IOT-A
Internet of Things Architecture

FP7 Contract no.: 257521

D3.6 - IoT Protocol Suite definition

Deliverable D3.6

Deliverable ID: IOT-A_WP3_D3.6
Deliverable Title: D3.6 - IoT Protocol Suite definition
Deliverable Editor: Michele Rossi (CFR)
Authors: Alexandru Serbanati (CATTID), Yosra Ben Saied (CEA), Alexis Olivereau (CEA), Alessandro Bassi (HEU), Angelo Castellani (CFR), Cristiano Tapparello (CFR), Konstantinos Koutsopoulos (CSE), Pierpaolo Giacomin (HEU), Michele Rossi (CFR)

PROPRIETARY RIGHTS STATEMENT
This document contains information, which is proprietary to the IOT-A Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written permission of the IOT-A consortium.
Contents

1 Introduction and Scope 5

2 Definitions

2.1 Communication entities terminology 6
2.2 Network Types 6
2.3 Network constraints 7
2.4 Terminal Types 8
2.5 Terminal constraints 9
2.6 Gateway 11

3 Networking Scenarios

3.1 Scenario S1 (Unicast) 12
3.2 Scenario S2 (Unicast) 13
3.3 Scenario S3 (Unicast) 13
3.4 Scenario S4 (Multicast) 13
3.5 Scenario S5 (Unicast and Multicast) 14
3.6 Scenario S6 (Anycast) 14
3.7 Required Functionalities 15

4 IoT Protocol Architecture

4.1 Protocol Architecture: General Description 15
4.2 Description of the Architectural Components 16
4.3 Networking Procedures 19
4.3.1 Unconstrained Network Domain (UCN) 20
4.3.2 Constrained Network Domain (CN) 21
4.4 Security Procedures 25
4.4.1 Cryptography enablement 26
4.4.2 Intrusion Detection Systems 27
4.4.3 Trust Management Systems 28
4.4.4 Privacy Preserving Procedures 30

5 Security Association for Unicast Communication

5.1 Constraints and requirements 31
5.2 Security Considerations 32
5.2.1 A DTLS-based Lightweight Security Association 33
5.3 Formal Protocol Description 35
5.4 DTLS Implementation 36
5.4.1 DTLS module 37
5.4.2 Cryptography module 37
5.4.3 Keyed-Hash Message Authentication Code (HMAC) 37

Date: June 11, 2013
Version: Final

Internet of Things Architecture © 1 / 103
5.4.4 Counter with Cipher Block Chaining Message Authentication Code
(Counter with CBC-MAC or CCM) .............................................. 39
5.4.5 Rijndael ................................................................. 42
5.4.6 Secure Hash Algorithm (SHA)-2 ........................................ 42

5.5 Testing ................................................................. 43
5.5.1 Basic scenario ......................................................... 43
5.5.2 Integration of 6LoWPAN ............................................. 45
5.5.3 Complete scenario with PSK .......................................... 45

5.6 Conclusion ........................................................... 46

6 Collaborative Security Protocols ........................................ 46
6.1 Review of Key Establishment Schemes ................................. 46
6.1.1 Algorithmic protocols, communication protocols .................. 47
6.1.2 Classification of key establishment protocols ....................... 47
6.1.2.1 Key transport vs. key agreement ................................ 47
6.1.2.2 Cryptographic primitives ......................................... 49
6.1.2.3 Authentication method ........................................... 50
6.1.2.4 Preliminary classification of key-establishment protocols ... 51
6.1.3 IoT key establishment: generic design decisions ................... 52
6.1.3.1 Security ............................................................. 52
6.1.3.2 Pervasiveness ....................................................... 52
6.1.3.3 Efficiency ........................................................... 53
6.1.3.4 Adoptability ........................................................ 53
6.1.3.5 Final classification of key-establishment protocols .......... 54
6.1.4 Applicability of existing key exchange schemes for IoT scenarios and related work .......................... 54

6.2 Proposed Solution for collaborative Keying ......................... 56
6.2.1 Considered network model .......................................... 57
6.2.2 Assumptions .......................................................... 57
6.2.3 Preparation of involved entities ..................................... 58
6.2.4 Key exchange description .......................................... 58
6.2.4.1 Collaborative key transport .................................... 58
6.2.5 Collaborative key agreement ........................................ 60

6.3 Performance Analysis ................................................. 63
6.3.1 Considered examples for key exchange ............................. 64
6.3.1.1 Example of key transport protocol ............................. 64
6.3.1.2 Example of key agreement protocol ........................... 65
6.3.2 Computational cost .................................................. 66
6.3.3 Communication cost ................................................ 67
6.3.4 Total energy cost .................................................... 71

6.4 Conclusion ........................................................... 71
7 Back Pressure Congestion Control for IoT

7.1 Distributed Congestion Control for Distributed IoT Networks .... 72
7.2 Back Pressure Congestion Control for 6LoWPAN ................. 74
7.3 Node Model .................................................. 74
    7.3.1 Link ................................................. 74
    7.3.2 Network ............................................. 74
    7.3.3 Transport ........................................... 74
    7.3.4 Application ......................................... 75
7.4 Layer 3 Device Types ........................................... 75
    7.4.1 Static ................................................ 75
    7.4.2 IdealBP .............................................. 75
    7.4.3 Griping .............................................. 76
    7.4.4 Deaf ................................................ 77
    7.4.5 Fuse ................................................ 77
7.5 Unidirectional Upstream Data Traffic ............................... 78
    7.5.1 System Parameters .................................... 78
    7.5.2 Performance Metrics ................................ 79
    7.5.3 Results for Upstream Unidirectional Traffic ............... 80
    7.5.4 Impact of varying the transmission rate $\lambda_{tx}$ ........ 80
    7.5.5 Impact of varying $N_{\text{retx}}$ ........................ 82
    7.5.6 Impact of varying $Q_{\text{thr}}$ ........................ 84
7.6 Back Pressure Congestion Control for CoAP ......................... 85
    7.6.1 L3 Devices for Bidirectional Back Pressure ............... 86
    7.6.2 System Parameters .................................... 87
    7.6.3 Performance Metrics ................................ 87
    7.6.4 Results for Bi-directional CoAP Traffic .................. 88
    7.6.5 Impact of varying $\lambda_{tx}$ ........................ 89
    7.6.6 Impact of varying $N_{\text{retx}}$ ........................ 90
    7.6.7 Impact of varying $Q_{\text{max}}$ ........................ 91
7.7 Results for Asymmetric Topologies and Cross-Traffic ............. 92
    7.7.1 Asymmetric node deployment ................................ 92
    7.7.2 Layer 3 devices for cross-traffic ........................ 93
    7.7.3 Results for CoAP cross-traffic .......................... 95
7.8 Conclusions on Congestion Control Algorithms ...................... 95

8 Conclusions 96
List of Figures

1. Communication scenario S1. .................................................. 13
2. Communication scenario S3. .................................................. 13
3. Communication scenario S4. .................................................. 13
4. Communication scenario S6. .................................................. 14
5. Protocol Architecture. ......................................................... 16
6. Networking Scenario. ......................................................... 19
7. Diagram of the Networking Architecture. .................................. 20
8. Usage diagram for the EXI processor and pre-processor. ............... 22
9. Authentication for Network Access Control. .............................. 27
10. Key Establishment. ............................................................ 28
11. Intrusion Detection. .......................................................... 29
12. Trust Management Systems. ............................................... 30
13. Lightweight DTLS security association. .................................... 34
14. Handshake protocol. ......................................................... 35
15. Diagram of the HMAC generation. ......................................... 38
16. Diagram of the CTR encryption (decryption) mode. ........................ 39
17. Diagram of the CBC-MAC mode of operation. .......................... 40
18. CCM forward transformation. .............................................. 41
19. CCM inverse transformation. .............................................. 41
20. Diffie-Hellman key agreement. ........................................... 48
21. Comparison of HIP-BEX and HIP-DEX. ................................. 56
22. Needham-Schroeder public key protocol. ................................ 64
23. Modified Needham-Schroeder protocol. ................................ 65
24. Distributed HIP Exchange. ................................................ 67
25. Network topology ............................................................ 78
26. Reception rate $\lambda_{rx}$ and multihop delay $D$ vs. the offered traffic load $\lambda_{tx}$. 79
27. Rejection rate $R$ vs. the offered traffic load $\lambda_{tx}$. .................. 82
28. Transmission overhead $N_{tx}$ vs. the offered traffic load $\lambda_{tx}$. .......... 83
29. Loss probability $P_{loss}$ vs. the number of retransmissions $N_{retx}$. .......... 83
30. Transmission overhead $N_{tx}$ vs. the queue threshold $Q_{thr}$. .......... 84
31. Received response rate $\lambda_{tx}$, and round-trip time $D$ vs. the offered request load $\lambda_{tx}$. ........................................... 87
32. Transmission overhead $N_{tx}$ vs. the offered request load $\lambda_{tx}$. .......... 88
33. Loss probability $P_{loss}$ vs. the number of retransmissions $N_{retx}$. .......... 90
34. Received response rate $\lambda_{tx}$ vs. the maximum queue length $Q_{max}$. .......... 91
35. Loss probability $P_{loss}$ vs. the maximum queue length $Q_{max}$. .......... 92
36. Average $P_{loss}$ for subnets S1, S2 and S3 for bidirectional CoAP traffic. .......... 93
37. Average $P_{loss}$ in the presence of bidirectional CoAP cross-traffic for $\lambda_{tx}^1 = 15$ re- q/client/s. ........................................... 95
1 Introduction and Scope

In this deliverable we illustrate the design of the final IOT-A protocol suite. At first, in Section 2 we define useful terms, classify network and terminal types (especially in terms of their capabilities / resources) and discuss revenant constraints. In Section 3 we define relevant network scenarios that entail the communication of IoT devices within constrained domains as well as through the Internet thanks to the exploitation of suitable IoT Gateways.

Protocol architecture: Section 4 contains the description of the proposed protocol architecture, including the general diagrams pertaining to the involved actors and the foreseen interactions among them, the description of each protocol component, along with the related networking and security procedures. Note that our protocol design takes into account the current developments that are being carried out by the scientific community as well as by relevant standardization committees such as IETF. Hence, we are fully aligned with these and we include them in our design. In addition to this, we provide novel features that complement and improve existing IoT technology.

These technological advancements are discussed in the following three sections of the deliverable. These are centered on the setup of end-to-end and secure communication channels through lightweight and possibly cooperative means and the design of novel transport layer protocols, specifically designed for constrained domains. In detail:

1. End-to-end secure channel establishment (Section 5): we present novel strategies to set up secure end-to-end communication channels through the involvement of a trusted IoT Gateway. These strategies entail the offloading of some computationally heavy security functionalities to the Gateway so as to reduce the computational burden for the constrained IoT devices. Note that this procedure is completely transparent to the servers of computers that access the IoT resource from the standard Internet, i.e., that reside outside the constrained domain. Moreover, our procedures also include the current IETF practice whereby the IoT Gateway is totally transparent and has no active involvement in the security channel establishment. This section contains a full description of the approach along with its implementation details and recommendations for use and future improvements.

2. Cooperative security protocols (Section 6): we present efficient key establishment protocols that are conceived to delegate cryptographic computational load to less resource-constrained nodes through a collaborative scheme. The idea is that highly resource-constrained IoT nodes can obtain assistance from more powerful nodes in their network in order to securely derive a shared secret with a peer. In this section a full design is presented: we first review, classify and evaluate the existing key establishment protocols. Thus, we describe the proposed cooperative key transport and key agreement schemes for IoT nodes. Finally, we the proposed solutions are fully evaluated and compared against existing non-cooperative protocols.

3. Novel congestion control protocols for constrained domains (Section 7): we
address the design of network architectures for the Internet of Things by proposing practical algorithms to augment IETF CoAP/6LoWPAN protocol stacks with congestion control functionalities. Our design is inspired by previous theoretical work on back pressure routing and is targeted toward Web-based architectures featuring bidirectional data flows made up of CoAP request/response pairs. Here, we present three different cross-layer and fully decentralized congestion control schemes and compare them against ideal back pressure and current UDP-based protocol stacks. Hence, we discuss extensive numerical results, which confirm that the proposed congestion control algorithms perform satisfactorily in the selected network scenarios for a wide range of values of their configuration parameters, and are amenable to the implementation onto existing protocol stacks for embedded IoT devices.

## 2 Definitions

In this section we define communication entities, network types, discussing and classifying their inherent resources in terms of memory, computational power, transmission rates and energy availability. Moreover, we discuss a few relevant communication scenarios, some of which will be explored in detail in the remainder of this document.

### 2.1 Communication entities terminology

With the terms resource and user in this document we mean the devices hosting the communication endpoint and, more specifically:

- **Resource (R)** is the device having access to the sensed data and can either be constrained or unconstrained.

- **User (U)** is the device requiring data from a resource and can either be constrained or unconstrained.

### 2.2 Network Types

In this document networks are classified as unconstrained (NTU) or constrained (NTC):

- **NTU**: Unconstrained networks are characterized by high speed communication links (e.g., offering transfer rates in the Mbit/s range or higher) as nowadays wired Internet. Link level transfer latencies are also short and mainly impacted by possible congestion events in the network rather than by the physical transmission technology.

- **NTC**: Constrained networks are characterized by relatively low transfer rates, typically smaller than 1 Mbit/s, as offered by, e.g., IEEE 802.15.4. These networks are also characterized by long latencies and this is due to several factors including: 1) the involved low rate physical layer technology and 2) the power saving policy of the terminals populating these networks, which may imply the periodic power off of their radios for energy efficiency purposes.


2.3 Network constraints

The previous classification enables a fast framing of networks, but at some moment it could be useful to be more precise in order to tackle some specific feature of a constrained network. Bormann et al. in their "Terminology for Constrained Node Networks" use the following definition:

- **Constrained Network**: A network where some of the characteristics pretty much taken for granted for Internet link layers in 2013 are not attainable.

This definition is even more general than our NTC definition, avoiding to list those characteristics. The same document also gives another refinement of the definition, proposing the term **Constrained Node Network**, defined as follows:

- A network whose characteristics are influenced by being composed of a significant portion of constrained nodes.

Another interesting definition while speaking of Constrained Networks is the definition of LLN (Low-power and Lossy Network) given by the ROLL (Routing Over Low-power and Lossy Networks) IETF WG in [1]:

- "Low power and Lossy networks (LLNs) are typically composed of many embedded devices with limited power, memory, and processing resources interconnected by a variety of links, such as IEEE 802.15.4, Low Power WiFi. There is a wide scope of application areas for LLNs, including industrial monitoring, building automation (HVAC, lighting, access control, fire), connected home, healthcare, environmental monitoring, urban sensor networks, energy management, assets tracking and refrigeration."

Unfortunately, this definition is not very helpful in outlining typical constraints of those networks, mainly relying on requirements arising by different vertical applications. Finally, **Challenged Networks**:

- A network that has serious trouble maintaining what an application would today expect of the end-to-end IP model, e.g., by:
  - not being able to offer end-to-end IP connectivity at all;
  - exhibiting serious interruptions in end-to-end IP connectivity;
  - exhibiting delay well beyond the MSL defined by TCP.

Delay Tolerant Networking [2] has being designed for networks falling in the *Challenged Networks* category.

Hence, the constraints describing NTCs more in detail:
• **Small frame sizes.** The frame size is the maximum number of bits that can fit in a single frame. For the current Internet it is generally assumed at least 1400 bits can fit in a single frame, and IPv6 requires a minimal MTU of at least 1280 bits. Some link layers do not allow to fit 1280 bits in a single frame. This constraint is an expression of the network medium, for instance IEEE 802.15.4 supports 127 bits at the physical layer.

• **Low bit rate.** The bit rate is rate at which the sent bits reach the destination. On the current Internet it is absolutely common to have link layers supporting several megabits. Due to a number of reasons, this assumption cannot be held for the IoT. The reasons of low bit rate range from constraints of the device itself (detailed in 2.5), Reduced Data Capacity, RDC, and power consumption, costs, media constraints, to regulatory constraints. Even considering wired networks, ICs supporting 10 Mbit/s ethernet, such as ENC28J60-I/SP, comes at a fraction of the cost of an IC supporting Fast ethernet.

• **High packet loss rate.** The packet loss rate is the rate of packets which do not get at destination. It can be caused by signal degradation over the transmission medium, channel congestion, deliberately by routing routines and by full queues both in intermediate equipment and terminals. In case of stateful protocols, it can cause severe service disruptions (a protocol is stateless if the nodes do not need to keep track of previous interactions to decide what to do during further interactions [3]).

• **High delay.** Delay, intended as one-way end-to-end delay, is the time difference between the instant a packet is received and then instant it is sent. The only kind of expected maximum delay in the current Internet as described in the TCP specification [4], is the Maximum Segment Lifetime and has been chosen arbitrary as 2 minutes, one-way. As a matter of fact, in modern UNIX implementation the `TIME_WAIT` timer is set by default to 60 seconds, causing an expected maximal delay of 30 seconds one-way. High delays can be caused by: the medium itself, intermediate equipment, routing events and by the device both at protocol stack level and for processing at application level.

• **High Packet Delay Variation** (also known as Jitter). PDV is the difference between end-to-end one-way delay of packet belonging to the same flow. High PDV is usually caused by routing.

• **Disconnection or Opportunistic connectivity.** This constraint is typical of Challenged networks. Disconnections can be found in any network, in particular due to faults, but in NTCs disconnections can be part of the normal operational behavior of the network, for two main reasons: device positioning change and duty-cycle (performing full disconnection, not to be confused with RDC).

### 2.4 Terminal Types

As far as terminals are concerned, the following three categories – unconstrained (TT1), constrained (TT2), and tag-type (TT3) – are defined as below:
• **TT1**: Unconstrained terminals have sufficient computational power and energy reserve to implement complex tasks, e.g., carrying out complex security functions, dealing with standard HTTP traffic, XML and supporting the high transmission rates that are typical of NT1 networks. Internet servers, fixed and laptop computers are classified as unconstrained terminals.

• **TT2**: Constrained terminals have: 1) reduced transmission capabilities in terms of transmission rates (usually smaller than 1 Mbit/s), 2) constraints in terms of energy reserve as these terminals are usually battery operated or co-powered through energy scavenging means, 3) constrained in terms of memory storage, typical values are RAM smaller than 10 Kbytes, ROM smaller than 100 Kbytes, 4) constrained in terms of computational capabilities, typically their micro-controllers have clock speeds smaller than 100 MHz.

• **TT3**: Tag-type terminals are typically not able to participate in an end-to-end IP communication. This is justified by their extreme limitations in computing power, memory storage and energy storage. Some typical Near Field Communication (NFC) tag-types are limited to a storage of 48 bytes. RFID tags may be even limited to send an ID without being able to store information (see NFC-Forum: http://www.nfc-forum.org/specs/spec_list/#tagtypes). The typical access to those kinds of tags is implemented via direct communication with a reader device, which is either a constrained or unconstrained device capable of exploiting end-to-end IP-based communications.

### 2.5 Terminal constraints

The terminal classification presented in the previous section allows the fast framing of terminal types. However, it does not allow outlining the actual nature of their constraints, and how they impact the functioning of the terminals themselves. To fill this gap, we identify four major terminal constraints that allows us to generate a more detailed classification:

**POWER**

The first constraint is **power**. We can outline four classes of devices regarding power:

• **P0**: battery-less devices. They have a capacitor which is charged by harvesting power (delivered by a small movement, light, etc.). The stored energy is just sufficient to send or receive a limited number of (nowadays 1 to 3) packets per processor wake-up. It is important to realize that **P0** devices are not limited in terms of the number of packets they can send, but are limited by their transmission rate.

• **P1**: battery-constrained devices. These devices are powered by a battery which limits both the devices' lifetime and the total number of packets they can send.

• **P2**: green devices. Independent of the energy source (external, battery, capacitor, etc.) these devices continuously switch on and off parts of the electronics to reduce the power consumption as function of the external demands on its operation. Moreover they can host some type of energy harvester (vibrational, solar, etc.), which allows...
their battery to be recharged. This, together with suitable energy-aware transmission protocols can ensure their perpetual operation.

- **P3:** power unconstrained devices. There is enough power available to have these processors with their attached equipment running continuously. Examples are devices connected to the power grid.

**CODE SIZE**

The second constraint is **code size.** Here, more than the actual numbers, it is important to understand the implications of having a limit in code size. Constrained code space affects functionalities that can be implemented by the terminal, among them: ability to handle multiple processes, single loop execution, time scheduling, application standard extensions (BACnet, SEP2, etc.), ports and request handling, the actual network stack layers covered IP, UDP, TCP, HTTP, CoAP, remote file system, TFTP, FTP, SNMP, MAC power management (RDC or else), memory management, packet multiplexing, big integer handling (very relevant for asymmetric cryptography), crypto primitives, 6LoWPAN and other functions.

A way to classify terminal according this constraint could be a list of terminal types accommodating a growing list of functionalities, as the following:

- **E0:** single loop controller with MAC (e.g. Ethernet);  
- **E1:** single loop controller with 6LoWPAN and UDP;  
- **E2:** single loop controller with one CoAP port;  
- **E3:** single loop controller with basic CoAP functionality;  
- **E4:** multiprocess controller with CoAP;  
- **E5:** multiprocess controller with full network IP stack.

Another sensible way to classify them is in terms of classes taken from the IETF draft-bormann-lwig-terms-00:

- **E0:** much smaller than 100Kb;  
- **E1:** around 100Kb;  
- **E2:** around 250Kb.

**RAM**

Another constraint is the **dynamic memory** (RAM). Again, we think that the most important thing here is to outline what is affected by the scarcity of such a resource. The amount of dynamic memory affects the following: size of inbound and outbound packet queues, size of packets (hot topic with regard to IEEE 802.15.4 and low-power IEEE 802.11), number of handled clients and size of handled per-client state, number of timers, and more. Again, a way to classify terminal according dynamic memory can take into account a growing list of accommodated functionalities, as the following:
• **R0**: stateless devices. The device cannot store anything. The space in memory is just enough to receive and send a packet.

• **R1**: data memory highly constrained devices. Memory for three packets and one acknowledged request plus cache.

