919, Aug. 2007.

[31] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and JP. Vasseur. *RPL: IPv6 Routing Protocol for Low power and Lossy Networks (work in progress)*. IETF ROLL working group, Mar. 2011.

[32] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. *Constrained Application Protocol (CoAP)*. IETF CoRE Working Group, Feb. 2011.

[33] IEEE draft std. 802.15.4e. *Part. 15.4: Wireless Medium Access Control (MAC) and PhysicalLayer (PHY) Specifications for Low-Rate Wireless PersonalArea Networks (LR-WPANs) Amendament 1: Add MAC enhancementsfor industrial applications and CWPAN*. IEEE Standard for Information Technology, Mar. 2010.

[34] International Society of Automation. *ISA-100.11a Wireless systems for industrial automation: Process controland related applications*. ISA, 2009.

[35] A. Tinka, T. Watteyne, and K. Pister. A Decentralized Scheduling Algorithm for Time Synchronized ChannelHopping. *Ad Hoc Networks*, 49(4):201–216, 2010.

[36] S. Berson S. Herzog R. Braden, L. Zhang and S. Jamin. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC 2205, Sep. 1997.

[37] IETF. MPLS-TP Internet Drafts and RFCs. Available online: http://wiki.tools.ietf.org/misc/mpls-tp/.

[38] T. Watteyne, A. Mehta, and K. Pister. Reliability Through Frequency Diversity: Why Channel HoppingMakes Sense. In *Proc. of Performance Evaluation of Wireless Ad Hoc, Sensor, and UbiquitousNetworks (PE-WASUN)*, Tenerife, Canary Islands, Spain, Oct. 2009.

[39] B. Kerkez, T. Watteyne, and M. Magliocco. Fleasibility analysis of controller design for adaptive channel hopping. In *Proc. of ICST Int. Workshop on Performance Methodologies and Tools for WirelessSensor Networks, WSNPERF*, Pisa, Italy, Oct. 2009.

[40] Z. Shelby and C. Bormann. *6LoWPAN: The Wireless Embedded Internet*. Wiley Series on Communications Networking & Distributed Systems. John Wiley & Sons, 2010.

[41] J.W. Hui and D.E. Culler. Extending IP to Low-Power, Wireless Personal Area Networks. *IEEE Internet Computing*, 12(4):37 – 45, July-Aug. 2008.

[42] J. Hui, D. Culler, and S. Chakrabarti. 6LoWPAN: Incorporating IEEE 802.15.4 into the IP architecture. Internet Protocol for Smart Object (IPSO) Alliance, White Paper, Apr. 2009.

[43] J.W. Hui and D.E. Culler. IPv6 in Low-Power Wireless Networks, Invitated Paper. *Proceedings of the IEEE*, 98(11):1865 – 1878, Nov. 2010.

[44] K. Jeonggil, A.Terzis, S.Dawson-Haggerty, D.E. Culler, J.W. Hui, and P. Levis. Connecting Low-power and Lossy Networks to the Internet. *IEEE Communications Magazine*, 49(4):96 – 101, April 2011.

[45] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. IETF, RFC 4944, Sep. 2007.

[46] J. Hui and P. Thubert. *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. IETF, RFC 6282, Sept. 2011.

[47] M. Crawford. *Transmission of IPv6 Packets over Ethernet Networks*. IETF, RFC 2464, Dec. 1998.

[48] P. Karn, C. Bormann, G. Fairhurst, D. Grossman, R. Ludwig andJ. Mahdavi, G. Montenegro, J. Touch, and L. Wood. *Advice for Internet Subnetwork Designers*. IETF, RFC 3819, July 2004.

[49] J.P. Vasseur, N. Agarwal, J. Hui, Z. Shelby, P. Bertrand, and C. Chauvenet. RPL: The IP routing protocol designed for low power and lossy netwroks. Internet Protocol for Smart Object (IPSO) Alliance, White Paper, Apr. 2011.

[50] J. Martocci. *Building Automation Routing Requirements in Low-Power and Lossy Networks*. IETF, RFC 5867, June 2010.

[51] A. Brandt, J. Buron, and G. Porcu. *Home Automation Routing Requirements in Low-Power and Lossy Networks*. IETF, RFC 5826, Apr. 2010.

[52] K. Pister and P. Thubert. *Industrial Routing Requirements in Low-Power and Lossy Networks*. IETF, RFC 5673, Oct. 2009.

[53] M. Dohler, T. Watteyne, T. Winter, and D. Barthel. *Routing Requirements for Urban Low-Power and Lossy Networks*. IETF, RFC 5548, May 2009.

[54] N. Bressan, L. Bazzaco, N. Bui, P. Casari, L. Vangelista, and M. Zorzi. The deployment of a smart monitoring system using wireless sensor and actuator networks. In *Proc. First IEEE Int Smart Grid Communications (SmartGridComm) Conf*, pages 49–54, 2010.

[55] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. *The Trickle Algorithm*. IETF, RFC 6206, Mar. 2011.

[56] J. Tripathi, J. C. de Oliveira, and J. P. Vasseur. A performance evaluation study of rpl: Routing protocol for low power and lossy networks. In *Proc. 44th Annual Conf. Information Sciences and Systems (CISS)*, pages 1–6, 2010.

[57] E. Baccelli, M. Philipp, and M. Goyal. The p2p-rpl routing protocol for ipv6 sensor networks: Testbed experiments. In *Proc. 19th Int Software, Telecommunications and Computer Networks (SoftCOM) Conf*, pages 1–6, 2011.

[58] A. Conta, S. Deering, and M. Gupta. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version6 (IPv6) Specification*. IETF, RFC 4443, Mar. 2006.

[59] JP. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthe. *Routing Metrics used for Path Calculation in Low Power and Lossy Networks (work in progress)*. IETF ROLL working group, Mar. 2011.

[60] P. Thubert. *RPL Objective Function Zero (work in progress)*. IETF ROLL working group, Sept. 2011.

[61] O. Gnawali and P. Levis. *The Minimum Rank with Hysteresis Objective Function (work in progress)*. IETF ROLL working group, Mar. 2012.

[62] G. Fairhurst and L. Wood. Advice to link designers on link automatic repeat request (ARQ). RFC 3366, IETF, Aug. 2002.

[63] A. Dunkels, T. Voigt, and J. Alonso. Making TCP/IP Viable for Wireless Sensor Networks. In *Proc. of the First European Workshop on Wireless Sensor Networks (EWSN 2004),work-in-progress session*, Berlin, Germany, Jan. 2004.

[64] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. RFC 2018, IETF, Oct. 1996.

[65] J. Postel. User datagram protocol. RFC 768, IETF, Aug. 1980.

[66] Constrained RESTful Environments (core). IETF Working Group. Available online: http://www.ietf.org/dyn/wg/charter/core-charter.html.

[67] C. Bormann, A.P. Castellani, and Z. Shelby. CoAP: An Application Protocol for Billions of Tiny Iinternet Nodes. *IEEE Internet Computing*, 16(2):62 – 67, 2012.

[68] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (URI): generic syntax. RFC 3986, IETF, Jan. 2005.

[69] Z. Shelby. Core link format, draft-ietf-core-link-format. Internet draft, IETF, June 2011.

[70] J. Rosenberg, H Schulzrinne, G. Camarillo, A. Johnston, J. Petersonand R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, IETF, Jun. 2002.

[71] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): core. RFC 3920, IETF, Oct. 2004.