• **R2**: data memory constrained devices. Data memory is in the order of magnitude of 3-5 packets handled and 1-3 cached requests.

• **R3**: data memory is in the order of magnitude to serve tens of packets and 10 cached requests.

For the packet size, we hereby refer to the typical sizes of IEEE 802.15-4, which is a commonly adopted standard for IoTs. Note that IPv6 packets would entail a higher memory utilization and the above classification would have to be adapted accordingly.

Another sensible way of classifying terminals is taken from draft-bormann-lwig-terms-00:

• **R0**: much smaller than 10Kb;

• **R1**: around 10Kb;

• **R2**: around 50Kb.

**COMPUTING POWER**

The last constraint that we consider is the terminal computing power. This constraint depends on multiple factors that vary a lot, like, controller speed (8 Mhz-100 Mhz), instruction sets and extensions, the availability of coprocessor (e.g., AES hardware implementations), byte size (from less than 8 bits for controllers with no network to 32 bits), and so on. The effect of those constraints is multi-faceted: Speed could limit the data rate making it unacceptable; the byte size could limit both the rate and increase the code complexity; specific coprocessors can be used to achieve simpler and more compact code and acceptable rates. Among all the above factors, the processor speed is the one which, besides from being more easily classified, in our opinion it better defines the computing power constraint. So, we use the following classification along the lines of draft-bormann-lwig-terms-00:

• **C0**: less than 8 Mhz;

• **C1**: around 32 Mhz;

• **C2**: around 100 Mhz.

### 2.6 Gateway

*Gateway* in this document refers to the communication entity that has the role of either bridging different networks - generally of different Network Types - or relaying messages between different Terminal Types. These two interworking aspects can be combined in case it is required by the communication scenario. In this context the Gateway is supposed to:
• offer a number of physical interfaces through which different Network Types can be accessed,
• interoperate with a number of communication protocols ranging from layer 2 to layer 4,
• implement an M2M logic in the context of IOT-A architectural model that allows it to integrate a wide range of devices.

The communication scenarios are presented in the next subsections in increasing order of complexity.

3 Networking Scenarios

In the following, we describe the types of communications that we foresee for IoTs:

• **UNI**: Unicast: involves the communication between two terminals.

• **MUL**: Multicast: a single terminal sends data to a group of terminals which may be located within the same or a different network domain.

• **RMU**: Reverse multicast: a group of terminals send data (possibly aggregated at some intermediate point) to a single terminal located within the same or a different network domain.

• **ANY**: Anycast: a single terminal (the sender) sends data to a destination terminal, which is selected based on a set of attributes specified by the sender rather than on its specific identity or network address. As an example, a monitoring server may be interested in the temperature in a given room and, as such, will issue a command to address any of the sensor located within the room (the physical location in this case will be the attribute).

• **BRO**: Broadcast: a single terminal (the sender) sends data to all terminals of a network. This can be performed either at Datalink or Network layer. In some cases it is possible to specify the distance the message should be sent to in terms of logical network topology. This function is essential for nodes that need to discover services over an unknown network that does not implement a Discovery component.

3.1 Scenario S1 (Unicast)

**S1 (UNI)**: a single user (U), placed within an NTU, communicates with a single resource (R) placed within an NTC. The two networks are connected through a gateway node (G). Forwarder nodes may be used within both networks to route the data: routing within the NTU may occur according to standard solution, i.e., IP routing, whereas dedicated solutions are needed for the NTC. A diagram for this scenario is shown in Fig. 1.
3.2 Scenario S2 (Unicast)

S2 (UNI): a single user (U), placed within an NTC, communicates with a single resource (R) placed within an NTU. The two networks are connected through a gateway node (G).

3.3 Scenario S3 (Unicast)

S3 (UNI): a single user (U), placed within an NTC, communicates with a single resource (R) placed within a different NTC. The two networks are connected through an NTU and gateway nodes (G) are used to connect constrained and unconstrained networks. Forwarders are used within the NTU to route the data and may also be used within the two NTCs. A diagram for this scenario is shown in Fig. 2.

3.4 Scenario S4 (Multicast)

Figure 1: Communication scenario S1.

Figure 2: Communication scenario S3.

Figure 3: Communication scenario S4.
S4 (MUL): in this scenario a single user positioned within the unconstrained network communicates through a multicast tree to multiple resources, which are located within the constrained network and belong to a multicast group. The corresponding diagram is shown in Fig. 3. This scenario comprises both the multicast case (MUL) where the user sends commands to multiple resources and the reverse multicast case (RMU) where the resources send data back to the user. This scenario also includes the cases where 1) users are placed within the NTC and resources are placed within the NTU and 2) users and resources are placed within constrained networks as in scenario S3 above.

3.5 Scenario S5 (Unicast and Multicast)

S5 (UNI / MUL): in this communication scenario one (multiple) user(s) communicates with a single (multiple) resource(s) and all terminals are placed within the same network, which can either be constrained or unconstrained. This scenario comprises unicast as well as multicast communication, depending on the number of users and resources involved.

3.6 Scenario S6 (Anycast)

S6 (ANY): in this scenario a user placed within the NTU communicates with a resource placed within the NTC. However, the user does not specify the full address of the resource but rather a set of attributes that the resource should satisfy and an address that identifies the NTC. This scenario can be further specialized as shown in Fig. 4 according to two cases:

- **Case a:** this first case entails the use of a smart (or application level) gateway. In this case, the user (U) first connects with the gateway through its own address, specifying a set of attributes for the resource of interest. After this, in a second step the gateway, which should know the attributes of the resources within its NTC and their current status, contacts the most appropriate resource (R) specifying the address of the original caller (U) in the message. From here on, the user U may connect directly to selected resource R as per scenario S1.

- **Case b:** in this second case the user (U) first connects with a server, specifying a set of attributes of interest for a given NTC. All the resources within the NTC are known...
to the server, which keeps a local copy of their attributes and statuses. Thus, the server sends back to the user a message which specifies the most suitable resource (R) within the NTC, including the corresponding network address. From here on, the user U may connect directly to the resource R as per scenario S1.

### 3.7 Required Functionalities

In each of the above scenarios we foresee end-to-end communication occurring among the involved terminals and this communication entails:

- **Addressing**: the usage of a common (or compatible) addressing scheme at the layer 3 of the networks involved. This functionality is mandatory.

- **Routing**: the use of suitable routing algorithms which will sit on top of the addressing scheme and may differ for the different networks involved. This functionality is mandatory.

- **Transport**: a transport solution to ensure a certain level of end-to-end reliability and congestion control. The transport functionality may be split at the gateway, which will thus act as a proxy. In this case the most appropriate transport algorithm can be implemented inside each network. The drawback of this solution is the higher complexity required at the gateway, which must include layer 4 algorithms. The transport functionality is mandatory.

- **Security**: security features can be enforced at almost all layers of the ISO/OSI protocol stack. However, the hard limits imposed by the hardware of constrained devices in IoT systems call for efficient security mechanisms. With this in mind, in IoTs we recommend the implementation of mechanisms that at least guarantee link-layer security. More specifically, we suggest the implementation of an end-to-end secure channel at network or transport level. Although optional for the establishment of a IoT communication channel, this functionality is crucial, and thus, it will be included in the communication stack developed in WP3.

### 4 IoT Protocol Architecture

In this section we detail the IoT-A protocol stack by first presenting, in Section 4.1, a high-level diagram of the protocol architecture. We subsequently delve into the description of its components in Section 4.2 and of its networking and security protocols in Sections 4.3 and 4.4, respectively.

#### 4.1 Protocol Architecture: General Description

A high-level view of the protocol architecture is shown in Fig. 5. On the right hand side of this figure we show the protocols that are responsible for the connectivity of the IoT device,
which entail the physical layer (PHY), the link layer (Link), the network layer (Network), an ID layer for the proper identification of IoT resources (ID), a transport layer (Transport) and, finally, an application layer, which is referred to in the figure as Machine-to-Machine (M2M). These protocols provide the functionalities that are needed for the IoT devices to connect to an existing network infrastructure, receive and send packets, possibly through multiple-hops (routing functionality), perform some error control at the link level and some combined error and congestion control over the entire path (which is provided by the Transport layer).

On the left hand side of the diagram we have instead the functionalities that allow the IoT devices to securely communicate across the network, exploiting authentication and encryption methods. As shown in Fig. 5, a tight interaction is enforced between communication protocols (right) and security functionalities (left). In fact, often, security aspects have to be taken into account into the design of link layer, routing, transport and M2M mechanisms. In what follows, we first detail the role of each component of our architecture, explaining the rationale behind its design and the way it interrelates with the other components. After that, we will describe the related networking and security functionalities.

4.2 Description of the Architectural Components

With reference to Fig. 5, in this section we discuss the various components of our IoT protocol architecture:

**Machine-to-Machine (M2M)** M2M resides in the lower part of the application layer of IoT devices. It addresses the lack of interoperability of current M2M technology and makes communication between different network elements possible either through the adoption of a common descriptive language or through message translation. M2M is also referred to as service layer. In our design, we envision the presence of a suitable translation proxy in the IoT gateway. This proxy is responsible for adapting the application layer language used...
outside the IoT network to the language used by the IoT devices. A possible option in this sense is HTTP to/from CoAP translation. This issue is treated in greater detail in the IoT-A WP3 deliverable D3.5.

Transport (TRA) layer provides end-to-end performance guarantees between communicating endpoints, especially in terms of in-order-delivery and reliability. A TCP-like transport is optional for IoT devices and depends on the required end-to-end Quality of Service (QoS). Standard TCP implementations are inefficient for use in network comprising wireless, low-power and low-data-rate IoT domains, due to the negative impact that long delays and especially high packet losses have on the performance of TCP implementations. To cope with this, in our design we envision the usage of TCP-split mechanisms. Specifically, standard TCP is used within the wired portion of the network (standard Internet), i.e., between some server and the IoT-Gateway, whereas, specialized Transport solutions are used within the IoT domain, so as to efficiently cope with their peculiarities. Our Transport protocol is presented and evaluated in detail in Section 7 of this deliverable.

Identification (ID) layer is an addition to standard unconstrained protocol stacks. It is designed to carry out the resource identification task within the IoT domain, that is hitherto been performed by IP addresses in their dual role of being both identifiers and network locators. Advantages of this approach include enhanced security, which can be embedded in routing and be coupled with an authentication service based on the node ID. The main drawbacks include the lack of a widely adopted protocol suite in the Internet and an increased protocol stack footprint. Considering its pros and cons, we keep this option open in our design by making it non-mandatory. The Host Identity Protocol (HIP) is a notable technological example for this concept [5].

Network (NET) layer houses the Internet protocol, takes care of node addressing and packet routing. We foresee IPv6 [6] as the most suitable Internet technology. As we shall see shortly, some adaptation is required for the constrained domain (namely, 6LowPAN IP header compression [7]), as compression of long IPv6 headers is beneficial to the transmission of IPv6 datagrams over constrained network domains.

Link layer (Link) is responsible for channel access, thereby determining how devices schedule their transmissions over the physical layer medium and also implements some error control over the link connecting the current IoT node to the next one.

Physical Layer (PHY) deals with modulation/demodulation, channel coding and transmission over the given medium. PHY is technology specific. Our aim is to make systems interoperable by filling the gaps that prevent different M2M technologies from communicating with each other. Therefore, PHY and Link layer issues are outside the scope of our architectural design.

The security blocks and their functions are as follows:
Bootstrapping and Authentication controls the network entry of nodes. Authentication is highly relevant to IoT and is likely to be the first operation carried out by a node when it joins a new network, for instance, after mobility. It is performed with a (generally remote) authentication server using a network access protocol such as the Protocol for Carrying Authentication for Network Access (PANA) [8]. For greater interoperability, the use of the Extensible Authentication Protocol (EAP) [9] is envisioned. Upon successful authentication, higher layer security associations could also be established (such as IKE followed by IPsec [10]) and launched between the newly authenticated endpoint and the access control agent in the associated network.

Static Profile represents the knowledge by an endpoint of its own resources (such as identity, battery, computing power, memory size, etc.) and the security settings it intends to use or needs from the network. The static profile can be read-only (preset by vendor), write-once (set by manufacturer) or rewritable (user enabled). Note that certain security primitives may be computationally prohibitive for IoT objects; a negotiation is thus required before the establishment of a secure channel so that the concerned endpoints can agree upon a cryptographic suite.

Collaborative Actions Management is invoked whenever a node cannot fulfill by itself a task it has to accomplish, for instance, when the task is too computationally intensive for the resource constrained IoT node. It interacts with a trusted entity in the constrained network topology to learn about possible assisting peers. To this respect, in Section 6 we detail a design where a secure channel for IoT objects is established thanks to the cooperation of a number of trusted nodes. Through their cooperation, a master key for the encryption of a communication session is established.

Identity and Key Management block ensures secure interaction between endpoints. It establishes node privacy by choosing a particular identity (or pseudonym) for use in the communication stack. It provides secure communications by agreeing on a key with a peer, for instance, via a dedicated Authenticated Key Exchange (AKE) protocol, which may be bound to the identities in use (as in the HIP case). Our design for the establishment of a secure unicast channel between an Internet server and an IoT node is detailed in Section 5.

Adaptation and Awareness block is responsible for configuring the protocol stack of the IoT node and gathering information on its current status. While interacting with the IP layer, for example, it would push security parameters (keys) to IPsec. It would also push IP addresses (or at least, suffixes) through the Identity and Key Management block. The Awareness part of the module contains knowledge about the status of current node and its capabilities (i.e., its Static Profile).

Group Security Management is responsible for enforcing security at the IP layer when dealing with multicast or broadcast communications.

Routing Security block implements a protocol solution aimed at mitigating classical routing
attacks. This module is likely to have relationships with Bootstrapping and Authentication and Local Trust Manager. These interactions are assumed to take place through the Adaptation and Awareness module. Routing Security can be instantiated through the implementation-dependent parameters of the routing protocol in use, e.g. RPL [11].

**Authorization Management (AuthZ Mgt.)** manages inbound and outbound access to Services, interacting with the existing Authorization infrastructure in order to retrieve certificates for accessing other resources and to verify whether authenticated users are authorized to access own resources when they do not use certificates.

### 4.3 Networking Procedures

In this section, we illustrate the mechanisms that allow the communication between a computer connected to the Internet (e.g., an Internet server) and an IoT node. As a reference networking scenario, we refer to that of Fig. 6, where we have an Internet server (left) that communicates to an IoT resource (right) through an IoT-Gateway. In the left hand side of this figure we have the communication occurring over the unconstrained network domain (termed UCN), that is between the Internet server and the IoT-Gateway. From here, the IoT-Gateway connects the IoT node through the communication over a constrained network domain (CN). Note that, in general, the protocols used over the wired Internet portion of this communication scenario differ from those exploited within the CN. This is necessary due to efficiency considerations and to the inherent limitations of IoT nodes in terms of memory, computational and communication resources. Given this, it is necessary to implement dedicated protocols in order to alleviate the burden, especially in terms of protocol overhead, of the protocols that are normally used within the UCN.

A more precise description of this communication scenario is depicted in Fig. 7, where we illustrate the protocols used within the UCN and the CN. These protocols are analyzed in the following subsections for each network segment.
4.3.1 Unconstrained Network Domain (UCN)

Standard Ethernet and Internet technology are exploited over the UCN to set up the communication channel between the Internet server and the IoT-Gateway. These cover the physical layer (PHY) and the link layer technologies exploited over the UCN. For the IP technology, we advocate the use of IP version 6 [6], given that this technology allows the allocation of a much larger number of addresses with respect to IP version 4. At the transport layer two configurations are possible:

S1) in the first and simplest case the User Datagram Protocol (UDP) is exploited, which means that no flow and error control is enforced over the established end-to-end channel,

S2) a TCP protocol is utilized, meaning that some Reno-like transport is used over the UCN to make sure that the data channel is rate-controlled and that the end-to-end communication is reliable.

A standard HTTP protocol is exploited over the CN right below the application layer and, on top of that, Simple Object Access Protocol (SOAP) [12] is used to exchange structured information at the application layer, relying on XML. Note that the architecture used at the Internet server side is quite standard and does not require any modification in existing Internet server architectures. As depicted in Fig. 7, the communication between the application layers of the IP server and of the IoT node occurs in an end-to-end fashion, however, some “non-conventional” functionalities are required at the IoT-Gateway to smoothly match UCN protocols (and functionalities) with the capabilities of IoT devices. The term non-conventional is used here to stress the fact that these functionalities do not exist in the standard Internet and are to be added to adapt Internet-based communications to the technology that is implemented (and being standardized, mainly by IETF) for the Internet of Things. Given this, it is clear that some additional technology is needed in the IoT-Gateway and also within the CN. This technology is described in greater detail in the next subsection.

---

**Figure 7: Diagram of the Networking Architecture.**
4.3.2 Constrained Network Domain (CN)

The Internet world is now in its mature age: the Internet Protocol (IP) is the most used network protocol along with the Hyper–Text Transmission Protocol (HTTP). However, only recently standardization bodies started to play a decisive role in interconnecting constrained devices with the Internet. Recent research efforts explore the performance and the practical feasibility of a REST-based approach [13] on top of a 6LoWPAN stack [14] in IoTs. In our design we consider these concepts. Web services have to be suitable in complex installations and easily deployable while retaining the flexibility typical of IP-based protocols. Recent research [15–17] has demonstrated the benefits of lightweight Web-based protocols accessing sensors resource data through Uniform Resource Identifiers (URI) and request methods (GET, PUT, POST, DELETE) [13, 18, 19]. However, Web Service-enabled IoTs still need a complete protocol stack definition for their direct integration in the Internet, proving that a Web-based system can smoothly bridge information, objects and new services through IoTs. Next, we detail the protocol stack that we have defined for the IoT CN domain so as to achieve seamless and smooth communication with the protocols used over the Internet network segment (UCN).

**XML/EXI** XML has been acknowledged as the de-facto standard for data representation and exchange but his great flexibility comes at the prize of being very redundant; to alleviate this, many solutions are available: blind compressors, such as gzip, bzip2, DTDPPM [20] treat XML as plain text files; a second group of compressors (e.g.: enhanced XMill [21], XMLPPM [22]) takes advantage of structure information the XML document, which can be given using a separate source or can be obtained from the document itself. An exhaustive survey on XML compression techniques can be found in [23]. To the best of our knowledge, the EXI is one of the first working groups focusing on optimizing XML for constrained devices and has been selected as the technology of choice for IoT devices by W3C [24].

The Efficient XML Interchange (EXI) format is a compact XML representation, currently being standardized by the World Wide Web Consortium (W3C) [24]. It is designed to support high-performance XML applications for resource constrained environments, significantly reducing bandwidth requirements and improving encoding/decoding performance. In light of this, we deem it a suitable solution for resource constrained IoT devices and we include it in our design in place of XML (as far as the CN segment is concerned).

EXI compression exploits information about the document structure to internally generate small tags based upon the current XML schema, the current processing stage and the context. Also, tags data representation is optimized to be as compact as possible. Although an efficient compression can be achieved from the XML schema, the standard defines other operating modes to produce a more-compact representation of the XML file using only partial or no XML schema information. The encoded XML document results in an EXI stream, which represents the document in binary format where every data tag of the document is encoded using an *event code*. Event codes are binary tags that preserve their value only in their assigned position within the EXI stream. Thus EXI implements event-based encoding: for efficient encoding, at any given point of an XML stream a set of grammars are used to understand which event is most likely to occur next. A set of grammars, representing the XML document structure, has to be produced before the actual EXI processing. The
sequence of events describes the sequence of finite-state machines defined using each different grammar as transition function. In an EXI stream every XML element is represented using a specific grammar; each grammar consists of a set of productions, defining the set of possible events in a specific state. EXI assigns an event–code (EC) to each production. The sequence of XML elements codified to ECs forms an EXI stream. When a new element of the EXI stream is parsed, a new grammar associated with the element is stacked upon the preceding one and the control is passed to a new automaton which is in charge of handling the new grammar, until the new element is completed and the control returns to the preceding routine.

![EXI processor and pre-processor diagram]

**Figure 8: Usage diagram for the EXI processor and pre-processor.**

We designed and implemented *libEXI* [25], an implementation of the EXI processor that has been specifically targeted for resource constrained MCUs (e.g., Texas Instrument MSP430). The design required to limit the number of features implemented. *libEXI* is a byte-aligned and schema-informed EXI processor, which encodes EXI streams using a preprocessed grammar-set (defining the XML schema in use) and a pre-processed C data structure set (representing a document compliant to the XML schema). Hence, *libEXI* can translate EXI streams into a structured memory representation which can be stored and process by CPU-constrained devices. Bit-aligned encoding, even if very efficient, showed to be too complex to match our requirements. As shown in Figure 8, our EXI library uses the results of a pre-processing phase: a Ruby pre-processor has to run on the XML schemas before *libEXI* can process EXI streams. This preprocessor extracts from any XML schema the set of grammars required for encoding and decoding EXI streams; in addition, it builds a set of C structures representing the XML document layers. The *libEXI* memory representation built by the pre-processor is an optimized translation of the XML document contents, needed for constrained devices to properly manage EXI streams. The *libEXI* processor uses a grammar stack to encode/decode EXI streams. Grammars contain the list of events as well as the information of which grammar has to be stacked to handle the next part of the EXI stream, and which
production will be the current grammar into when the control returns to it. Any new grammar piled in the grammar stack corresponds to a new execution of the encode/decode function call: in this way, there is a one-to-one mapping between the processor internal stack and the current grammars stack. This library has proven very effective and much more efficient in terms of execution time and number of parallel EXI session that can be supported by an IoT node with respect to competing architectures. Quantitative results can be found in [25].

**HTTP/CoAP.** CoAP [26] is currently being defined within the CoRE working group of the IETF, which aims at providing a REST-based framework for resource-oriented applications optimized for constrained IP networks and devices, by designing a protocol set able to cope with limited packet sizes, low-energy devices and unreliable channels. CoAP is based on the REST architectural style and is designed for easy stateless mapping with HTTP, and for providing Machine-to-Machine (M2M) interaction. HTTP compatibility is obtained by maintaining the same interaction model, using a subset of the HTTP methods.

Nodes supporting CoAP provide flexible services over any IP network, and they also provide a solid communication framework to connect sensor nodes to the Internet. Any HTTP client or server can interoperate with CoAP-Ready endpoints by simply installing a translation proxy between the two devices.\(^1\) This will not be a burden for the proxy, since these translation operations have been designed not to be time and computationally demanding. Also, CoAP features a transaction layer between the application protocol and UDP to provide basic reliability and session matching support.\(^2\)

We designed and developed a TinyOS CoAP implementation using the 6LoWPAN header-compression (HC) library from Harvan and Schoenwaelder (6lowpan [27]) which implements the first version of 6LoWPAN HC (RFC 4944) [28]. CoAPP is a monolithic component providing client and server functionalities; it handles session data regardless of its type (either client or server), thus optimizing its memory usage. The actual implementation of this component can handle up to COAP_MAX_TRANSACTIONS transactions simultaneously, a value that can be chosen arbitrarily at build time by trading between memory occupation and flexibility.

The **CoAPClient** interface provides the CoAPP module with a TinyOS command to send any arbitrary request to a CoAP endpoint, and a TinyOS event to manage the response it gets back. Next, we show the TinyOS code of the interface:

```c
interface CoAPClient {
    command coap_tid_t request (
        coap_absuri_t* absuri,
        coap_method_t method,
        coap_content_t* content,
        bool acked);

    event void response (
        coap_tid_t tid,
        coap_status_t status,
        coap_content_t* content ); }
```

\(^1\)In Fig. 7 these translation functionalities are provided by the IoT-Gateway.

\(^2\)These functionalities are provided by the transport layer in the ISO/OSI stack.
The interface defines different custom data types to provide better readability and high-level operations. When a request command is issued the user must provide i) absuri describing endpoint host, port and URI of the requested resource, ii) method specifying which method is used to access the requested resource, iii) content providing a pointer to the content to be attached to the request, if present, iv) acked to request a response message; the request command provides the user with the coap_tid_t internally assigned to the transaction. A response event is triggered when the related reply is received. This response contains i) a tid field identifying the transaction, ii) a status field containing the status code resulted after processing the request and iii) a pointer to the content piggybacked in the response. The CoAPServer interface provides the CoAP module with server capabilities: external components can use this interface to serve resources using a CoAP server. The CoAPServer and the the CoAPClient are complementary in the sense that commands issued using one interface trigger events managed by the other interface and viceversa.

```c
interface CoAPServer {
    event void request (
        coap_rid_t rid,
        coap_absuri_t* uri,
        coap_method_t method,
        coap_content_t* content,
        bool toack );

    command error_t response (
        coap_rid_t rid,
        coap_status_t status,
        coap_content_t* content ); }
```

In order for a request to be properly processed, the following data is needed: i) a rid value internally assigned to univocally identify the request, ii) the uri of the requested resource, iii) method describing the access method, iv) a pointer to the content, if present, and v) a toack flag to signal if the client requested an ACK. The response command can be used by the serving module together with the following parameters, i) a rid to match the related request, ii) status value resulted from the processing of the request and iii) content pointer to data to be sent in the response.

The CoAPServer interface is characterized within the CoAP module by a port parameter identifying on which port the CoAP service has to be activated in the IoT node. The client/server architecture of the CoAP module allows the implementation of lightweight Web services on constrained IoT nodes. Moreover, it makes it possible to implement M2M interactions, such as publish/subscribe, and to create multiple Web servers and services without burdening a constrained node system.

**Transport.** We note that in case we use option S2, the transport protocol is split at the IoT-Gateway, which means that a first TCP connection, exploiting standard TCP Reno-like protocols is operated over the UCN, terminating it at the IoT-Gateway. From here, a second Transport connection is maintained for the network segment covering the CN. This architectural solution is also known in the literature as “TCP-split”, as the transport layer is actually split onto two independent sessions, one for each network segment. As said above, plain
TCP is not suitable for use within constrained network domains due to their long delays and high error rates. Additionally, the type of traffic patterns occurring over CNs substantially differ from those in the Internet, and TCP was not designed with these new patterns in mind. Overall, TCP-split allow us to retain standard TCP over the Internet UCN network segment, leaving untouched the protocols operating over the standard Internet and optimize the transport protocol used within the constrained network segment (referred to in Fig. 7 as TCP*), so that this transport will effectively cope with the typical IoT constraints in terms of delay, data-rate and traffic patterns. Note that similar solutions have also been proposed for satellite networks due to the long delays and high error rates of satellite channels, see, e.g., [29]. Our design of a transport protocol for CNs is detailed and fully evaluated in Section 7 of this deliverable. When option S1 (i.e., UDP) is used, a split approach is no longer needed at the transport layer and UDP can be plainly used along the entire path, i.e., without the need for protocol splitting or adaptation at the IoT-Gateway.

**6LowPAN.** While full IPv6 addresses can be used within the UCN, these are generally too long to be efficiently conveyed over the CN. For this reason 6LowPAN is implemented [14] so as to compress full-length IPv6 addresses and use their compressed version within the CN. Compression and decompression are performed at the IoT-Gateway, which seats on the edge of the CN and connects all IoT nodes residing within the CN with those in the UCN. 6LowPAN has been standardized by IETF and has now become the de-facto standard for IoT addressing over constrained domains.

### 4.4 Security Procedures

Before reviewing the security procedures that must be defined in the framework of the IoT communications, it is worth trying to refine and clarify this concept. Throughout this section, we consider that a security procedure is a set of actions performed by an entity, aiming at a security-related objective. To that respect, a security procedure can be distinguished from the mere application of a security technique to an entire data flow (e.g., obfuscation or integrity protection): oppositely, security procedures are defined as being time-bound, which does not prevent them from being recurring.

A security procedure aims at preventing a malicious attacker from carrying out harmful actions against a wide range of assets including users, data, devices or relationships between logical entities. We believe that a categorization of security procedures according to the attacker profile (internal vs. external attacker) and the mitigation techniques (proactive vs. reactive) leads to a more rational classification of security procedures than the reliance on the traditional integrity, confidentiality and availability criteria. Instead, focusing on the attacker profile on the one hand, and on the mitigation families on the other hand, leads to the identification of three main security procedures families, as depicted in Table 1.

Having defined this nomenclature, our first recommendation would be not to use it dogmatically. Permeability does occur between adjacent cells. Intrusion detection often leads to intrusion prevention. Cryptographic means for mitigating insider attacks do exist. And some honeypots can be designed to attract and lure both insider and outsider attackers.

Another category of attackers can be defined if one considers those that are interested in
Table 1: Classification of security procedures.

<table>
<thead>
<tr>
<th>Outsider Attacker</th>
<th>Insider Attacker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prevention</td>
<td>Cryptography</td>
</tr>
<tr>
<td>Reaction</td>
<td>Intrusion Detection Systems</td>
</tr>
<tr>
<td></td>
<td>Trust Management Systems</td>
</tr>
</tbody>
</table>

gaining information about supposedly private information, such as the location of a user’s carrying an IoT device. The privacy attacker can be either an insider or an outsider. He would, for example, take advantage of a mobile node regularly sending messages using the same source identifier in order to reconstruct the mobility pattern of that node and deduct its owner and/or the activity this owner has carried out. Only preventive security measures can be taken against privacy attackers.

This leads us to classify the security procedures under four main categories: 1) Cryptography enablement procedures, which set up cryptographic security systems aimed at deterring outsider attacks; 2) Intrusion Detection Systems (IDSs), which react to outsider attacks; 3) Trust Management Systems (TMSs), which assess the behaviors of otherwise authorized insiders; and finally 4) privacy preserving procedures, which enforce privacy against malicious entities.

4.4.1 Cryptography enablement

Cryptography enablement corresponds to the security procedures that aim at providing two or more nodes with the cryptographic material they need to establish cryptographically secure (encrypted and or integrity-protected) communications with one another. These security procedures take the form of Authenticated Key Exchange (AKE) protocols, wherein two or more nodes are required to make the proof of their identities, and derive or otherwise obtain key(s) bound to these identities. Eventually, these protocols are often concluded by meeting the "explicit key confirmation" property, by which nodes prove to one another that they successfully obtained the key(s), whereas the protocol design guarantees that only legitimate (properly identified) nodes would have been in the position of doing so.

Two security procedures providing cryptographic enablement are especially used in both the legacy Internet and in the Internet of Things.

- **Authentication for network access control** is the procedure by which a node joining a network gets authenticated and receives in return cryptographic material that it is to use with a network access control enforcer. This latter is an edge network node that controls the data flowing through the network, making sure that it originates at an authorized node.

- **Key establishment** pertains to the establishment of a session key between two nodes that did not previously get involved in secure transactions with one another, and therefore need to bootstrap a symmetric cryptography algorithm between them, in order to set up confidentiality and/or integrity protection services.
Generic models of authentication for network access control and key establishment are respectively depicted in the following Fig. 9 and Fig. 10.

![Diagram of Authentication for Network Access Control](image)

**Figure 9: Authentication for Network Access Control.**

### 4.4.2 Intrusion Detection Systems

The objective of an Intrusion Detection System (IDS) is to react on some events, identified as malicious. An IDS relies therefore on a signature database, which allows it to categorize events as characterizing an attack pattern. Upon attack detection, various sub-procedures may be triggered, including node reconfiguration or issuance of an attack report to neighboring nodes or, more often, to a central server. A generic Intrusion Detection procedure is depicted in Fig. 11.
In the field of the Internet of Things, intrusion detection systems can raise issues with respect to the amount of resources they require from the monitoring entities, both in terms of energy (the mere monitoring during a long period of time may quickly deplete energy resources) or memory (an efficient intrusion detection system requires a large attack signature database, which would likely be impossible to maintain for a highly-constrained node). These aspects are approached, in part, in [30].

4.4.3 Trust Management Systems

Trust management systems refer to systems able to track the trustworthiness of service providers within a collaborative topology, in order to identify benevolent, selfish and malicious nodes within that topology. The corresponding set of security procedures relates to the operations of:

1. assessing a service quality, basing either on node’s own interactions with a service provider or the observation of other nodes’ interactions with their respective service providers;
Act in accordance with the identification result. Action may take the form of self-configuration or report(s) sent to server node and/or local peers.

2. **generating a service evaluation report**, containing at least a description and assessment of the provided service, and a set of parameters characterizing the context in which this service was provided;

3. **processing this report internally** (in which case the report is called a first-hand report) and/or transmitting this report to peers (in which case the report is called a second-hand report) [31].

Trust management systems are not necessarily related to elaborate (applicative) services but are also used for securing the use of collaborative networking services, such as routing. The importance, in the field of the IoT, of associating context information to service assessment reports is especially emphasized in [32].

A generic trust management system is depicted in Fig. 12.
4.4.4 Privacy Preserving Procedures

This last group of security procedures pertains to the preservation of private data for which confidentiality by itself is not sufficient. For example, the mere traffic ‘to’ and ‘from’ identifiers (IP addresses) might leak sensitive information about the fact that a given user, unequivocally associated to the source IP address, is accessing a recognizable service provided by the destination IP address node. For this reason, a privacy security procedure may consist in managing anonymity for the node, by ensuring that random identifiers are used in place of long-term identifiable ones. Likewise, unlinkability of successive packets sent by a single user moving through different locations might be required. In turn, this would require that the anonymity solution be regularly re-instantiated. Better results could be achieved if this re-instantiation happens synchronously between multiple neighboring IoT nodes - this objective may be answered by a dedicated, synchronous, security privacy management procedure. Nevertheless, this can only be seen as a privacy procedure example: privacy schemes are generally too variable to be categorized under a generic procedure model.

5 Security Association for Unicast Communication

In this section of the deliverable we discuss security procedures for constrained IoT devices. We start with the description of a general security architecture along with its basic procedures, discussing how its elements interact with the constrained communication
stack and exploring pros and cons of popular security approaches at various layers of the ISO/OSI model. Thus, we discuss a practical example for the establishment of end-to-end secure channels between constrained and unconstrained devices. The proposed method is lightweight and allows the protection of IoT devices through strong encryption and authentication means, so that constrained devices can benefit from the same security functionalities that are typical of unconstrained implementations, without however having to execute computationally intensive operations. To make this possible, we advocate the usage of trusted unconstrained nodes for the offload of computationally intensive tasks. According to our design, no modifications are required in the protocol stacks of unconstrained nodes.

5.1 Constraints and requirements

Data networks, especially wireless, are prone to a large number of attacks such as eavesdropping, spoofing, denial of service and so on. Legacy Internet systems mitigate these attacks by relying on network layer, transport layer or application layer encryption of the underlying data. Though some of these solutions are applicable to the IoT domain, the inherently limited processing and communication capabilities of IoT devices prevent the use of full-fledged security suites.

The setup of secure end-to-end channels is possible nowadays within the unconstrained network (UCN) domain through a number of mature technologies such as IPsec [10], SSL/TLS [33] or DTLS [34], which however cannot be directly leveraged by constrained network (CN) nodes due to memory space and processing power constraints. To solve these problems, in this section we discuss a suitable security architecture for IoT with the following objectives:

1. security suites currently employed within UCNs shall continue to be used with no modifications on the UCN side;

2. originating security handshakes/procedures are handled differently within the CN so that constrained nodes can handle their complexity, thus being able to establish end-to-end secure channels,

3. unconstrained nodes shall not notice any deviation from their standard procedures.

The solution discussed here is based on the offloading of computationally intensive tasks to a trusted and unconstrained node; this node is then responsible for the calculation of the master session key on behalf of the constrained IoT devices under its jurisdiction. IoT Gateways (GW), placed on the edge between UCN and CN as shown in Fig. 6, have the role of adapting the communication between these two domains and could as well be used for this purpose. In fact, their role usually involves the adaptation between different protocol-layer implementations, which entails physical (PHY) and link (LL) layers but could also encompass all layers up to and including the application layer (M2M). The fact that GWs are generally UCN devices of type TT1 means that they can also be used for scaling down the functionalities (including security) from the UCN to the CN domain and also for managing security settings in peripheral (i.e. CN) networks. In order to maintain the end-to-end approach, GWs
have to be invisible from the viewpoint of the communicating endpoints. Despite end-to-end security, lower layers may continue using heterogeneous security features across network sub-domains or for point-to-point communication.

5.2 Security Considerations

We now consider the network scenario as shown in Fig. 6, where an unconstrained node wants to communicate via HTTP/CoAP with an IoT constrained node. As per communication security, CoAP does not itself provide protocol primitives for authentication or data encryption. The idea is that, wherever required, they can be provided by a secure communication protocol such as IPsec [10] (network layer) or DTLS [34] (transport layer) or object security (within the payload). IPsec works at the network layer and permits the establishment of end-to-end authenticated and encrypted channels between endpoints. Datagram Transport Layer Security (DTLS) operates at the transport layer providing analogous security features. It builds on Secure Sockets Layer (SSL) technology but assumes UDP as the transport protocol. Endpoints that require authorization for certain operations are expected to include one of these two types of security.

IPsec vs DTLS: Each technology has its own pros and cons. Technically, both IPsec and SSL are mature technologies providing all we need in terms of security features, although they reside in different layers of the ISO/OSI stack. Pros and cons of these technologies lie rather in the limits of existing implementations and in their usage models. For IPsec, we note that existing implementations are hardly compatible with each other and often require some manual configuration; in fact, most IPsec solutions for setting up Virtual Private Networks (VPN) require third-party hardware and/or software. Moreover, in order to access an IPsec VPN, a given endpoint must have an IPsec client application installed. This is both an asset and a drawback. The former resides in the fact that a client machine, in order to gain access to a given VPN, is required not only to have a compatible IPsec client installed, but also to have it properly configured (and this entails a further level of security). The drawbacks are that the maintenance of the client software (with valid licenses) on all required client machines can be expensive and the installation can be quite complex and may even require some human intervention. Conversely, SSL technology is nowadays implemented in nearly all Web browsers and has reached a good maturity level; its open source implementations also exist (e.g. OpenSSL, http://www.openssl.org/) and they interoperate satisfactorily. This means that almost every computer in the world is already equipped with the necessary software to connect to an SSL VPN. Another upside of SSL VPNs is that they allow more precise access control. In fact, operation at the transport layer (SSL/TLS or DTLS) allows VPN tunnels to be allocated to specific applications rather than to a specific machine (or often to an entire LAN). So, users on SSL VPN connections can only access the applications that they are configured to access rather than an entire network. Probably, the major drawback of SSL technology is that they only work for Web-based applications and as such they do not always provide native access to all network resources and SSL VPNs are not usable for file sharing or backup operations. This use is not impossible but requires the addition of SSL support onto the application(s) of interest, which somewhat increases its(their) complexity. It is still unclear as to which technology is the best for IoT, although recent devel-
opments from IETF seem to encourage the adoption of DTLS. For a technical comparison of IPsec and SSL/DTLS see, e.g., [35]

In the next subsection, we propose a lightweight methodology to establish an end-to-end secure channel, requiring minimal involvement of constrained IoT devices, while keeping all protocol operations unchanged within the UCN. This is achieved through the delegation of some of the steps associated with channel establishment to a trusted Gateway (GW). Our proposal considers a DTLS security association and can be seen as a particular instance of the security architecture of Section 4.3. Note, however, that a totally similar procedure would also work for IPsec.

5.2.1 A DTLS-based Lightweight Security Association

Here, we discuss an example for the establishment of an end-to-end secure communication channel between a Constrained Device (CD of type, e.g., TT2) and an Unconstrained Device (UD) with the following assumptions:

1. The constrained node uses 6LoWPAN for addressing and CoAP as the application layer protocol.
2. The unconstrained node uses IPv6 for addressing and HTTP as the application layer protocol.
3. The constrained node is already authenticated with the Gateway (GW).
4. There exists a security policy allowing secure communications (authentication and confidentiality) within the constrained network domain (and in particular between GW and the CDs).
5. The gateway is a trusted entity.

Our proposal makes it possible for an unconstrained node to set up a secure connection based on the DTLS protocol [34] with an IoT device, while moving the master session key generation and authentication processes from the IoT node to the trusted gateway. DTLS, which provides data encryption and authentication, allows the setup of an end-to-end secure connection between two peers by encrypting the datagrams. The cryptographic keys are generated and exchanged according to the Elliptic Curve Diffie Hellman key exchange scheme [36]. With the aid of these mechanisms, the logic for key generation and authentication is moved from the IoT node to the corresponding GW, thus relieving the IoT device from the computational burden associated with the generation of cryptographic data. Fig. 13 illustrates the relevant steps involved in the procedure. Finally we note that, by additionally requiring that this security policy is available at the link layer, most of the computation still required at the IoT device can be offloaded to the hardware (most commercial radios in fact already provide some capabilities in this sense).

The UD sends a service request to the CD (message A in Fig. 13), including the desired security level. This request is routed through the Internet and the GW. The GW intercepts the request and acts as if it were the desired Constrained Device (CD, see right hand side in
Fig. 13). Specifically, it responds on behalf of CD (taking into account its Static Profile, see Section 4.2), by sending a message B which includes the security policy supported by the constrained device (which should be a valid subset of that indicated by UD through message A). Note that the security policy supported by CD, which are part of the CD Static Profile, can be made available to the Gateway in a secure manner throm the security policy and channels already available in the constrained network domain.

Upon receiving message B, UD generates the initial keying material for the establishment of a DTLS session and sends it along with a further message C. Thus, once this message C is received at the GW, the latter generates the keying material (for UD) on behalf of the CD, including it into a further message D that is sent from the GW to the unconstrained device UD. Thus, GW sends the master key to the constrained device exploiting a secure channel (that needs to be established beforehand). Upon receiving message D, UD obtains its own master key, while the constrained terminal CD has its master key dispatched in a secure way by the gateway after receiving message C. Once the key exchange phase is over, the authentication credentials are exchanged between the unconstrained node and the gateway. Messages E and F are used to authenticate the two peers (this is performed over an encrypted channel, using the shared secret key that both endpoints obtain from messages A,B,C and D). If GW accepts UD’s authentication credentials, an OK message is sent to the CD. Upon the receipt of this OK message, UD and CD can communicate with each other by exploiting the channel secured by DTLS.

During the communication phase (DATA), in case UD and CD respectively use HTTP and CoAP, the gateway needs to perform CoAP/HTTP translations. Due to this, the gateway
must be able to decrypt the packets exchanged by the communicating endpoints. Hence, the assumption that the gateway is trusted and that it knows the master keys for the active connections does not appear to be excessively bold in this case. When instead CoAP is used by both endpoints the Gateway does not necessarily have to decrypt the exchanged DATA packets; from this point on, GW becomes fully transparent and the secure flow can be redirected to any other Gateway without affecting security and connectivity. Note that the translation from IPv6 to 6LoWPAN does not require the gateway to decrypt the messages flowing through it, as shown in [37]. This latter case is particularly interesting as it allows the exploitation of a trusted Gateway in the initial setup phase, by effectively establishing an end-to-end channel, for which no state is maintained at the edge IoT Gateway. Note that the above procedure entails the offloading of some of the computational burden, namely the generation of the master key, from the IoT device on to a suitable (trusted) Gateway node. The latter is thus required to act on behalf of the constrained devices under its jurisdiction, impersonating them during the initial DTLS handshakes. A totally similar procedure is executed in the opposite direction, i.e., when CD is the initiator.

5.3 Formal Protocol Description

![Handshake protocol diagram](image)

Figure 14: Handshake protocol.
The setup of the lightweight security association of the previous section can be combined with the basic handshake protocol required to establish a DTLS communication. The corresponding DTLS handshake procedure with key exchange is presented in Fig. 14. The computationally intensive tasks required by the handshake protocols are the ones related to the secret key generation, that can be obtained using different algorithms. In our solution, we limit ourself to the Elliptic curve Diffie-Hellman (ECDHE) exchange procedure [36]. When this ciphersuite is used, the ServerKeyExchange and ClientKeyExchange messages also include the Diffie-Hellman parameters [38]. Thus, with our security association, the gateway intercepts these messages and generates the secret key on behalf of the CD, at which point, it will then communicate the results. By doing so, the most intensive computations are delegated to the gateway, while the DTLS association is completely transparent to the unconstrained node (note that the latter can either act as a client or a server).

For a detailed description of the DTLS protocol, see [39]. More information on the key exchange algorithms used in DTLS can be found in [33] and [40].

5.4 DTLS Implementation

The implementation of the secure communication protocol of the previous sections has been done using the TinyOS operating system for constrained devices [41]. TinyOS permits the software development through different modules with ad hoc functionalities. The interoperability among them is obtained through interfaces that can provide commands and require the definition of events. Moreover, we based our implementation on tinydtls [42], which provides a simple datagram server with DTLS support for embedded systems. Since tinydtls is developed in accordance with the IETF RFCs [34], thus being compatible with other implementations for unconstrained devices, our proposed solution will extend the DTLS support to a broader range of devices.

The software is contains the following modules:

1. DTLS;
2. Cryptography;
3. Keyed-Hash Message Authentication Code (HMAC);
4. Counter with Cipher Block Chaining Message Authentication Code) (Counter with CBC-MAC or CCM);
5. Rijandael (that provides Advanced Encryption Standard, AES);
6. SHA-2.

Clearly, DTLS is the main module as it represents the entry point of our implementation and contains all the logic required to handle a secure communications like, e.g., the state machine used to establish a DTLS session, the handshake protocol definition and the structure of the different messages used by the protocol. The cryptography module is used to handle
all the encryption/decryption operations. HMAC defines the algorithm used for the message authentication, CCM provides the actual implementations of the encryption and decryption functions, while Rijndael and SHA-2 provides the implementations of the Advanced Encryption Standard (AES) and a set of cryptographic hash functions, respectively. In the next sections we briefly describe each of the aforementioned modules separately.

5.4.1 DTLS module

The Datagram Transport Layer Security (DTLS) module represents the core of our implementation and integrates all the functionalities to establish a secure communication. DTLS provides communication security for datagram protocols, it is based on the Transport Layer Security (TLS) protocol of which it is intended to provide similar security guarantees. This datagram-compatible version of the protocol is specifically designed to be similar to TLS with the minimal amount of changes needed to fix problems created by the reordering or the loss of packets. Thus, the datagram semantics of the underlying transport are preserved by the DTLS protocol so that the application will not suffer from the delays associated with stream protocols.

We notice that this is the only module that needs to be included in a TinyOS application in order to incorporate the security functionalities described in this section. It provides the commands to initialize a new context, connect to a host and functions to handle a received datagram and send data as a DTLS message.

In order to start a DTLS communication some handshake messages need to be exchanged between the client and the server. As mentioned in Section 5.3, the way in which the constrained node handles the handshake of Fig. 14 has been adapted in order to delegate the computation of the secret key to the gateway.

5.4.2 Cryptography module

The cryptography module is the fundamental component of our DTLS implementation since it provides all the functionalities to authenticate and encrypt/decrypt sensitive information. It defines the key exchange algorithm and associated ciphersuites. From the DTLS RFCs [34], the supported key exchange algorithm are pre-shared Key (PSK), Diffie-Hellman and RSA, while the ciphersuites are based on different combinations of MAC algorithms and AES. The implementation of the MAC algorithms and AES are provided by the relative modules, presented in the next sections. In our implementation, since we are considering performance-constrained environments, we only accounted for ciphersuites with the PSK key exchange algorithm, that only uses symmetric key algorithms. The support to the Diffie-Hellman key generation and exchange will be provided through the lightweight security association of Section 5.2.1.

5.4.3 Keyed-Hash Message Authentication Code (HMAC)

This module provides the functionalities for message authentication. In particular, HMAC is a specific construction to calculate a message authentication code (MAC) involving a
cryptographic hash function in combination with a secret cryptographic key. As with any MAC, it may be used to simultaneously verify both the data integrity and the authentication of a message. The definition of the procedure is general and thus any cryptographic hash function, such as MD5 or SHA-256, may be used in the calculation of a HMAC. Usually, the resulting MAC algorithm is termed according to the hash function that it uses, e.g., HMAC-MD5 or HMAC-SHA256. In our implementation we decided to use SHA256 as the hash function. However, the module is quite general and can be adapted to use any hash function. Note that the cryptographic strength of the HMAC depends upon the cryptographic strength of the used hash function, the size of its hash output, and on the size and quality of the key. The algorithm for computing the HMAC of a generic message breaks up a message into blocks of a fixed size and iterates over them utilizing a hash function. For example, MD5 and SHA-256 operate on 512-bit blocks. The size of the output of HMAC is the same as that of the underlying hash function (128 or 256 bits in the case of MD5 or SHA-256, respectively), although it can be truncated if desired.

![Diagram of the HMAC generation.](image)

**Code 1: HMAC pseudocode**

```plaintext
function hmac(key, message){
    opad = [0x5c * blocksize] // blocksize is that of the underlying hash function
    ipad = [0x36 * blocksize]

    if (length(key) > blocksize) then
        key = hash(key) // 'hash' is the underlying hash function
    end if

    for i from 0 to length(key) - 1 step 1
        ipad[i] = ipad[i] XOR key[i]
        opad[i] = opad[i] XOR key[i]
    end for

    return hash(opad || hash(ipad || message)) // || is concatenation
}
```

Date: June 11, 2013
Version: Final
Fig. 15 shows how to compute the HMAC of a generic message $M$ to be authenticated, where $K$ represents the secret cryptographic key padded with extra zeroes to the input block size of the hash function, $ipad$ and $opad$ are, respectively, the inner and outer padding (hexadecimal constants required to the security proof behind the HMAC construction), and $HASH$ is a cryptographic hash function. For additional clarity, in Code 1 we present the pseudocode of the HMAC procedure.

### 5.4.4 Counter with Cipher Block Chaining Message Authentication Code (Counter with CBC-MAC or CCM)

This module offers the functionalities to authenticate and encrypt a message using Advanced Encryption Standard (AES) in CCM mode. Counter with CBC-MAC is a mode of operation for cryptographic block ciphers that provides both authentication and confidentiality by combining the encryption functionalities offered by the Counter mode (CTR) with the authentication functionalities offered by the CBC-MAC mode. In what follows, we explain in greater details both of them.

**Counter mode (CTR)** [43], is a mode of operation that is used in conjunction with a symmetric key block cipher algorithm, such as AES, to obtain confidentiality over several blocks using the same cipher key, and to avoid pattern recognition by a possible intruder. For each block, a counter block, that must be different among the blocks for all messages under a given key (also called nonce), is encrypted using a block cipher with a given key to produce the keystream. The resulting output block is then XORed with the corresponding plaintext block to obtain the ciphertext block and vice versa. For the last block, which may be a partial block, only the most significant bits related to the partial block length are used whereas the remaining bits are discarded. For both CTR encryption and decryption, the forward block cipher functions can be performed in parallel. The CTR mode of operation is shown in Fig. 16, for both encryption and decryption, where $K$ is the symmetric cipher key.

![Diagram of the CTR encryption (decryption) mode.](image)
Cipher Block Chaining with Message Authentication Code (CBC-MAC) is a procedure to generate a message authentication code using a block cipher in CBC [43] operation mode. The message to be authenticated is encrypted in CBC mode with a deterministic initialization vector consisting of an all-zero block. The sequence to be encrypted must be padded to a multiple of the cipher block size (block size), simply by adding zeroes at the end of the message. Further, the padded data is divided into blocks, each matching the block size. The first message block is XORed with the initialization vector before being encrypted using a symmetric-key block cipher with the same key. The result is then XORed with the second message block and encrypted again. The last message block is XORed with the previous block and given as input to the block cipher. The resulting block, or part of it (leftmost bits), represents the authentication tag. Fig. 17 shows the CBC-MAC mode, where plaintext $i$ represents the $i$-th message block.

![Diagram of the CBC-MAC mode of operation.](image)

As CBC-MAC is not secure for variable-length messages, usually the length of the message is included in the first block. Since the input block to each encryption operation depends on the result of the previous one, it cannot be run in parallel. At the receiver side, the message authentication code is computed and compared with the received one to verify authentication and integrity of the received data message.

Counter with CBC-MAC (CCM) mode [44, 45] provides both authentication and confidentiality of data by combining the techniques of CTR and CBC-MAC modes. CCM is a generic combined encryption and authentication block cipher mode that is also specified by the IEEE 802.11 MAC standard. The key insight is that the same secret key can be used for both authentication and encryption [46], provided that the counter values used in the encryption do not collide with the (pre-)initialization vector used in the authentication. The block cipher used in the implementation of CCM is AES-128 as specified in [47]. This block cipher is used with symmetric keys with the same size as that of the block cipher, which in our implementation has been set to 128 bits. The implementation of CCM includes two procedures: forward transformation (Fig. 18), executed at the transmitter side to encrypt and authenticate link layer packets before their...
transmission over the channel and inverse transformation (Fig. 19), executed at the receiver side to decrypt and validate the data. The forward transformation involves the execution of: an input transformation (CCM-1), an authentication procedure (CCM-2) and an encryption transformation (CCM-3), as shown in Fig. 18. CCM-1 involves the transformation of the two CCM input strings \( m \) (the original message) and \( a \) (an associated data sequence) onto the strings \( \text{PlainTextData} \) and \( \text{AuthData} \), to be used by the encryption transformation and the authentication procedure, respectively. Thus, the authentication transformation step tags the \( \text{AuthData} \) using the tagging transformation and produces an authentication tag. Finally, the \( \text{PlaintextData} \) and the authentication tag formed earlier are encrypted using the encryption transformation step resulting in an encrypted message called \( \text{Ciphertext} \) and an encrypted authentication tag.

![Diagram of CCM forward transformation](image1)

**Figure 18: CCM forward transformation.**

![Diagram of CCM inverse transformation](image2)

**Figure 19: CCM inverse transformation.**

The CCM mode inverse transformation involves the execution of a decryption transformation and an authentication checking transformation as shown in Fig. 19. The decryption transformation is similar to the encryption transformation explained above, resulting in a decrypted authentication tag and the \( m \) string. Further, the authentication checking transformation step uses the input transformation explained above to form \( \text{AuthData} \) by using as inputs the \( a \) string and the \( m \) string established during decryption transformation. Furthermore, it employs authentication transformation, with \( \text{AuthData} \) as input, to form \( \text{MACTag} \) and compares it with the authentication tag established during decryption transformation. If both the tags are equal, the authentication is valid; otherwise the resultant \( a \) and \( m \) strings are rejected thus completing the overall CCM procedure. Further details about all the CCM implementation steps described above are available in [45, 46] and are beyond the scope of this deliverable.

We note that the procedures presented in this Section require to handle lost and out of order messages. However, this can be obtained through the DTLS protocol that implements all the functionalities required to handle a non standard sequence of messages (see [39]). In our implementation, we ported and adapted the HMAC implementation from tinydtls [42] into TinyOS modules.
5.4.5 Rijndael

The Rijndael module provides the implementation of the Advanced Encryption Standard (AES). AES is a cryptographic algorithm defined by the US National Institute of Standards and Technology (NIST) in 2001, as Federal Information Processing Standard (FIPS) Publication 197 [47] and used by US Federal departments and agencies or non-Federal Government organizations to protect sensitive and unclassified electronic data.

It is a symmetric block cipher used to encrypt and decrypt information with the same key, based on the Rijndael algorithm. Unlike Rijndael that handles more block sizes and key lengths, the AES is designed to encrypt and decrypt data blocks of 128 bits using cipher keys of 128, 192, or 256 bits. It has supplanted the Data Encryption Standard (DES) in many cryptography applications.

In our implementation we ported and adapted the Rijndael implementation contained in the crypto sub-system of the OpenBSD operating system [48] into a TinyOS module. Our module provides all the commands required to encrypt and decrypt data.

We note that, constrained devices usually provide hardware AES implementation so that all the repeated computations can be offloaded to the hardware.

5.4.6 Secure Hash Algorithm (SHA)-2

The SHA-2 module provides the implementation of a set of cryptographic hash functions (SHA-224, SHA-256, SHA-384, SHA-512) designed by the U.S. National Security Agency (NSA) and published in 2001 by the NIST as a U.S. Federal Information Processing Standard. The term hash function refers to an algorithm that takes an arbitrary set of data elements as input and transforms (hashes) them into a single value of fixed length. The idea behind it is that the computed hash value may be used to verify the integrity of the copies of the input data without providing any possibility to derive them. This is because the hash function is irreversible and thus the hash value may be freely distributed or stored, as it is used for comparative purpose only.

Clearly, the level of security provided by a hashing algorithm is entirely dependent upon its ability to produce a unique value for any specific set of data. Thus, since the size of the input data is usually greater that the size of the output hash value, a hash function can produce the same hash value for two different sets of data. This condition is referred to as a collision, which raises the possibility that an attacker may be able to computationally craft sets of data which provide access to information secured by the hashed values of pass codes or to alter sensitive data in a fashion that would not change the resulting hash value and would thereby escape detection. SHA-2 consists of a set of four hash functions that differ in the size of the hash value called digest (224, 256, 384 or 512 bits) and computational complexities.

In our implementation we ported and adapted the SHA2 implementation from Aaron Gifford [49] into a TinyOS module.
5.5 Testing

In this section we show some results about the testing of the secure implementation presented before. We start by describing a simple scenario in which a constrained node (Client) initiates a secure communication with another constrained node (Server). This basic scenario is used as a test case of our DTLS implementations. Starting from this, we will show how we can integrate our implementation with existing 6LoWPAN and CoAP modules in order to realize the lightweight security association presented in Section 5.2.1. The implementation and testing of the scenario presented in this section have been done using the Zolertia Z1 Platform [50].

5.5.1 Basic scenario

The basic scenario is used as a test case for our DTLS implementation. It entails a simple communication example where two constrained nodes communicate through a secure DTLS channel. To this end, we added the DTLS support to a simple TinyOS application that send a string using the radio. An example of such application can be easily obtained from the example applications contained in the TinyOS package. To add the DTLS support to such application we need to modify the configuration file as shown in Code 2.

### Code 2: Sample application with DTLS, Configuration

```c
// TestAppC_DTLS.nc

#include "dtls.h"

...

implementation{
    ...
    components dtlsAppC;
    App.DTLS -> dtlsAppC;
}
```

In the module implementation, instead, we need to implement the event of the DTLS interface, create a DTLS context with the information about the destination address and port, and call the function to connect to the remote host. An example of the module implementation is given in Code 3.

### Code 3: Sample application with DTLS, Module

```c
// TestC_DTLS.nc

#include "dtls.h"

module TestC_DTLS{
    uses interface DTLS;
    ...
```
implementation {

...  

  event int16_t DTLS.get_key(struct dtls_context_t *ctx, const session_t *session,
                           const unsigned char *id, size_t id_len, const dtls_key_t **result) {
    // Implementation of how to obtain the secret key
  }

  event int16_t DTLS.send_to_peer(struct dtls_context_t *ctx, session_t *session,
                                  uint8 *data, size_t len) {
    // Implementation of how to send data to a remote peer
  }

  event int16_t DTLS.read_from_peer(struct dtls_context_t *ctx, session_t *session,
                                      uint8 *data, size_t len) {
    // Implementation of what to do with the received decrypted data
  }

  event int16_t DTLS.signal_event(struct dtls_context_t *ctx, session_t *session,
                                  dtls_alert_level_t level, uint8_t code){
    // Implementation of how to send an event to the remote peer
  }

  ...

  static dtls_context_t *dtls_context;

  event void Boot.booted() {
    ...
    // Initialize the DTLS module
    call DTLS.dtls_init();
    // Create a DTLS context
    dtls_context = call DTLS.dtls_new_context();
  }

  ...

  // Container for information about the remote peer
  static session_t dst;

  event void AMControl.startDone(error_t err) {
    ...
    // Connect to the remote peer using DTLS, this start the handshake protocol
    call DTLS.dtls_connect(dtls_context, &dst)
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
    ...

}
As shown in Code 2 and 3, the steps required to enable a secure communications with DTLS to a generic applications are quite simple. In this basic scenario we do not use any key exchange mechanism. Instead, we consider that each constrained node is programmed with a pre-shared key that will be used for both authentication and encryption. In this case, the handshake protocol does not include some of the messages presented in Fig. 14. In particular, in order to initiate a secure communication, the client indicates its willingness to use the pre-shared key authentication by including one or more PSK ciphersuites in the ClientHello message. If the DTLS server also agrees to use pre-shared keys, it selects one of the PSK ciphersuites, places the selected ciphersuite in the ServerHello message, and includes an appropriate ServerKeyExchange message. The client indicates which key to use by including a “PSK identity” in the ClientKeyExchange message. To help the client in selecting which identity to use, the server can provide a “PSK identity hint” in the ServerKeyExchange message. If no hint is provided, the ServerKeyExchange message is omitted. After this exchange, client is able to send an encrypted message that notifies the server that it has finished its operation and, after receiving it, the server replies with an encrypted end message. The handshake is thus completed and the nodes can start sending encrypted application data. From our initial test, the additional memory consumption due to the DTLS module are 2 Kbytes of RAM and 12 Kbytes of ROM.

5.5.2 Integration of 6LoWPAN

In this scenario, we integrated our DTLS implementation with an existing 6LoWPAN implementation. In particular, we have considered the SIGLoWPAN implementation [51] of 6LoWPAN. We have modified the application discussed in the previous section in order to add the support for 6LoWPAN and we have tested the previous scenario with the new addressing provided by IPv6. Furthermore, we have introduced an additional node and we let it act as a relay node between the client and the server. Thus, we have realized and tested a secure DTLS communications between two constrained node with a gateway in between.

5.5.3 Complete scenario with PSK

In our final testing scenario we have tested the communication between an unconstrained and a constrained node. To this end, we have created a simple desktop application using the tinydtls implementation [42] and we have configured a constrained node to act as a base station in order to route the traffic from the desktop application to the constrained network. By doing so, we have been able to create a secure DTLS communication between a constrained and an unconstrained node, using a gateway to connect the two entities. In this scenario, we are still considering the use of PSK. We are currently working on the final step of the development that involves the implementation of the key exchange algorithm of
Section 5.3. Note that this last step is quite straightforward since, according to the security association presented in Section 5.2.1, it can be seen as a particular case of this scenario. This is because the gateway takes care of the key generation process and forwards the secret key to the constrained node. When this key is received by the constrained node, the latter can use it as done for the PSK scenario presented in this section.

5.6 Conclusion

In this section of the deliverable, we have proposed a lightweight procedure to set up secure end-to-end channels between unconstrained (and remote) peers and IoT devices, where the lightweight character of our proposal is due to the offload of computationally intensive tasks from constrained devices. We have provided some insight about our DTLS implementation as well as a description of our testing scenarios. Our current work is devoted to the refinement of the lightweight security procedure, along with the finalization of its implementation and experimentation in real testbeds. Finally, we are interested in implementing and testing different ciphersuites in order to better assess their applicability to IoT devices.

6 Collaborative Security Protocols

In this section, we explore efficient key establishment protocols that are conceived to delegate cryptographic computational load to less resource-constrained nodes in a collaborative scheme, through exploiting the heterogeneity of IoT nodes. We present a family of lightweight key exchange protocols for the Internet of Things, which allows a highly resource-constrained IoT node to obtain assistance from more powerful nodes in order to securely derive a shared secret with a peer.

We first review, classify and evaluate the existing key establishment protocols. We then describe the proposed cooperative key transport and key agreement schemes for IoT nodes. Our solutions rely on widely adopted key establishment protocols, which we extend to distribute their heavy computational operations to collaborating nodes. Two distribution techniques, simple partition and threshold distribution, are detailed for each key establishment scheme. Finally, we discuss performance of the proposed techniques compared to the basic protocols.

6.1 Review of Key Establishment Schemes

Key establishment protocols, also named key exchange protocols, are used to "provide shared secrets between two or more parties, typically for subsequent use as symmetric keys for a variety of cryptographic purposes" as stated in [52]. These purposes include the use of symmetric ciphers and message authentication codes, which are in turn used as security primitives for enabling various security protocols such as source authentication, integrity protection or confidentiality. The word "protocol" in the above definition could be misleading, because it is used in multiple contexts in which its sense changes slightly. It has thus to be clarified first.
6.1.1 Algorithmic protocols, communication protocols

A protocol can be defined as "a multi-party algorithm, defined by a sequence of steps precisely specifying the actions required of two or more parties in order to achieve a specified objective" [52]. This definition however encompasses two kinds of protocols that exist in the world of telecommunications and that collide in the field of security. On one hand, classical communication protocols of the OSI model – as specified for example in the Internet Engineering Task Force – define how two or more networked entities interoperate. These protocols include precise packet format specification along with state machine definitions. On the other hand, cryptographic algorithmic protocols define how two or more logical entities carry out a cryptographic operation. They define mandatory elements for doing so, such as the data structures that have to be transported, and the corresponding order. They do not specify, however, how data are to be transported (e.g., encoding, optional parameters, resilience support, networking parameters, etc.).

Let us take the example of the key establishment operation for the IPsec protocol. The key establishment communication protocol for IPsec, in the sense of the first definition, is the Internet Key Exchange (IKE, [53]) protocol. However, the key establishment algorithmic protocol for IPsec, in the sense of the second definition, is the cryptographic protocol on which IKE relies, that is, the Diffie-Hellman protocol. This distinction is clear and easily understandable. Things become more complex however when a single communication protocol, such as EAP, can leverage on a multitude of distinct algorithmic protocols. Complexity increases even more when the algorithmic protocol within a well-known telecommunication security protocol such as TLS can be entirely modified through the mere change of one bit in the handshake sequence. The distinction between these types of protocols is therefore of high importance. Unless otherwise stated, this chapter deals with cryptographic algorithmic protocols.

6.1.2 Classification of key establishment protocols

Key establishment protocols can be classified according to three criteria: the key delivery scheme (key transport or key agreement), the underlying cryptographic primitive family (symmetric or asymmetric) and the authentication method. The number of involved peers (2, peer-to-peer or 3, server-assisted) is sometimes added to these criteria. These notions are discussed in what follows.

6.1.2.1 Key transport vs. key agreement

A two-party key transport protocol is a protocol that runs between two peers, in which one or more secret value(s) are generated at one or both peers and securely transferred to the

---

3Actually, a protocol suite made of the AH and ESP protocols.
4In the literature, these protocols can be designated as "methods", "algorithms" or "sub-protocols".
5A key establishment protocol runs between two or more parties. In this paper though, we focus on peer-to-peer (pairwise) key establishment and do not consider the joint setup of a group key between more than two parties.
other peer. The resulting key is obtained as a function of the transferred secret values and possibly of other parameters that may have been exchanged as part of key transport.

In a one-pass key exchange, only one secret value is sent from one of the peers to the other. The established key may be either this secret value itself, or may be derived from it along with other parameters, such as nonces. In a two-pass key exchange, both peers exchange secret values that are used as input for the key generation function. Note that it is generally not a good idea to let one partner entirely control the key value.

A variety of server-assisted key transport is the distribution of a session key from a central server (key distribution center) to two peers. This requires of course that the central server be able to perform the distribution in a secure manner, e.g. through pre-established secured channels to both peers. Another, less frequent, variety of server-assisted key transport consists for the server to let one peer generate the session key, obtain it from this peer, and retransmit it over another secure tunnel to the second peer. In this second variety the assisting server is called a key translation center.

A two-party key agreement protocol is a protocol that runs between two peers, in which the resulting key is derived at both peers from public information exchanged between the peers. While the said public information might take the form of an encrypted secret, the decrypting of this encrypted secret by either the recipient peer or by the originating peer itself is never required.

The Diffie-Hellman (DH) protocol [54] is the best known and most widely used key agreement protocol. It requires that two peers A and B first agree on appropriate prime \( p \) and generator \( g \). A and B then choose secret values, respectively \( a \) and \( b \), compute the corresponding public values, respectively \( g^a \mod p \) and \( g^b \mod p \), and exchange these public values with each other. The same Diffie-Hellman shared secret \( K \) is then obtained at A by computing \( (g^b \mod p)^a \mod p \) and at B by computing \( (g^a \mod p)^b \mod p \). The Diffie-Hellman protocol is depicted in Fig. 20.

![Figure 20: Diffie-Hellman key agreement.](image)
An often claimed security property of the Diffie-Hellman protocol is the perfect forward secrecy. This property ensures that the established secret could not be retrieved even though all long-term secrets of both peers are divulged. In the base Diffie-Hellman protocol, \( a \) and \( b \) are random numbers that are dynamically chosen as part of the key management protocol and immediately erased from memory afterwards. They could therefore not be qualified as "long-term secrets", which ensures that the Diffie-Hellman protocol fulfills the perfect forward secrecy property. This should not be generalized to all key agreement protocols, though. Some key agreement protocols are based on key pre-distribution. For example, the variant of the Diffie-Hellman protocol used in the HIP-DEX key establishment communication protocol (reviewed in what follows) requires that the Diffie-Hellman secrets \( a \) and \( b \) be statically fixed and remain the same in all key establishment operations. This use of Diffie-Hellman leads to losing the perfect forward secrecy property that is generally associated with it.

### 6.1.2.2 Cryptographic primitives

Both key transport and key agreement exist in embodiments that rely either on symmetric or on asymmetric cryptography. These cryptographic primitives should not be confused with those of the authentication mechanisms that may be integrated with the key establishment protocol and that are the subject of the next subsection. Let us take again the example of the Diffie-Hellman key agreement protocol. Diffie-Hellman is based on asymmetric cryptography primitives (actually, most of key agreement protocols are). Yet Diffie-Hellman, natively unauthenticated and vulnerable to man-in-the-middle attacks, has to rely on authentication techniques, some of which can be based on symmetric techniques.

Considering only the key delivery scheme and the cryptographic primitive type, four cases are possible:

- **Key transport with symmetric cryptographic primitives:** this category regroups algorithms in which two peers, already owning a shared key, derive another one. Such operation typically happens when a symmetric key has to be refreshed, or when an ephemeral secret (e.g. transient session key) has to be derived from a long-term one.

- **Key transport with asymmetric cryptographic primitives:** This category comprises various key establishment protocols ranging from simple one-pass encryption of a secret key with a public key to more complex X.509 keying protocols.

- **Key agreement with symmetric cryptographic primitives:** A corresponding protocol, Blom’s scheme, is presented in [52]. Although interestingly dissociating the key agreement notion from the Diffie-Hellman protocol, one cannot help but notice that such algorithmic protocols are not used by main (and even minor) communication protocols.

- **Key agreement with asymmetric cryptographic primitives:** With rare exceptions, this category is composed of the Diffie-Hellman protocol and its variants.
6.1.2.3 Authentication method

Authentication for a pairwise key establishment protocol relates to the ability, for one or both nodes that undertake it, to bind the established key material with the identity of its peer. While it is generally a good thing to have a pairwise key establishment protocol authenticate both peers to each other, it is not always the case. Commonly, only one peer is authenticated to the other; the authentication of the other peer, if required, has then to be ensured by another mechanism, possibly at another layer.

Authentication brings us back to the distinction we introduced in the beginning of this chapter between algorithmic and communication protocols. Some algorithmic protocols natively provide authentication. This is the case, for example, of a one-pass key transport protocol wherein a session key k is sent from a node A to its peer B, encrypted with B’s public key. This protocol achieves indeed more than confidential key delivery: it proves to A that a node knowing k must be identified as B, since only B is expected to have been able to decipher the message containing k.\(^6\) On the other hand, as mentioned above, the Diffie-Hellman protocol does not natively provide authentication. The Diffie-Hellman public values have therefore to be authenticated at communication protocol level, as is done by the IKE protocol, which ensures through digital signatures or keyed hashes that their origins can be validated.

Like those of key establishment protocol, the cryptographic primitives that underlie the authentication method can be classified as symmetric vs. asymmetric techniques. With the objective of defining the best practices for an IoT key establishment protocol, it is worth, though, going beyond this distinction and considering the underlying identity models. The categories of authentication that can be distinguished are listed hereafter. For clarity reasons, this list is made simpler by assuming that mutual authentication is desired, and that both peers use the same authentication method with each other.

- **Shared secret-based authentication**: This is the classical symmetric authentication scheme wherein two parties are statically configured with, or otherwise acquire, a common shared secret mapped to their respective identities.

- **Static public key authentication**: In this asymmetric authentication scheme, the two parties are statically configured with their respective public keys, mapped to their respective identities. Proving the knowledge of the corresponding private key implicitly ensures ownership of the matching identity.

- **Certificate-based authentication**: This is a variant of the previous category, wherein the mapping of a public key to an identifier is not a static configuration parameter but is obtained in the form of a signed certificate. Certificate-based authentication requires that a third party, the certificate authority, be trusted by both authenticating peers.

- **Cryptographically generated identifiers**: This family of asymmetric techniques changes the implicit assumption that any kind of identifier can be authenticated, provided that

---
\(^6\) The two steps of ensuring that only B may know the key k and obtaining the proof that some node knows the key k are respectively designated in [52] as implicit key authentication and key confirmation. Together, they form the explicit key authentication property.
it is securely bound to a public key. These techniques assume indeed that the authenticated identifier of a node is obtained from the node public key, e.g. in the form of a hash of this public key. Mechanisms are then defined in order to build protocol stack identifiers (typically, IPv6 addresses) from these cryptographically generated identifiers.

- **Identity-based authentication**: This last set of asymmetric techniques bases on the Identity Based Cryptography paradigm wherein, oppositely to the previous category, a node’s public key is derived from its identity (whatever the format of this identity). Like in all asymmetric techniques, a node proves its identity by providing a proof of knowledge of the corresponding private key.

6.1.2.4 Preliminary classification of key-establishment protocols

In this subsection we provide a global view of the existing algorithmic protocols, in order to ease the identification among them of the best candidates for IoT key establishment. This synthetic global view is provided in the form of a table, on which we chose to superpose the most known/used communication protocols. Usability of algorithmic protocols within communication protocols currently in use in today’s Internet is indeed a criteria that should not be left aside: the Internet of Things will definitely not start with a “clean slate” design approach, but will likely have to interoperate with widely adopted protocols of legacy Internet.

<table>
<thead>
<tr>
<th>Key delivery scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Transport</td>
</tr>
<tr>
<td>Symmetric</td>
</tr>
<tr>
<td>MIKE</td>
</tr>
<tr>
<td>Static public key</td>
</tr>
<tr>
<td>Certificate</td>
</tr>
<tr>
<td>Cryptographically generated</td>
</tr>
<tr>
<td>IBAKE</td>
</tr>
<tr>
<td>Identity-based authentication</td>
</tr>
</tbody>
</table>

Table 2: Classification of key establishment protocols with main key establishment communication protocols represented in overlay.

As can be seen in Table 2, the existing key establishment communication protocols are mainly based on asymmetric cryptography, be it for the delivery/agreement scheme itself, or for the authentication method implemented within the protocol. Empty cells in the table are mostly found in the symmetric key transport and symmetric key agreement columns.
Symmetric key transport protocols do exist but they essentially consist in key refresh / key derivation protocols, which we found did not fully qualify as key establishment protocols. Only the MIKEY protocol is included in the column, since it is generally used to distribute session keys from long-term shared keys. Symmetric key agreement protocols are uncommon and require complex setup (pre-distribution).

6.1.3 IoT key establishment: generic design decisions

This section reviews the generic design decisions that are involved in the identification of a key establishment protocol for the Internet of Things. These decisions fall into four main categories: those that are related to the fulfillment of security requirements, those that are related to pervasiveness (the Internet of Things is to encompass a wide variety of devices and networks, including legacy Internet), those that are related to efficiency (among IoT devices, some are resource-constrained) and those that are related to adoptability or interoperability (the IoT should preferably use proven and deployed technologies and protocols).

6.1.3.1 Security

Contrary to wireless sensor network security, security in the IoT context involves end-to-end communications. The decentralized and bidirectional IoT communication paradigm also rules out the definition of static client and server roles: depending on the context, it is expectable that an IoT node will act alternatively as a client and as a server. These considerations translate into two security requirements. On one hand, end-to-end security should be provided. This means that only the two participants involved in the pairwise key exchange protocol should have access to the generated key. On the other hand, mutual authentication has to be provided. The two peers that establish a key between them should in the meantime authenticate to each other and bind the generated key to their respective identities.

6.1.3.2 Pervasiveness

By qualifying the Internet of Things as “pervasive”, we refer to its foreseen universality, as a communication network interconnecting many more nodes than in today’s Internet, and actually encompassing today’s Internet. Pervasiveness puts additional requirements on a key establishment protocol for the IoT. Especially, it makes it highly unlikely that two nodes wishing to generate a key between them can leverage on a pre-existing security relationship based on long-term shared secrets or static public keys. For this reason, dynamic asymmetric key delivery schemes and authentication methods should be preferred when designing an IoT key establishment protocol.

Pervasiveness also means that any two nodes may have to interoperate with each other, without considering their respective nature. Special care should therefore be taken, when designing an IoT key establishment protocol, to make sure that two nodes with important differences in capabilities be nevertheless able to generate a key with each other.

Date: June 11, 2013
Version: Final

Internet of Things Architecture ©
6.1.3.3 Efficiency

Efficiency has always to be considered when designing a new protocol. Four criteria are especially relevant when assessing cryptographic protocol efficiency: the number of exchanged messages, the bandwidth that is needed, the complexity of computations, and the possibility of pre-computations. Importance of these criteria increases when designing a protocol that will have to be run by highly resource-constrained nodes with low computational power, low memory, and limited battery capacity. Overall energy consumption, induced by both computations and message exchanges, is a good metric for these nodes. A protocol will be defined as more efficient than another if it obtains a metric value inferior to that of the other, while providing the same security level.

6.1.3.4 Adoptability

The Internet of Things will not emerge through the definition of entirely novel protocols. The approaches that rely on key generation schemes or authentication methods of limited usage should not be favored. Of course, interoperability mechanisms with these latter should be developed when desirable, though.

At this stage, it is worth quickly describing the roles of the two most widely adopted end-to-end security protocols we refer to in Table 2.

The Internet Protocol security (IPsec) [55] resides at the Network Layer of the OSI Model, which enables it to function independently of any application. It creates a secure (encrypted and/or integrity-protected) and authenticated tunnel between two endpoints, through which data can be exchanged safely, without being vulnerable to eavesdropping, packet forging/replaying or sender spoofing attacks.

Transport Layer Security TLS [56] provides the same security services at the transport layer while still being application-independent. Hence it can encapsulate higher-level protocols layering on top of the transport layer protocols. TLS has been designed to work with reliable transport protocols providing in-sequence delivery, such as the Transmission Control Protocol (TCP). Recently, a datagram-oriented variant DTLS has been proposed to operate on top of datagram-oriented transport protocols, such as the User Datagram Protocol (UDP). Both IPsec and TLS have the same design and provide equivalent security measures.

IPsec and TLS communication protocols rely on the use of cryptographic mechanisms such as encryption/decryption block ciphers and hash functions, in order to ensure the required security services for a communication. In turn, each of these mechanisms requires an initial key establishment phase allowing two communicating entities to authenticate each other and set up the required cryptographic keys. TLS protocol is preceded by a handshake protocol called TLS Handshake, which is responsible for key establishment and authentication. Likewise, the Internet Key Exchange [53] protocol and the Host Identity Protocol Base Exchange (HIP BEX) [57] are both designed to perform keying for IPsec protocol.

Each of these key exchange schemes independently implements specific techniques and cryptographic algorithms to derive a secret key and ensure the required mutual authentication between the endpoints of a communication.
6.1.3.5 Final classification of key-establishment protocols

From the design decisions reviewed above, we can adapt our initial classification of key establishment protocols in order to identify among them the most suitable to the Internet of Things. The results of this identification are presented in Table 3.

<table>
<thead>
<tr>
<th>Authentication method</th>
<th>Key delivery scheme</th>
<th>Server-assisted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric</td>
<td>Shared secret</td>
<td></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Static public key</td>
<td>Low security</td>
</tr>
<tr>
<td></td>
<td>Certificate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cryptographically generated</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Identity-based authentication</td>
<td>Low adoption</td>
</tr>
</tbody>
</table>

Table 3: Refinement of the key establishment protocols classification.

Table 3 was obtained as follows. First, solutions relying on key distribution were discarded, as they did not meet end-to-end security requirement. Then, solutions relying on symmetric cryptography or assuming initial knowledge of peer public key were discarded as they did not meet the pervasiveness requirement. It has to be noted that these first two requirements do not contradict each other: dynamic obtaining of asymmetric public keys through certificate and induced reliance on a certificate authority are different from letting the trusted third party generate the keys for both peers, and be thus in position to launch a key escrow attack. Finally, we discarded the solutions that were based on identity-based cryptography, which we considered not sufficiently adopted.

6.1.4 Applicability of existing key exchange schemes for IoT scenarios and related work

Key establishment schemes used by IPsec and TLS as described above involve heavy public key cryptographic primitives and impact both energy and storage resources of a communicating entity. The resource constraints of most IoT components limit the implementation of these complex cryptographic mechanisms required to perform the key establishment, which could rapidly drain their resources and reduce the network performance. Existing end-to-end security protocols with their actual resource intensive design could not directly cope with the envisioned scenarios and requirements in the IoT. The feasibility of these security standards
has to be revisited to adapt them to the IoT scenarios. In the literature, several key establishment solutions have been proposed for resource-constrained nodes. The initial definition of requirements for an IoT key establishment protocol that we produced in the beginning of this section will now help us in assessing the relevance of these proposals.

Most proposed approaches rely on symmetric cryptography primitives due to their reasonable resources consumption. These solutions [58], typically referred to as symmetric key establishment schemes, are the most adopted for resource-constrained devices, like, e.g., sensor nodes. These schemes are based on pre shared keys between different nodes belonging to the same sensor network. In view of the IoT scenarios, however, a sensor node is considered as a part of the Internet able to establish end to end communications with external entities without requiring any initial knowledge of each other or any prior authentication context.

The use of public key cryptography as proposed in TLS and IPsec eliminates the need for pre-defined secured context between the two endpoints of a communication but the underlying cryptographic primitives are too heavy to run on resource constrained nodes. Independently from the IoT design and security challenges, many solutions have been proposed to reduce the computational cost of the public key cryptography. Elliptic Curve Cryptography (ECC) was the top choice among public key cryptographic algorithms due to its lower energy consumption, fast processing time, compact signatures, and small key size. In the last years, many popular security protocols have been reviewed to enable the use of ECC algorithm. In [59], a comparative performance analysis has been conducted between RSA based TLS handshake and ECC based handshake. Results have shown that using ECC reduce significantly the energy consumption of the TLS key establishment process compared to RSA. Nevertheless, the required energy and memory costs of ECC are still non negligible, being in the order of magnitude of one millijoule. In practice, this would be hindering for resource constrained nodes in the IoT. Alternatively, [60] [61] focus on making the well-known RSA public key cryptosystem [62] more adapted to resource constrained devices using a small RSA public exponent (e) and a short key size. This comes, however, at the price of a lower security level [63].

HIP Diet Exchange (DEX) [64] has been recently proposed as a lightweight variant of HIP specifically designed to reduce the heavy cryptographic cost of its basic key exchange. The most requiring part in HIP BEX is the computation of the Diffie-Hellman key, immediately followed by the generation of the Diffie-Hellman public keys. Signatures computations and verifications for mutual authentication are still non negligible operations from the point of view of a resource-constrained node.

HIP DEX proposes that a node use a long-term Elliptic Curve Diffie-Hellman (ECDH) public value as its Host Identifier. DEX then adapts the key exchange, as depicted in Fig. 21.

The Host Identifier being itself the Diffie-Hellman public key, there is no need to authenticate it through asymmetric cryptography. The knowledge of the DH key is enough to prove that a node is a legitimate entity in the exchange.

As compared to HIP BEX, the single computation of the long-term Diffie-Hellman public values eliminates their generation costs, though at the expense of the perfect forward secrecy property. Likewise, the use of Elliptic Curve Diffie-Hellman and the fact that no other asymmetric cryptography operation is required make the key exchange lighter. However,
even ECDH-based key exchange [59] remains too heavy to be supported by a resource-constrained node.

This unsuitability of the purposely designed HIP-DEX protocol is representative of the need for novel IoT-specific solutions. According to the literature, the design of an efficient key establishment approach for existing security standards that clearly addresses the heterogeneous IoT communications has not been undertaken yet [65]. Further careful design is required to reduce the energy cost of key establishment schemes while taking into account the heterogeneous nature and the end to end security requirements of the IoT.

### 6.2 Proposed Solution for collaborative Keying

Our proposed solution adopts a holistic approach and focuses on reducing the computational load expected to be performed on constrained devices instead of proceeding by only reducing the cost of cryptographic primitives, as proposed before. The main rationale of our approach is to make a highly resource-constrained node able to establish secure contexts with other unconstrained nodes within a heterogeneous IoT architecture. We redesign existing key establishment schemes and propose to delegate their heavy cryptographic load to less constrained nodes in their neighborhood. During the key exchange, these assisting nodes, or “proxies”, take charge of the session key derivation, in a collaborative and distributed manner. The reconstitution of the session key is completed at the two endpoints of the communication in order to guarantee its secrecy.

Several constraints have been considered in the design of our approach (i) the collaborative scheme must not come at the expense of a key disclosure risk or a collusion attack (ii)
in case of a proxy unavailability or a greedy behavior, the system should continue to run properly (iii) each proxy is required to prove its legitimacy by proving that it is authorized by the constrained node to act on its behalf.

6.2.1 Considered network model

Our network model considers a global IoT infrastructure that interconnects heterogeneous nodes with different capabilities in terms of computing power and energy resources. We especially consider in this work three different types:

- **Highly resource-constrained nodes**, unable to support the computation cost of asymmetric cryptographic operations required by the key exchange phase while requiring end to end security (e.g., sensor nodes of type TT2).

- **Proxies in the neighborhood**, less constrained and therefore able to perform cryptographic operations (e.g., terminals of type TT1). These nodes may either be dedicated assisting servers or nodes belonging to the same local infrastructure, though being less impacted by energy constraints (e.g., having energy harvesting capability).

- **Unconstrained nodes not belonging to the same local infrastructure** with high energy, computing power and storage capabilities (e.g., line-powered remote servers of type TT1).

The considered scenario in this work is modeled by a highly resource-constrained sensor node (the source node A) that needs to exchange sensitive data with an external server (the destination node B) on an end-to-end basis. These two entities are supposed to have no prior knowledge of each other.

6.2.2 Assumptions

- After the initialization phase, every sensor node shares pairwise keys with a subset of its one-hop neighbors. These keys may have been generated during a specific bootstrapping phase using a trusted key management server or through more subtle mechanisms such as transitive imprinting.

- The highly resource-constrained node is able to identify a set of less resource-constrained nodes that are available for supporting heavy cryptographic operations on its behalf.

- There exists an unconstrained local trusted entity within the sensor network that owns a shared secret with all nodes in the sensor network and a public/private key pair.

- The external server does not communicate with the sensor network trusted entity but are statically configured with or able to validate its public key.
6.2.3 Preparation of involved entities

As an initial phase, the resource constrained sensor node A carefully selects the \( P_1, \ldots, P_n \) proxies that will assist its key exchange based on the reputation of the nodes in the network and their actual resource capabilities.

Our approach requires that these nodes will process messages on behalf of the resource constrained node during the key exchange. Hence authorization and authentication questions arise at the proxy side, since these nodes should be provided with a representativeness proof. This proof could be a certificate including the proxy’s public key associated with the right “authority to sign on behalf of A”, all of which are signed with the source’s private key and delivered ‘offline’ to the proxy, regardless the current exchange. However the use of long-time authorization certificates can be diverted for malicious exploits.

Hence, the certificate should include other dynamic parameters added by the source node in order to restrict the ability of proxies to act on its behalf, such as the identity of the destination node, a session nonce, or an expiration date. In this case, the authorization proof should be delivered ‘online’ to the proxy during the protocol exchange. Nevertheless, managing dynamic certificates would be hindering for the constrained sensor node.

For this reason, we propose to move the computational load required to dynamically manage authorization proofs from the sensor node to a local trusted entity T, which will be the only entity able to assert that a proxy node is authorized to sign on behalf of A. On the other hand, the verification of each proxy’s certificate would be also heavy for the destination node; we propose to rely on the technique of authenticated dictionaries such as the Merkle tree [66] or one way accumulators [67] to efficiently authenticate participants and validate their membership to the group of selected proxies at the server side.

Upon receiving their proof material, proxies are prepared to participate to the collaborative process.

6.2.4 Key exchange description

In order to give a clear description of the proposed collaborative process, in what follows, we separately treat each of the involved procedures.

6.2.4.1 Collaborative key transport

Collaborative one-pass key transport

During the one-pass key transport mode, the highly resource-constrained node A generates a random secret key x and relies on a set of proxies to deliver it to the server B using asymmetric cryptography. We propose two techniques to distribute computations required for the secret key delivery.

a. Simple secret partition: A starts by splitting the secret x into n parts \( x_1, x_2, \ldots, x_n \) and then sends each part \( x_i \) to the corresponding proxy \( P_i \). Upon reception of the \( x_i \) secret key part, the proxy \( P_i \) encrypts it using the server’s public key and signs the result using its private key.
We propose to use the lightweight one-time signature scheme of Lamport [68] in order for the proxy to sign messages on behalf of the constrained node. This signature scheme is especially lightweight and computationally efficient compared to other signature schemes [69]. Two drawbacks could possibly mitigate its practical applicability: on one hand, a public/private key pair should be used only once since information about the private key is divulged along with the signature itself. On the other hand, a long key will be needed to sign a long message, since the private (resp. public) key is the concatenation of all private (resp. public) values, as numerous as the message blocks and being each as long as the associated hash function output. Nevertheless, neither of these shortcomings affects our approach, which addresses one-time exchanges of short messages. In this case, we propose that T generates the Lamport private / public keys \((L_{K_i}^{-1}, L_K)\) for each proxy \(P_i\) and securely provides it with this key material along with the authorization proof in the same message.

After receiving the required key materials, the proxy signs the encrypted secret key \(x_i\) and then sends the result to the server B. In turn, B verifies the integrity of the received message using \(P_i\)'s public key and eventually decrypts \(x_i\).

We assume that each proxy \(P_i\) has initially contacted B in order to request for its certificate and to provide it with its own certificate and its proof material. In response, B verifies that the proxy has supplied a valid public key along with information from T asserting that it is a valid proxy assisting A in its key establishment process.

Having received all \(x_i\) fragments, B becomes able to recover the original secret key \(x\).

b. Threshold secret distribution: at this stage, it is worth noting that the proposed solution is based on reliable deliveries of all secret fragments \(x_i\) in order to be able to reconstitute the source's secret key at the destination node. A single missing message from a proxy makes the information incomplete for the server and may causes failure of the protocol exchange. Assuming that proxies behave as honest and reliable participants could be difficult in practice: even in scenarios where dedicated trustworthy proxies are made available to resource-constrained nodes, reliability of those proxies is not guaranteed. Hence, in case of unavailability or non-cooperative behavior of a proxy, a retransmission operation, optionally preceded by a new proxy assignment may have to take place. However, the proposed system will suffer from an additional latency.

In order to reinforce the reliability of the proposed distributed scheme, a forward error correction scheme [70] will be applied by the source A to handle losses and missing secret parts from assisting nodes.

The principle of forward error correction scheme is to add redundant parity packets to the original message, divided into multiple packets, in order for it to be recovered by the receiver even if some packets were altered or lost during the process of transmission. Let \(n\) be the total number of sent blocks, \(k\) (\(k<n\)) is the minimum number of blocks required to reconstruct the original message.

First, the source node applies this error redundancy scheme to the secret key before the split process. Then, the server B becomes able to reconstruct the session key provided that a sufficient number of packets from assisting nodes are received, without requiring the reception of all of them. This technique protects our solution from unreliable delivery in proxy → server connection, though the source node should perform more computational operations.
in the initial phase, to add redundancy to the secret key.

**Collaborative two-pass key transport**

In two-pass key transport mode, a random secret key $x$ generated by the constrained source A and a second random secret value $y$ generated by the server B are used to compute the session key.

We propose to apply the same collaborative approach as described in the one key transport scheme to deliver the secret $x$ from the source to the server. After having received a sufficient number of $x_i$ fragments, the server obtains the secret value $x$. At this stage, it generates in turn a secret key $y$ to be provided to the resource-constrained client. However, this latter cannot decrypt and verify the authentication and the integrity of the received value because of its resource constraints. For this purpose, we propose that the proxies support also the reception of the secret key $y$ on behalf of A in a cooperative manner. These nodes take charge of the computational load required to verify the received message from the server and then transmit it securely to the source. Yet, the divulgence of the secret key $y$ to the proxies would affect the security of our system. In order to preserve the secrecy of $y$, we propose to have it encrypted with the secret key $x$ reassembled at the server. The $x$-encrypted secret key $y$ is MACed with the secret $x$ and then signed with the server's private key. It is finally sent to each proxy $P_i$, which has to verify the integrity of the received packet from the server before decrypting it. Then the packet content (that is, $y$ encrypted and MACed with $x$) is securely transmitted to the client. As long as an appropriate number of the same packet is received from different proxies, the client ensures the validity of the transmitted message from the server. Consecutively, it checks the MAC in order to ensure that the server has obtained the same secret $x$ and verify the message integrity. Once the client A receives a valid message, it can obtain the transmitted secret value $y$ in order to complete the set-up of the session key.

### 6.2.5 Collaborative key agreement

The key agreement process involves heavy cryptographic computations on both parties. The most requiring part is the computation of two modular exponentiations, respectively for the generation of the Diffie-Hellman public keys and the setup of the Diffie-Hellman key. Applying the same collaborative approach in the above section, we propose to delegate the heavy cryptographic load to less constrained nodes in the neighborhood. In the two following subsections, we describe two techniques that we introduce in order to distribute the computations required by the Diffie-Hellman protocol and therefore to enable the key agreement protocol. For each of these techniques, we explain how the source’s DH private key is shared among proxies (how A computes $a_i$), how the server retrieves the source’s DH public key from the proxies’ $g^a$ mod $p$, how the server computes the shares $B_i$ of its own DH public key (how B computes $B_i$) and how the proxies use $B_i$ to obtain the $K_i$ shares of the DH session key $K_{DH}$, eventually used by A to retrieve $K_{DH}$.

**Secret exponent integer partition**

The technique described in this subsection is the simplest approach for enabling distributed
DH key exchange. The secret exponent $a$ of the source is split into $n$ parts $a_1, \ldots, a_n$ chosen such that:

$$\sum_{i=1}^{n} a_i = a \mod p .$$  \hspace{1cm} (1)

Upon reception of $a_i$, each proxy $P_i$ computes its part of the initiator’s DH public key $g^{a_i} \mod p$ and delivers it (signed) to the server. The computation of the source’s DH public key eventually occurs at the server and amounts to the product of the values received from the proxies, following:

$$\prod_{i=1}^{n} g^{a_i} = g^{\sum_{i=1}^{n} a_i} \mod p$$

$$= g^{a} \mod p .$$  \hspace{1cm} (2)

In turn, the server sends a share $B_i$ of its DH public key to each proxy $P_i$. With this first simple partition technique, $B_i$ is equal to the server’s DH public key for each proxy. The computation by each proxy of the share $K_i$ of the DH session key occurs then as follows:

$$K_i = B_i^{a_i} = (g^{b} \mod p)^{a_i}$$

$$= g^{b, a_i} \mod p .$$  \hspace{1cm} (3)

Eventually, the computation of the DH session key is made by the source, which obtains $K_{DH}$ as:

$$K_{DH} = \prod_{i=1}^{n} K_i = \prod_{i=1}^{n} g^{b, a_i} \mod p$$

$$= g^{b, a} \mod p .$$  \hspace{1cm} (4)

According to this expression, the resource-constrained node only spends $n-1$ modular multiplication operations instead of two modular exponentiation operations, with exponents of considerable length ($a$ and $b$ should have twice the length of the generated secret $K_{DH}$, as per [71]).

**Secret exponent threshold distribution**

The previous solution is based on reliable multiple hop-by-hop deliveries of secret fragments, each fragment $a_i$ being the $i$th summand of a modular integer partition of the source’s DH private key. The server needs therefore to receive all messages from all proxies in order to be able to reconstruct the source’s public key. A single missing message from a proxy makes the information incomplete for the server and may block the protocol exchange.

In order to reinforce the reliability of the proposed distributed scheme, this kind of defective proxy play has been carefully considered in the design of this second proposed approach.
for key agreement. We have implemented a robust technique that ensures a consistent recovery of the source’s DH public key at the server even in case of a proxy misbehaviour or unreliability. Note that the technique introduced above for key transport could not be adapted to a key agreement protocol, where the secret exponent a is never retrieved at B’s side.

The proposed enhanced distributed approach is based on the use of a (k, n) threshold scheme, wherein the n proxies obtain a polynomial share instead of a partition element, k polynomial shares are enough to reconstruct the source secret through the technique of Lagrange polynomial interpolation. This threshold scheme satisfies the two properties that the integer partition solution fails to provide:

1. **Recovery**: the server can recover the source’s public key provided that enough k values from proxies are received, without requiring the reception of all of them.

2. **Secrecy**: nothing is learned about the secret exponent a even if k-1 shares of it are disclosed. In other words, data delivered to the server through proxies in order to compute the source’s public key will not reveal partial information about secret exponent.

Given a polynomial function f of degree k-1 expressed as:

\[ f(x) = q_0 + q_1 x + \ldots + q_{k-1} x^{k-1} \]

with \( q_1, q_2, \ldots, q_{k-1} \) random, uniform and independent coefficients and \( q_0 = a \). Applying the Lagrange formula, the polynomial f can be retrieved as follows:

\[
 f(x) = \sum_{i=1}^{k} \left( f(i) \times \prod_{j=1 \atop j\neq i}^{k} \frac{x - j}{i - j} \right). 
\]  

(5)

From (5), the secret exponent a can be computed given any subset of k values of f(x):

\[
 a = f(0) = \sum_{i=1}^{k} \left( f(i) \times \prod_{j=1 \atop j\neq i}^{k} \frac{-j}{i - j} \right). 
\]  

(6)

In order to bootstrap the threshold distributed key agreement, the source calculates n values f(1), ..., f(n) of the polynomial f, with n > k, and sends each f(i) to the corresponding proxy \( P_i \). Each proxy computes then its part of the source’s DH public key \( g^{a_i} \mod p = g^{f(i)} \mod p \) and sends it to the server.

Upon the reception of a subset P of k values transmitted by the proxies, the server starts by computing the \( c_i \) coefficients as follows:

\[
 c_i = \prod_{j \in P \atop j \neq i} \frac{-j}{i - j}. 
\]  

(7)
Then, B computes the source’s DH public key $DHI$ based on the Lagrange formula:

$$\prod_{i \in P} (g^{f(i)} c_i \mod p) = g^{\sum_{i \in P} f(i) \times c_i \mod p}$$

$$= g^{f(0) \mod p}$$

$$= g^{a \mod p}.$$  

(8)

In order to prepare the computation of the DH session key at the source side, B starts calculating for each proxy $P_i$ ($i \in P$) the value $B_i = g^{b \times c_i \mod p}$ ($c_i$ being the $i$th coefficient calculated in the previous phase). $P_i$ is unable to compute the coefficient $c_i$ since it has no knowledge about the subset $P$ of concrete participating proxies. Having received this value, each proxy $P_i$ uses its share $f(i)$ of the source’s private exponent to compute $K_i = B_i^{a_i} = g^{b \times c_i \times f(i) \mod p}$. Each proxy delivers then this computed value to the source $A$. Upon reception of these $k$ values, the source computes the DH session key $K_{DH}$ as follows:

$$K_{DH} = \prod_{i \in P} g^{b \times f(i) \times c_i \mod p}$$

$$= g^{b \times \sum_{i \in P} f(i) \times c_i \mod p}$$

$$= g^{b \times f(0) \mod p}$$

$$= g^{a \times b \mod p}.$$  

(9)

By applying the threshold technique to improve the effectiveness of the distributed approach, the source is led to perform more computational operations in the initial phase, in order to calculate the $n$ values of the polynomial that it sends to the $n$ proxies. The cost of the computation can be better estimated if one considers another way of writing $f(x)$, as:

$$f(x) = (...((q_{k-1} \times x + q_{k-2}) \times x + q_{k-3}) \times x + ... \times x + q_0).$$

(10)

According to this expression, A performs for each computation of $f(i)$: $(k-1)$ multiplications between a scalar and a large number and $(k-1)$ summations of two large numbers. It is worth noting that $k$ and $n$ are small numbers, smaller than the number of secure relationships that the source is able to maintain. On the other hand, the polynomial coefficients are as large as the DH private key of the source.

6.3 Performance Analysis

As described above, our solution proposes to offload the expensive cryptographic computations to powerful proxies during a key exchange process; hence ensuring significant energy savings at the constrained device. Nevertheless, a communication overhead is imposed due to the message exchanging between the source, the trusted entity $T$ and the proxies. A performance analysis is therefore required to assess the respective efficiency of the proposed collaborative approaches and compare them with the basic approaches used for the key exchange.
6.3.1 Considered examples for key exchange

We consider in this section how our proposed collaborative approach, under its integer partition and threshold distribution embodiments, can be applied to IoT key establishment protocols as identified in Table 3. In accordance with this table, our examples are one asymmetric key transport protocol, and one asymmetric key agreement protocol. As explained above, we choose to base the former on a two-pass key exchange, which is more secure since it does not let a single entity entirely control the key. However, the obtained results are largely indicative of those of one-pass key transport protocols.

6.3.1.1 Example of key transport protocol

We consider in this performance analysis section Needham-Schroeder public key protocol [72] as an example of two-pass key transport protocols that has been extensively studied in the literature. Assuming, that users A and B hold each other’s authentic public key, the protocol exchange is depicted in Fig. 22.

![Figure 22: Needham-Schroeder public key protocol.](image)

Needham-Schroeder protocol provides a mutual entity authentication without requiring the use of digital signatures. Upon reception of the first message, B recovers the secret value x and then retransmits it to A which has originally generated it. As a result, B implicitly proves the possession of the private key corresponding to its claimed public key. This provides both entity authentication of B to A and assurance that B has properly received x. In turn, B verifies the entity authentication of A upon recovering its secret value y received from A in the last message. Finally, a joint key is derived as a combination of the two exchanged secret values x and y.

We assume that A is a resource constrained node unable to support public key cryptography and consequently incapable to process (encrypt and decrypt) messages during the Needham-Schroeder protocol exchange.

For this reason, we apply our proposed collaborative approach in order to delegate encryption (in the first and second message) and decryption (in the third message) operations.
required by A to a set of assisting nodes. The modified Needham-Schroeder protocol exchange is presented below in Fig. 23.

![Diagram of the modified Needham-Schroeder protocol](image)

Figure 23: Modified Needham-Schroeder protocol.

Applying our collaborative scheme, we allow the protocol exchange between the two entities A and B and ensure the required mutual entity authentication and key confirmation. The proposed technique to send x and y from B to A (encrypting y using x) gives A the possibility to ensure a proper delivery of x to B as well as to recover the secret value y for the entity authentication of B, without divulging neither x nor y at proxies. Having recovered y, A retransmits it to B relying on proxies’ assistance which makes it possible for B to check, in turn, the entity authentication of A.

6.3.1.2 Example of key agreement protocol

We consider for our performance tests the Base Exchange mechanism of the Host Identity Protocol (HIP) as an example of key agreement protocols. The reader may wonder why the Diffie-Hellman protocol is not analyzed in this section, and why a key establishment communication protocol is chosen instead. Indeed, the Needham-Schroeder protocol reviewed
above was an algorithmic one. The underlying reason is simple: contrary to Needham-Schroeder, Diffie-Hellman requires additional authentication mechanisms. The combination of Diffie-Hellman and an authentication mechanism is generally provided in the form of a key establishment communication protocol. Among these, the HIP BEX protocol is emerging as the most likely solution to be used in the IoT. The worldwide nature of the IoT addressing a lot of scenarios requires indeed a global identification framework. The maturity of HIP, its ability to provide the identifier ownership and its capability to ensure the identifier locator split paradigm make it the most suitable protocol for identification in the IoT.

HIP Base Exchange (BEX) performs the authenticated Diffie-Hellman protocol to establish a session key between two HIP peers designated in the literature as the initiator and the responder.

The entire of HIP BEX was already depicted above (Fig. 21). We apply in this section our proposed collaborative approach to reduce its computational requirements at the resource constrained initiator side.

The packet exchange that makes up the modified key exchange is illustrated in figure below.

6.3.2 Computational cost

In order to precisely quantify the energy savings at the constrained source node, we have implemented the cryptographic operations it performs in Needham-Schroeder protocol and HIP BEX protocol, considering both their basic and collaborative approaches. We have evaluated their cryptographic energy costs using Crypto++ library [73]. With respect to error correction, we have chosen to rely on the Reed-Solomon (RS) code [74] in the threshold distributed approach of Needham-Schroeder protocol. In our simulation, we use RS (5, 4) (n=5, k=4) codes where we generate 1 parity packet for 4 source packets. The computational energy cost of RS code was evaluated using IT++ library [75].

Test programs for individual computational operations were run on an Intel i3 processor and the corresponding number of processor cycles for each was retrieved. In order to be able to induce the energy cost on a resource-constrained device from the number of cycles on a powerful processor, we disabled advanced features on our test processor (hyperthreading, multi-core, variable clock speed). Eventually we were able to consider that the energy cost for a sensor (\(E_{\text{TelosB}}\), expressed in Joules) can be derived from the number of cycles measured on the i3 (\(C_{\text{i3}}\), using the following equation:

\[
E_{\text{TelosB}} = \frac{U_{\text{TelosB}} \cdot I_{\text{TelosB}}}{N_{\text{TelosB}}} \times \frac{\text{Register\_size}_{\text{i3}}}{\text{Register\_size}_{\text{TelosB}}} \times \alpha \times C_{\text{i3}}
\]

(11)

Where \(\alpha\) is a coefficient representing the richer instructions of the i3 and approximated to 2 in our analysis.

Computational cost results for distributed Needham-Schroeder (representative of a two-pass key transport protocol) and distributed HIP (representative of a key agreement protocol) exchanges are respectively presented in the Tables 4 and 5 below.
6.3.3 Communication cost

In this subsection we assess the communication energy costs of the proposed distributed approaches at the constrained initiator. These costs are made of the costs of transmission, reception and listening.

The energy consumption of a node in listening mode can be equivalent to its consumption in reception mode since the transceiver remains active in both modes (see Table 6). Authors in [76] assess the energy cost of cryptographic algorithms is WSNs nodes and reveal the impact of listening on the total energy cost. However they did not consider this element in their estimates. Reference [77] includes the listening cost to estimate the energy cost of ECDH-ECDSA and Kerberos protocols on TelosB and MICAz sensors and insists on
its importance comparing results with a prior work that estimates communication cost considering only transmission and reception costs. This comparison shows an energy overhead of 45% when the listening cost is taken into account.

We use the power consumptions presented in the Table 6 as an energy model of the different operating modes (transmit, receive and listen) for the TelosB platform [77]. As reported in [77] we consider an effective data rate of 75 kbps for a 250 kbps claimed one. This important

<table>
<thead>
<tr>
<th>Cryptographic operations</th>
<th>Energy cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic approach</td>
<td></td>
</tr>
<tr>
<td>Pk_encrypt_x+</td>
<td>1.6 mJ</td>
</tr>
<tr>
<td>Pk_decrypt_(x,y)+</td>
<td>23.35 mJ</td>
</tr>
<tr>
<td>Pk_encrypt_y</td>
<td>1.6 mJ = 26.55 mJ</td>
</tr>
<tr>
<td>Distr. approach</td>
<td></td>
</tr>
<tr>
<td>n*(encrypt_xi +</td>
<td></td>
</tr>
<tr>
<td>compute_MAC +encrypt_yi+</td>
<td></td>
</tr>
<tr>
<td>compute_MAC)+</td>
<td></td>
</tr>
<tr>
<td>verify_MAC+</td>
<td></td>
</tr>
<tr>
<td>decrypt_(x_encrypt_y)+</td>
<td></td>
</tr>
<tr>
<td>decrypt_y</td>
<td></td>
</tr>
<tr>
<td>n* 4.94µJ + 0.28µJ + 24.7 µJ + 24.7 µJ = 350.6 µJ+</td>
<td></td>
</tr>
<tr>
<td>Threshold Distr. Approach</td>
<td></td>
</tr>
<tr>
<td>encode_reed_solomon+</td>
<td></td>
</tr>
<tr>
<td>n*(encrypt_xi +</td>
<td></td>
</tr>
<tr>
<td>compute_MAC) +</td>
<td></td>
</tr>
<tr>
<td>k*(encrypt_yi+</td>
<td></td>
</tr>
<tr>
<td>compute_MAC)+</td>
<td></td>
</tr>
<tr>
<td>verify_MAC+</td>
<td></td>
</tr>
<tr>
<td>decrypt_(x_encrypt_y)+</td>
<td></td>
</tr>
<tr>
<td>decrypt_y</td>
<td></td>
</tr>
<tr>
<td>n* 4.94µJ + 0.28µJ + 24.7 µJ + 24.7 µJ = 5<em>3</em>(0.09 µJ + 0.05 µJ) + 5<em>66.77µJ + 0.05 µJ + 24.7µJ + 4</em>(24.73µJ + 1.27 µJ) + 290 µJ + 2.1 µJ = 763.22 µJ</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Energy costs of cryptographic operations required by the different evaluated approaches on a TelosB processor for the Needham-Schroeder protocol.

<table>
<thead>
<tr>
<th>Cryptographic operations</th>
<th>Energy cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Approach</td>
<td></td>
</tr>
<tr>
<td>verify_signR1 +</td>
<td>1.24 mJ</td>
</tr>
<tr>
<td>compute_exp_ga +</td>
<td>58.97 mJ</td>
</tr>
<tr>
<td>sign{Soln, DHI, HII} +</td>
<td>24.55 mJ</td>
</tr>
<tr>
<td>verify_signR2 +</td>
<td>1.24 mJ</td>
</tr>
<tr>
<td>compute_exp_gab +</td>
<td>104.73 mJ</td>
</tr>
<tr>
<td>verify_MAC_KDH</td>
<td>2.1 µJ = 190.73 mJ</td>
</tr>
<tr>
<td>Integer Partition</td>
<td></td>
</tr>
<tr>
<td>n*(encryptI21i+</td>
<td></td>
</tr>
<tr>
<td>compute_MAC +</td>
<td></td>
</tr>
<tr>
<td>decryptR22i+</td>
<td></td>
</tr>
<tr>
<td>verify_MAC) +</td>
<td></td>
</tr>
<tr>
<td>compute_mult_gai+</td>
<td></td>
</tr>
<tr>
<td>verify_MAC_KDH</td>
<td></td>
</tr>
<tr>
<td>n* 66.77µJ + 2.47µJ + 24.73µJ + 1.27 µJ + 290 µJ + 2.1 µJ = 763.22 µJ</td>
<td></td>
</tr>
<tr>
<td>Threshold Distr. Approach</td>
<td></td>
</tr>
<tr>
<td>n*(k-1)*(comp_mult_f(i))+</td>
<td></td>
</tr>
<tr>
<td>compute_add_f(i) +</td>
<td></td>
</tr>
<tr>
<td>n*(encrI21i+comp_MAC) +</td>
<td></td>
</tr>
<tr>
<td>decryptR22i+</td>
<td></td>
</tr>
<tr>
<td>verify_MAC) +</td>
<td></td>
</tr>
<tr>
<td>compute_mult_gb(i) +</td>
<td></td>
</tr>
<tr>
<td>verify_MAC_KDH</td>
<td></td>
</tr>
<tr>
<td>n* 0.09 µJ + 0.05 µJ + 5*66.77µJ + 0.05 µJ + 24.7³µJ + 1.27 µJ + 290 µJ + 2.1 µJ = 744.48 µJ</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Energy costs of cryptographic operations required by the different evaluated approaches on a TelosB processor for the HIP BEX protocol.
decrease of the data rate is discussed in [78]. From the previous exchange descriptions, we obtain in the Table 7 below the number of exchanged bytes by the source node in Needham-Schroeder and HIP BEX protocols, considering both the basic exchange and the distributed approaches.

We consider that the constrained node is listening during a delay corresponding to the latency of communications (Tx, Rx) and packets propagation ($\Delta$) as well as the processing of packets at the proxies and the responder. We estimate below the listening durations required by the constrained node in the considered approaches:

with $\text{Proc}(A)$, $T(A)$, $R(A)$ being respectively the processing, transmission and reception durations at node $A$ and $\Delta(A \rightarrow B)$ being the packet delivery latency from node $A$ to node $B$. Assuming that the server is an unconstrained node while proxies are 10 times less constrained than the server, this duration is respectively 401 ms and 403 ms for Needham-

Table 6: Power consumption of TelosB at 4 MHz with a transmit power of 5 dBm (from [77]).

<table>
<thead>
<tr>
<th></th>
<th>Transmit</th>
<th>Receive</th>
<th>Listen</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TelosB platform</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmit</td>
<td>54 mW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive</td>
<td>61 mW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Listen</td>
<td>60 mW</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Sent and received bytes in the Needham-Schroeder and HIP BEX protocols.

<table>
<thead>
<tr>
<th></th>
<th>Basic approach</th>
<th>Distributed approach</th>
<th>Threshold distributed approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Needham-Schroeder protocol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sent (bytes)</td>
<td>256</td>
<td>360</td>
<td>324</td>
</tr>
<tr>
<td>Recv (bytes)</td>
<td>128</td>
<td>420</td>
<td>336</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Basic approach</th>
<th>Distributed approach</th>
<th>Threshold distributed approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIP BEX protocol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sent (bytes)</td>
<td>608</td>
<td>2272</td>
<td>2272</td>
</tr>
<tr>
<td>Recv (bytes)</td>
<td>468</td>
<td>1308</td>
<td>1047</td>
</tr>
</tbody>
</table>

Table 8: Communication Energy costs on a TelosB processor for the Needham-Schroeder protocol.

<table>
<thead>
<tr>
<th></th>
<th>Basic Approach</th>
<th>Distributed Approach</th>
<th>Threshold Distributed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit cost</td>
<td>1.47 mJ</td>
<td>2.07 mJ</td>
<td>1.86 mJ</td>
</tr>
<tr>
<td>Receive cost</td>
<td>0.83 mJ</td>
<td>2.72 mJ</td>
<td>2.17 mJ</td>
</tr>
<tr>
<td>Listen cost</td>
<td>23.84 mJ</td>
<td>24.66 mJ</td>
<td>24.66 mJ</td>
</tr>
<tr>
<td>Energy cost</td>
<td>26.14 mJ</td>
<td>29.45 mJ</td>
<td>28.69 mJ</td>
</tr>
</tbody>
</table>
Schroeder basic exchange and HIP BEX while it respectively amounts to 411 ms and 453 ms for the distributed Needham-Schroeder and HIP BEX approaches. We also assume that the proxy is one hop away from the initiator and that a 200 ms propagation delay is required to route packets from the source to the server. Finally, the energy costs induced by communications in both basic approach and distributed approaches are shown in Table 8 (Needham-Schroeder) and Table 9 (HIP BEX).

<table>
<thead>
<tr>
<th></th>
<th>Basic Approach</th>
<th>Integer Partition</th>
<th>Threshold Distributed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit cost</td>
<td>3.50 mJ</td>
<td>13.09 mJ</td>
<td>13.09 mJ</td>
</tr>
<tr>
<td>Receive cost</td>
<td>3.03 mJ</td>
<td>8.47 mJ</td>
<td>6.78 mJ</td>
</tr>
<tr>
<td>Listen cost</td>
<td>24.18 mJ</td>
<td>26.58 mJ</td>
<td>26.58 mJ</td>
</tr>
<tr>
<td>Energy cost</td>
<td>30.71 mJ</td>
<td>48.14 mJ</td>
<td>46.45 mJ</td>
</tr>
</tbody>
</table>

Table 9: Communication Energy costs on a TelosB processor for the HIP BEX protocol.

<table>
<thead>
<tr>
<th></th>
<th>Basic Approach</th>
<th>Distributed Approach</th>
<th>Threshold Distributed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp.</td>
<td>26.55 mJ</td>
<td>0.10 mJ</td>
<td>0.44 mJ</td>
</tr>
<tr>
<td>Comm.</td>
<td>26.14 mJ</td>
<td>29.45 mJ</td>
<td>28.69 mJ</td>
</tr>
<tr>
<td>Total energy cost</td>
<td>52.69 mJ</td>
<td>29.55 mJ</td>
<td>29.13 mJ</td>
</tr>
</tbody>
</table>

Table 10: Compared total (computations + communications) energy costs on a TelosB processor for the Needham-Schroeder protocol (basic approach), compared with its distributed approaches.

<table>
<thead>
<tr>
<th></th>
<th>Basic Approach</th>
<th>Integer Partition</th>
<th>Threshold Distributed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp.</td>
<td>190.73 mJ</td>
<td>0.76 mJ</td>
<td>0.74 mJ</td>
</tr>
<tr>
<td>Comm.</td>
<td>30.71 mJ</td>
<td>48.14 mJ</td>
<td>46.45 mJ</td>
</tr>
<tr>
<td>Total energy cost</td>
<td>221.44 mJ</td>
<td>48.9 mJ</td>
<td>47.19 mJ</td>
</tr>
</tbody>
</table>

Table 11: Compared total (computations + communications) energy costs on a TelosB processor for the HIP BEX protocol (basic approach), compared with its distributed approaches.
6.3.4 Total energy cost

Gathering the computation and communication costs, we provide the total energy costs of the two examples of key exchange protocols considering the basic and collaborative approaches in Tables 10 and 11.

As shown in Tables 10 and 11, our estimated costs confirm the efficiency of the cooperative scheme we propose. The energy savings at the source node amount to respectively 45% and 75% of what is consumed by the Needham-Schroeder and the HIP BEX protocols. Energy savings can be increased by reducing the duration of the listening mode. Using LPL (Low Power Listening) protocols [79], the source node can be temporarily put into a sleep mode when waiting for the protocol run between proxies and server.

The results also show that the energy costs of the threshold distributed approaches in the two examples of protocols are almost similar to those of the simple distributed approaches; contrary to what may have been expected if one had only considered the overhead introduced by the addition of redundant parity packets in the threshold distributed Needham-Schroeder approach, or the generation of the polynomial shares in the threshold distributed HIP BEX approach. This overhead certainly makes the secret distribution more complex, but meanwhile it reduces the energy cost of messages processing at the source node, which receives and deciphers $k$ packets instead of $n$.

In a nutshell, simulation results prove the viability of the proposed threshold distributed approach in the studied context of IoT keying, which involves highly resource-constrained nodes such as the TelosB platform. Although being computationally more complex than the simple distributed approach, the threshold distributed approach performs better when considered globally. Furthermore, it introduces the important recovery and secrecy properties, essential for a collaborative protocol such as the one we propose.

6.4 Conclusion

This chapter presented a novel collaborative approach for key establishment in the context of the IoT, by which a resource-constrained device delegates its expensive computational load to assisting nodes, on a distributed and cooperative basis. In order to enable this collaborative behavior, two distributed techniques have been proposed and carefully designed for both the key transport and key agreement modes. These techniques have then been assessed and compared to basic key exchange standards from the points of view of cryptographic and communication costs. Simulations results first show that our proxy-based scheme significantly increases the energy savings at the constrained device compared to existing standards. They also show that the threshold distribution should be the preferred approach for a constrained node taking part to the collaborative process we propose for key exchange.
7 Back Pressure Congestion Control for IoT

In this section of the deliverable we address the design of network architectures for the Internet of Things by proposing practical algorithms to augment IETF CoAP/6LoWPAN protocol stacks with congestion control functionalities. Our design is inspired by previous theoretical work on back pressure routing and is targeted toward Web-based architectures featuring bidirectional data flows made up of CoAP request/response pairs. Here, we present three different cross-layer and fully decentralized congestion control schemes and compare them against ideal back pressure and current UDP-based protocol stacks. Hence, we discuss results obtained using ns-3 through an extensive simulation campaign for two different scenarios: unidirectional and upstream flows and bidirectional Web-based CoAP flows. Our results confirm that the proposed congestion control algorithms perform satisfactorily in both scenarios for a wide range of values of their configuration parameters, and are amenable to the implementation onto existing protocol stacks for embedded sensor devices.

7.1 Distributed Congestion Control for Distributed IoT Networks

In the last few years, we have witnessed considerable advances in terms of protocol design for wireless sensor networking. These have led to a solid understanding of the problems related to channel access, routing and data gathering, delivering efficient protocol stacks and ultimately spurring the standardization of protocols for data collection and addressing. The work in this section considers network protocols recently standardized by IETF, namely, CoAP [80] and 6LoWPAN [28], whose combined use permits Web-based bi-directional communications between sensor devices and Internet servers. 6LoWPAN provides header compression and specifies communication profiles that allow the implementation of IPv6 addressing. CoAP is a stateless protocol that is aimed at replacing HTTP for lightweight and resource constrained devices. As such, it implements a reduced set of functionalities with respect to HTTP. While CoAP and 6LoWPAN provide the basis for Web-oriented protocol stacks for embedded devices and natively support UDP traffic, they do not fully address the congestion problem, and only provide some conservative recommendations, as we discuss below in Section 7.6.

The Internet protocol suite, i.e., TCP/IP, has been designed adopting the “end-to-end argument” [81], which has proven to be very effective in networks of smart terminals operating bulk data transfers. However, TCP congestion control (CC) [82] has been designed with an implicit assumption: data transfers causing congestion are usually long enough to be efficiently controlled through end-to-end CC algorithms. By their own nature, slow start and congestion avoidance are techniques that converge after some time and after a potentially large amount of data has been transferred. However, when the amount of data required to create congestion on the network is very small, these techniques do not provide an efficient solution to the CC problem. In addition to this, TCP is known to be severely impacted by the long delays that are typical of constrained networks.

Here, following the research lines identified in [83], we develop practical congestion control algorithms for constrained Internet of Things (IoT) exploiting 6LoWPAN technology. These networks are characterized by very constrained processing, memory and communication
capabilities [84], a potentially large number of nodes, and infrequent communication patterns which very much differ from standard Internet flows.

Our present work quantifies the benefits of implementing congestion control at layer 3 by exploiting practical and lightweight algorithms based on the concept of back pressure routing by Tassiulas and Ephremides [85]. Since its conception, Back Pressure (BP) policies have been extensively explored, leading to distributed theoretical algorithms that achieve optimal throughput performance in distributed networks. Practical applications of these schemes have also been studied in several papers such as [86], which applies a similar policy to the queues of wireless sensor nodes to realize an efficient data collection protocol. However, that solution makes strong use of channel snooping and poses limitations on the implementation of radio duty-cycling (RDC). In [87] the authors propose CODA, a distributed algorithm that uses explicit messages to detect congestion and therefore can also work in the presence of RDC. An evaluation of CODA in 6LoWPAN networks can be found in [88], where the authors measure the loss probability and the number of delivered packets. While these studies on CODA are of interest for the application of congestion control principles in energy efficient networks, some practical issues still remain open, namely: i) the explicit BP messages are not provided in standard existing protocols, and therefore cannot be used in standard networking stacks, and ii) there is no discussion on some important issues such as the effect of the required number of hop-by-hop retransmissions and of bidirectional CoAP traffic support.

Our main objective is to systematically compare through detailed simulations different lightweight BP approaches, including existing as well as original algorithms, in order to assess their suitability for the implementation into IoT devices and their benefits in terms of performance gains.

The main contributions of our research are the following:

- We propose a number of practical and lightweight congestion control algorithms for constrained devices, devising CC policies based on distributed back pressure control, with the objectives of detecting and alleviating network congestion, providing reliability and ultimately controlling the injection of data traffic into the network.

- We present extensive simulation results by comparing the performance of the proposed CC policies with that of ideal back pressure algorithms and showing that layer 3 BP congestion control is feasible on constrained IoT devices, and results in significant performance gains at the expense of minimal additional complexity.

- We present protocols and results for unidirectional and upstream data traffic as well as for bidirectional CoAP flows.

We organize the presentation of our technical and scientific achievements as follows. In Section 7.2 we describe the system model and present our practical BP-based congestion control algorithms for constrained devices. First, in Section 7.5 we evaluate the performance of these algorithms focusing on unidirectional and upstream data collection. Then, in Section 7.6 we consider bidirectional communication scenarios such as those arising from CoAP-based Web-services. Finally, we draw our conclusions in Section 7.8.
7.2 Back Pressure Congestion Control for 6LoWPAN

In the following, we present some CC designs that are explicitly tailored to constrained networks featuring infrequent communication patterns. Specifically, we propose to perform congestion control actions at the network layer, as this allows the implementation of BP algorithms that work on aggregates of datagrams, i.e., on IP queues. Note that working on aggregates is desirable due to the nature of the traffic found in 6LoWPAN networks, which usually reaches considerable volumes only when the output of multiple nodes is combined. Moreover, this results in a lower complexity in terms of software structure, memory utilization and communication requirements for the control of network queues.

7.3 Node Model

Each node has been modeled according to the Internet Host model [89], which classifies protocols into Link, Network, Transport and Application layers.

7.3.1 Link

6LoWPAN has been specifically designed for the IEEE 802.15.4 PHY/MAC [90]. Thus, in our model each node is equipped with an IEEE 802.15.4 radio transceiver operating at 2.4 GHz with a nominal available transmission rate of 250 kb/s. Layer 2 operates according to the IEEE 802.15.4 standard and the total number of transmissions per packet is limited to a maximum of 7.

7.3.2 Network

IPv6 and 6LoWPAN belong to this category; our node has been equipped with a standard layer 3 device (L3D) operating as follows. For each IP datagram, received either from the applications residing in the upper layers or from the radio, L3D first understands whether this datagram has to be delivered to the local host. As a second step, L3D looks in the Internet routing table, extracts the next hop toward which the datagram has to be sent, and places the received datagram into the layer 3 queue for outbound traffic. This queue is managed according to a First-In First-Out Drop Tail (FIFO-DT) discipline. Note that we account for a single IP queue at layer 3, which is a realistic limitation and is typical of constrained devices. Our L3Ds implement hop-by-hop layer 3 retransmissions and different BP control algorithms, as specified in Sections 7.4 and 7.6.1.

7.3.3 Transport

The UDP transport protocol is adopted. UDP only performs a checksum check for every received datagram, without any further processing or buffering operations.
7.3.4 Application

We have considered two usage scenarios:

S1) **Unidirectional flows (Section 7.5):** for the study of unidirectional upstream data traffic, we have adopted the Iperf [91] protocol. It permits to evaluate at the receiver the number of lost packets, the number of out-of-order deliveries, the multi-hop delay and the per-packet jitter. Data sources emit UDP traffic at a constant bit rate (CBR), except for the cases where the local layer 3 queue is full. In these cases, the emission of the datagram is delayed until a layer 3 queue slot becomes available.

S2) **Bidirectional flows (Section 7.6):** to evaluate the effectiveness of the proposed congestion control algorithms for bidirectional traffic, we have used the Constrained Application Protocol (CoAP) [80] to transport Iperf messages. CoAP implements a lightweight bidirectional exchange targeted to client-server architectures. In this case clients are placed outside the constrained IoT domain and emit CoAP requests at a constant bit rate. These requests are sent to a border gateway and from there to the IoT nodes. Upon receiving these requests, IoT nodes reply with CoAP responses that flow in the opposite direction.

7.4 Layer 3 Device Types

We advocate the implementation of congestion control through the use of practical back pressure techniques, which are embedded into the layer 3 device of each sensor node. Next, these augmented L3Ds are presented in detail, whereas their performance evaluation is carried out in Sections 7.5 and 7.6, where we respectively look at unidirectional and bidirectional flows.

7.4.1 Static

This is the L3D described in Section 7.3, which does not account for any congestion control mechanism. It is a baseline scheme considered here to gauge the advantages offered by the following BP schemes.

7.4.2 IdealBP

This L3D refrains from transmitting as long as the queue length at the next hop is higher than that of the local queue. This behavior mimics the ideal BP policies devised by Tassiulas and Ephremides [85]. Note that in actual implementations nodes can only know the queue length at the next hop through the exchange of proper control signals. For IdealBP, in our simulations this information is made available to any node through a genie. Although IdealBP is impractical, we have considered it here to validate the BP approach and also see how much its performance deviates from that of the practical algorithms that we propose next.
According to IdealBP’s BP policy, the datagram at the current node is transmitted to the next hop whenever their queue differential is positive and the remote queue length at layer 3 is smaller than a pre-determined threshold $Q_{thr} > 0$. This threshold is required because it could happen that multiple devices concurrently send their datagrams (one per device) to the same next hop. In this case, the queue at the next hop could overflow even though the preceding queue differential was positive.

### 7.4.3 Griping

This device uses an explicit BP signal on congestion and is similar to the CODA BP policy [87], that has also been evaluated in [88]. Differently from [87] and [88], in Griping subsequent BP control messages must be transmitted at least $K$ seconds apart, where $K$ is a tunable parameter. In fact, we noticed that close transmissions of BP control messages toward the same source lead to inefficiencies in terms of transmission overhead. Griping works as follows.

- **Receiver**: whenever a Griping L3D receiver gets a new layer 3 datagram and its layer 3 queue length $Q$ is larger than a threshold $Q_{thr} > 0$, it transmits a unicast BP control message back to the source of that datagram. Subsequent BP control messages transmitted by the same source must be spaced by at least $K$ seconds. The transmission of a new BP is canceled when this condition is not met.

- **Transmitter**: at any time, each Griping transmitter sends its own datagrams at a rate that is updated according to an Additive Increase and Multiplicative Decrease (AIMD) approach. Specifically, the Griping transmitter works considering reference “time slots” of $T$ seconds each. Within each time slot it can transmit up to $W$ datagrams. Every time the Griping transmitter receives a BP message, $W$ is halved, thus effectively halving the transmission rate. If no BP message is received in $T$ seconds, then $W$ is increased by one unit. In our simulations, the parameters $K$ and $T$ have been tuned and subsequently set to 100 ms and 750 ms, respectively.

**Remarks.** Due to its simplicity, Griping is amenable to the implementation on constrained nodes. Moreover, we note that this technique does not require any interaction with the PHY and MAC layers and therefore does not rely on their specific implementation. This makes it possible to implement Griping with radio duty cycling, which is a critical feature for wireless sensor networks.

Layer 3 losses in Griping occur in two cases.

- **C1) Receiver side**: a packet is correctly received at layer 2 and is passed to layer 3, where the network queue is full. The packet is thus discarded and a layer 3 queue overflow occurs.

- **C2) Sender side**: a packet is discarded when none of the allowed retransmissions at layer 3 has led to its successful reception.

---

7Note that the concurrent transmissions occur at layer 3; lower layers will multiplex these transmissions so as to avoid collisions, by possibly retransmitting collided packets.
7.4.4 Deaf

This is an alternative approach that aims at removing the complexity associated with the transmission of BP control messages. *Deaf* is implemented as follows.

- **Receiver:** a *Deaf* receiver stops sending layer 2 acknowledgements whenever the layer 3 queue length $Q$ is larger than a predefined threshold $Q_{thr} > 0$. The stopped acknowledgment flow is perceived by the *Deaf* transmitter as an implicit BP notification.

- **Transmitter:** a *Deaf* transmitter handles layer 3 retransmissions as follows. After a new layer 3 datagram transmission, a back off timer is initialized for this datagram to $T_{wait}$ seconds, where $T_{wait}$ is drawn from a random variable uniformly distributed in $[0, 2^n T]$, where $T$ is the duration of a layer 3 time slot and $n$ is a function of the total number of transmissions performed for this datagram. Thus, the retransmission of this datagram at layer 3 will occur no earlier than $T_{wait}$ seconds from its current transmission attempt. $n$ is updated as follows: after each retransmission of the datagram, a transmission counter, $n_{tx}$, is incremented by one and the length of the previous interval for $T_{wait}$ is adapted by picking a new value of $n$ as: $n = \min(n_{tx}, n_{tx}^{max})$, with $n_{tx}^{max} = 4$. $T$ has been set to 0.1 s and for the first transmission of the datagram $n_{tx}$ is initialized to zero. Note that $T_{wait}$ is doubled at each failure of the same datagram, which is equivalent to halving the associated transmission rate. The underlying layer 2 notifies layer 3 about the transmission status (either failed or successful) of every upper layer datagram. We say that a layer 3 datagram failure event occurs whenever the maximum number of retransmissions is reached for this packet, which is still unsuccessfully delivered.

**Remarks.** Note that this technique does require some cross-layer interaction between layer 3 and layer 2. In fact, a given layer 2 frame is not acknowledged by *Deaf* whenever the overlying layer 3 communicates to the lower layer a failure for that packet. However, this does not require the processing of further PHY- or MAC-layer metrics. For this reason, we argue that *Deaf* does not interfere with radio duty-cycling as others cross-layer approaches usually do. Note that congestion control in this case is enforced by spacing apart the retransmissions of the same packet at layer 3. This amounts to decreasing the actual layer 3 transmission rate through implicit notifications from the *Deaf* receiver. Also, there is a semantic violation in that retransmissions are performed at layer 3, which is in contrast with standard congestion control that is implemented in the transport protocol at layer 4. Also, *Deaf* never acknowledges layer 2 packets whenever the layer 3 queue is full. Thus, event C1 above never occurs and packets can only be discarded due to event C2.

7.4.5 Fuse

*Fuse* combines the BP actions of *Griping* and *Deaf*. As we shall see below, this combined action effectively reaps the benefits of both *Deaf* and *Griping* BP policies. Specifically:

- **Receiver:** the *Fuse* receiver behaves as *Griping* as long as its queue length is smaller than the maximum queue size $Q_{max}$. When the layer 3 queue is full, *Fuse* combines the BP actions of *Griping* and *Deaf*, i.e., it stops sending layer 2 acknowledgments, as...
Deaf does, but also continues to send explicit congestion notification messages, as Griping does.

- Transmitter: the Fuse transmitter implements the AIMD rate control policy, as discussed above for the Griping transmitter.

7.5 Unidirectional Upstream Data Traffic

In this section we analyze the performance of the back pressure algorithms of Section 7.4 when applied to 6LoWPAN networks adopting RPL [92] and transmitting data packets over unidirectional and upstream flows, i.e., from some source sensor nodes to the border router which interconnects the constrained sensor network to the unconstrained Internet, see the first case study of Fig. 25.

7.5.1 System Parameters

The following parameters have been chosen to evaluate their impact on the performance.

- **Offered Traffic Load** $\lambda_{tx}$ defines the rate at which each source emits UDP datagrams at layer 3 and is measured in packets per second.

- **Number of retransmissions** $N_{\text{retx}}$ controls the maximum number of retransmissions. Specifically, when $N_{\text{retx}} = 0$ retransmissions are disabled at both layer 2 and layer 3.
When $N_{\text{retx}} = 1$, retransmissions are disabled at layer 3, whereas a maximum of $N_{L2_{\text{retx}}} = 7$ transmissions per packet is possible at layer 2. When $N_{\text{retx}} > 1$, the maximum number of allowed layer 2 transmissions is set to 7, and the maximum number of layer 3 retransmissions is set to $N_{\text{retx}} - 1$.

- **Maximum queue length** $Q_{\text{max}}$ defines the maximum available memory for the layer 3 queue, which is the same for every node and is expressed in terms of number of layer 3 packets.

- **Queue threshold** $Q_{\text{thr}}$ is the threshold on the queue length at the L3D receiver that is used to assess when a control action is required by the BP algorithms, as detailed in Section 7.4.

### 7.5.2 Performance Metrics

To compare the performance of the various L3Ds, the following metrics have been considered.

- **Reception rate** $\lambda_{\text{rx}}$ defines the average rate at which layer 3 packets are correctly received by the destination, and is measured in packets per second.

- **Multihop delay** $D$ refers to the time taken by a packet to be correctly received by the border router (BR, see Fig. 25) from its transmission instant at the source (one of the 9 leaf nodes of the routing tree of Fig. 25). $D$ is obtained averaging the packet delay over all packets that are correctly received by the border router.

- **Loss probability** $P_{\text{loss}}$ represents the probability that an emitted layer 3 datagram is lost either due to buffer overrun or because the maximum number of retransmissions

![Figure 26: Reception rate $\lambda_{\text{rx}}$ and multihop delay $D$ vs. the offered traffic load $\lambda_{\text{tx}}$.](image)
has been reached. $P_{\text{loss}}$ is computed as the ratio of the total number of datagrams lost in the network to the total number of datagrams emitted by all sources.

- **Rejection rate** $R$ defines the average rate at which packets from the application are rejected by the network layer due to a full layer 3 queue. In this case, application layer packets are not lost at layer 3 but their insertion into the layer 3 queue is denied and an error message is sent to the application layer. Upon receiving this error message, the application slows down its transmission rate, temporarily stopping its transmission flow and resuming it whenever new buffer space becomes available at layer 3.

- **Transmission Overhead** ($N_{\text{tx}}$) Represents the average total number of layer 2 packets that are transmitted in the network for the successful end-to-end (from a leaf node to the border router) delivery of a single layer 3 datagram. This metric accounts for the number of layer 2 packets that are sent to carry layer 3 data messages as well as layer 3 BP control messages, such as those sent by Griping and Fuse for the explicit signaling of a congestion event.

### 7.5.3 Results for Upstream Unidirectional Traffic

Simulations have been run in ns-3 [93], using IEEE 802.15.4 at the PHY and MAC layers. At layer 2 packets have a fixed size of 127 bytes, including 12 bytes for the layer 2 headers. Layer 3 datagrams have a fixed size of 115 bytes, which means that a layer 3 datagram fits into a layer 2 packet and fragmentation is not needed. A tree topology has been built, see Fig. 25, containing 9 leaf nodes, 4 routers (R) and a border router (BR). This topology is representative of a typical routing scenario for 6LoWPAN networks adopting RPL [92], a recently standardized routing protocol for low-power lossy networks.

In the following results, errors due to wireless transmissions have not been considered as neglecting channel impairments makes it possible to isolate and characterize the effects that are solely due to the considered congestion control algorithms, which is the main purpose of our present study.

The simulation duration is set to 200 seconds, and $Q_{\text{max}}$ is set to 31 packets to resemble typical limitations of IPv6/6LoWPAN stacks on constrained hardware platforms (see [94]). Each of the points in the following graphs is obtained averaging 180 independent simulation runs.

### 7.5.4 Impact of varying the transmission rate $\lambda_{\text{tx}}$

In Fig. 26 we show $\lambda_{\text{rx}}$ and $D$ for each L3D device as a function of $\lambda_{\text{tx}} \in [1, 30]$ pkt/s. The remaining system parameters have been set to: $N_{\text{retx}} = 15$, $Q_{\text{max}} = 31$ packets, and $Q_{\text{thr}} = 15$ packets.

Considering the Static device, as long as the input rate $\lambda_{\text{tx}}$ remains smaller than a certain saturation threshold $\lambda_{\text{sat}}$ (of about 9 pkt/s in Fig. 26), we have that the reception rate $\lambda_{\text{rx}}^{\text{Static}}$ is approximately equal to $\lambda_{\text{tx}}^{\text{Static}}$ and $D^{\text{Static}}$ is stable and small. This means that the network can effectively serve the injected data traffic. Here, the packet delay is typically dominated by the transmission and propagation delays over the multi-hop paths from the sources to the BR,
whereas the queueing delay is negligible. As $\lambda^{\text{Static}}_{tx}$ grows larger than $\lambda^{\text{sat}}_{\text{sat}}$, $\lambda^{\text{Static}}_{rx}$ saturates reaching the so called saturation throughput. At this point, $D^{\text{Static}}$ grows abruptly and this is due to the queueing component of the delay, that considerably increases as $\lambda_{tx}$ becomes higher than the actual layer 2 service rate. Similar performance tradeoffs are observed for all L3Ds.

While there are no substantial differences between Static and the other L3Ds in terms of $\lambda_{rx}$, we note that all the other devices obtain an average delay $D$ increased by a factor of about 4 during congestion if compared with Static. This is due to the fact that these devices put off the transmission of new layer 3 packets when the network is congested, whereas Static keeps transmitting at a fixed rate, irrespective of the congestion status of the network. Also, Griping and Fuse account for the longest delay, and this is due to their explicit transmission of back pressure messages. From this first figure we observe that back pressure tends to increase the delay but is able to retain most of the throughput performance of the greedy Static transmission policy.

The rejection rate $R$ has been plotted in Fig. 27 for the same simulation parameters. For BP devices, flow congestion actions are taken as soon as $\lambda_{tx}$ becomes equal to $\lambda^{\text{sat}}_{\text{sat}}$ and are enforced as long as $\lambda_{tx} \geq \lambda^{\text{sat}}_{\text{sat}}$. These actions correspond to increasing the layer 3 rejection rate $R$. We note that IdealBP has the lowest rejection rate and the highest reception rate among all BP schemes, and thus, as expected, it is the best performing algorithm, i.e., the one able to fully exploit the benefits of back pressure.

$R$ of Deaf, Griping and Fuse is very similar and close to that of IdealBP. Moreover, their back pressure policy becomes effective when $\lambda_{tx} \geq \lambda^{\text{sat}}_{\text{sat}}$, which is testified by the prompt increase in $R$ when the network operates beyond the saturation point.

Static keeps sending packets at the maximum possible rate, irrespective of the queue status at the relays. This moves to the right the value of $\lambda$ for which layer 3 queues are filled up and packets start to be rejected (the increase of $R$ becomes apparent for $\lambda_{tx} \geq 20$ pkt/s in Fig. 27). However, as we shall see below the drawback of this aggressive transmission behavior is that layer 3 queues are subject to higher loss rates.

As we show shortly, Griping, Deaf and Fuse have a substantially smaller $P_{\text{loss}}$ than Static as they reject only the data traffic that the network cannot sustain, mimicking IdealBP’s behavior. Note that layer 3 rejection does not imply discarding packets but rather slowing down the packet generation rate at the application.\footnote{Rejecting traffic that cannot be successfully handled by the network results in improved performance for all users. This can be supported with minimal impact by those applications featuring elastic data traffic. Otherwise, the application will see some degraded performance.}

In Fig. 28 we show the transmission overhead $N_{tx}$ as a function of the offered traffic load $\lambda_{tx}$. As expected, IdealBP has the best performance among all schemes as it applies BP control by leveraging at no cost the exact and instantaneous knowledge of all network queues, which is provided in the simulations through a genie. As $\lambda_{tx}$ increases beyond $\lambda^{\text{sat}}_{\text{sat}}$, all the remaining schemes show a degraded performance in terms of $N_{tx}$. Deaf is the scheme that leads to the highest transmission overhead and this is inherent in its design, as this scheme tends to hit the maximum number of retransmission attempts while handling congestion control. Static is the second-worst as in this case congestion is emphasized through the careless injection of data traffic. Griping and Fuse both perform very close to IdealBP as the corresponding
BP policies explicitly send congestion notifications to the senders and this has the effect of timely slowing down the volume of data that is injected into the network, alleviating the congestion.

### 7.5.5 Impact of varying $N_{\text{retx}}$

Fig. 29 shows the loss probability $P_{\text{loss}}$ as a function of $N_{\text{retx}}$. The remaining system parameters have been set to: $\lambda_{\text{tx}} = 20 \text{ pkt/s}$, $Q_{\text{max}} = 31 \text{ packets}$, and $Q_{\text{thr}} = 15 \text{ packets}$. Note that a transmission rate $\lambda_{\text{tx}} > \lambda_{\text{sat}}$ has been chosen so as to measure the ability of the different L3Ds to handle network congestion.

From Fig. 29 we observe the expected result that $P_{\text{loss}}$ generally decreases as $N_{\text{retx}}$ grows. This decrease is faster for Griping, Fuse and IdealBP as these algorithms use explicit signaling to detect congestion. The initial $P_{\text{loss}}$ decrease is slower for Deaf which therefore shows worse $P_{\text{loss}}$ performance for small values of $N_{\text{retx}}$, say, $N_{\text{retx}} \leq 7$. As expected, Static has the worst reliability performance as retransmissions are disabled for this scheme.

Also, Griping has a floor at $P_{\text{loss}} \simeq 0.02$, which is due to the inherent delay incurred in the explicit BP notification from the relay nodes. In fact, between the instant when a BP message is issued by a relay node and the instant when the controller at the corresponding source node enforces some back pressure action, the transmission rate remains equal to the one that has caused the congestion and, in turn, layer 3 losses are possible at the receiver node due to the overflow of its buffer. Thus, a vulnerable period exists between the instant when congestion is detected at the relays (that is, when their queue size increases beyond $Q_{\text{thr}}$) and the instant when the layer 3 flow is effectively slowed down at the sources. During this vulnerable period, losses due to buffer overflow are likely to occur. For Deaf, losses are still present due to the exhaustion of the overall number of retransmissions per packet per hop (at both layer 2 and layer 3) and for this reason its $P_{\text{loss}}$ monotonically decreases with an increasing $N_{\text{retx}}$. 

![Figure 27: Rejection rate $R$ vs. the offered traffic load $\lambda_{\text{tx}}$.](image)
Fuse has the best $P_{\text{loss}}$ performance and the reason for this is the combined effect of Griping and Deaf. In particular, the explicit signaling of Griping allows for a prompter reaction to congestion events, which substantially decreases the probability that Fuse reaches the maximum number of retransmissions. Moreover, the vulnerable period issue is solved as, whenever the receiver’s queue is filled up, Deaf’s BP control is invoked and packets that overflow from this queue are subsequently retransmitted by the corresponding sender (due to the stopped acknowledgement flow).

For what concerns previously shown performance metrics, all of them stabilize for small values of $N_{\text{retx}}$ to the values shown in Figs. 26, 27, and 28.

Furthermore, when $N_{\text{retx}} = 0$, network congestion goes undetected and back pressure algo-
rithms are never activated. In fact, in this case packets are transmitted but never retained in local queues, which are therefore filled up at a much slower pace. Thus, increasing $N_{\text{retx}}$ allows the fill-up of layer 3 queues and, in turn, the detection of congestion events: in fact, the rejection rate $R$ is positive for $N_{\text{retx}} > 0$. $N_{\text{retx}} = 0$ leads to poor performance on all metrics for all BP schemes.

Even $N_{\text{retx}} = 1$ leads to substantial throughput improvements in terms of $\lambda_{\text{rx}}$. Setting $N_{\text{retx}} = 2$, which means up to 7 layer 2 and just 1 layer 3 retransmission, grants a throughput that is very close to the maximum achievable for the given network setup. The throughput increase is always accompanied by a corresponding increase in the delay performance, which also stabilizes for small values of $N_{\text{retx}}$. Counterintuitively, $N_{\text{tx}}$ remains stable for all BP devices when $N_{\text{retx}} > 1$.

### 7.5.6 Impact of varying $Q_{\text{thr}}$

Fig. 30 shows the impact of $Q_{\text{thr}}$ for $N_{\text{retx}} = 15$, $\lambda_{\text{tx}} = 20$ pkt/s, $Q_{\max} = 31$ packets. Here, we only show the plot for $N_{\text{tx}}$, the other metrics are just discussed as their behavior is similar to what observed above. *Static* is represented as a horizontal line in the plot, since its behavior does not depend upon $Q_{\text{thr}}$.

As expected, $\lambda_{\text{rx}}$ grows for increasing $Q_{\text{thr}}$, as a larger threshold lowers the probability of having buffer under-runs, thus leading to higher throughput efficiencies. For the average delay $D$, an increasing $Q_{\text{thr}}$ puts off the enforcement of back pressure control actions. Correspondingly, the number of packets stored in layer 3 queues and their average delay both increase. On the other hand, very low values of $Q_{\text{thr}}$ lead to long delays too as in this case back pressure control is almost always active, i.e., transmission rates are often slowed down and this implies longer L3D service times.

As expected, $R$ decreases monotonically with $Q_{\text{thr}}$ for all BP schemes as the rate of back pressure control actions is lowered for increasing values of $Q_{\text{thr}}$. 

Figure 30: Transmission overhead $N_{\text{tx}}$ vs. the queue threshold $Q_{\text{thr}}$. 

Date: June 11, 2013

Version: Final
The behavior of $P_{\text{loss}}$ differs among the considered layer 3 devices. These results are just commented but not plotted for the sake of space. $\text{IdealBP}$ shows no losses at all as its control action is deterministic and immediate. Lowering $Q_{\text{thr}}$ for $\text{Griping}$ implies an earlier enforcement of back pressure policies, which leads to a larger buffer space to compensate for incoming packets during its vulnerable period. Thus, an increasing $Q_{\text{thr}}$ implies larger buffer overflow probabilities (i.e., larger $P_{\text{loss}}$). Conversely, for $\text{Deaf}$, $P_{\text{loss}}$ decreases with increasing $Q_{\text{thr}}$, ranging between 0.2% and 0.01%. This is because using the $\text{Deaf}$ device packet losses only occur whenever a source reaches the maximum number of allowed re-transmissions for a datagram (see C2 above). This event for $\text{Deaf}$ is more likely to occur when $Q_{\text{thr}}$ is small, because in this case BP congestion control is activated more often, which means that layer 2 packets are acknowledged less frequently and this leads to more layer 2 failures. $\text{Fuse}$ has the best performance among all L3Ds, with a $P_{\text{loss}}$ that is smaller than $10^{-4}$ for $Q_{\text{thr}} \leq 26$, whereas its $P_{\text{loss}}$ converges to that of $\text{Deaf}$ as $Q_{\text{thr}}$ approaches $Q_{\text{max}}$. This good performance is due to the combined effect of $\text{Griping}$ and $\text{Deaf}$ BP control.

Finally, from Fig. 30 we observe that $\text{IdealBP}$ has the best overhead performance, whereas $\text{Deaf}$ has the worst. $\text{Griping}$ performs very close to $\text{IdealBP}$ for all values of $Q_{\text{thr}}$. $\text{Fuse}$ performs very close to both $\text{IdealBP}$ and $\text{Griping}$ for $Q_{\text{thr}}$ smaller than $Q_{\text{max}}/2$, whereas as $Q_{\text{thr}}$ increases toward $Q_{\text{max}}$ the transmission overhead of $\text{Fuse}$ converges to that of $\text{Deaf}$. This is representative of the fact that the overhead in the latter case is dominated by the re-transmissions due to the stopped acknowledgment flow.

### 7.6 Back Pressure Congestion Control for CoAP

Current trends in IoT networking involve the use of Web services on constrained IoT devices. These entail the bi-directional exchange of messages according to a request/response paradigm, see Fig. 25, as per the REST architectural style [95]. The Constrained Application Protocol (CoAP) [80] defines a simple, efficient, and flexible protocol to allow REST architectures to scale down to smart objects, by preserving interoperability with HTTP [96].

In this section, we apply back pressure congestion control to bidirectional CoAP traffic. In this case, typical communication patterns amount to the transmission of CoAP control messages from the outside Internet network to the constrained IoT nodes and their subsequent CoAP responses. CoAP makes it possible for IoT resources to be accessible as Web services, and in particular makes them available on the Internet as HTTP Web services (through CoAP to/from HTTP mapping, see [96]).

Referring to the second case study of Fig. 25, CoAP requests are sent over the constrained network as IPv6 datagrams flowing from the 6LoWPAN border router down to the leaf nodes. Upon receiving these requests, leaf nodes reply with IPv6 datagrams carrying the corresponding CoAP responses; the latter datagrams flow from the leaf nodes to the border router.

Note that the congestion problem is only marginally handled by the CoAP specification, which recommends a fixed congestion window of 1 packet at the CoAP senders. However, this static window may result in underutilized transmission resources when the network has some residual transport capacity and is as well inefficient when even this small window value suffices to create congestion. Differently, our approach is to prevent network queues from
overrunning and as well to avoid the injection by the border router of an excessive number of requests into the constrained network. The latter objective is accomplished at the border router through the rejection of requests coming from the external Internet network using a “503 Service Unavailable” error response, which signals to the requesting client that the wanted resource is temporarily unavailable. This combined control is the purpose of our study in the following.

7.6.1 L3 Devices for Bidirectional Back Pressure

Differently from traditional BP, when bidirectional traffic is taken into account some additional mechanisms need to be added to the congestion control policies. In fact, BP should not slow down response traffic, because any dropped CoAP response would mean a network loss from the client’s perspective.

Taking into account the fact that every CoAP request solicits a CoAP response flowing along the reverse path, the length of the network queues is still a valid measure of network congestion. However, this measure alone is not entirely representative of the number of outstanding CoAP requests that are still waiting for a corresponding CoAP response message. Note that these responses may still cause buffer overruns as they are transmitted over the constrained network. Ideally, one would need to track the number of outstanding CoAP requests, so as to gauge the expected future load due to CoAP responses and shape the data traffic accordingly. However, such a task is generally too complex for resource constrained IoT devices.

Aiming at a lightweight design, in the following we modify the L3Ds of Section 7.4 with the objective of pushing back CoAP requests based on the queue length metric alone. While suboptimal, this solution entails little changes on current CoAP stacks and incurs low communication overhead. The goal of this section is to check whether, in spite of its simplicity, our queue-length-based control can provide satisfactory performance and also check which are the most important parameters that have to be tuned for its successful utilization.

Hence, the L3D devices of Section 7.4 have been modified as follows:

- **Static** This device does not apply any congestion control algorithm and is unchanged with respect to that of Section 7.4.

- **IdealBP** This device only applies its queue-length-based differential BP to the CoAP request traffic flowing from the border router to the leaf IoT nodes (incoming CoAP requests).

- **Griping** This device emits its explicit back pressure messages only upon the reception of CoAP requests (inbound CoAP traffic), whereas congestion control is not applied to CoAP responses (outbound CoAP traffic).

- **Deaf** This device implements the backoff policy of Section 7.4 and refrains from transmitting layer 2 acknowledgements when the corresponding layer 3 datagrams are CoAP requests.

- **Fuse** This device extends the Fuse BP policy of Section 7.4 adding a further threshold $Q_{thr2}$ such that $Q_{thr} < Q_{thr2} < Q_{max}$. The Deaf BP policy is activated when the queue len
length grows beyond the new threshold $Q_{\text{thr}2}$, whereas the behavior of the Griping BP policy remains unchanged. This second threshold allows the activation of BP congestion before the layer 3 queue is filled with packets and this leaves some room to accommodate the CoAP reverse traffic.\footnote{In other terms, the difference $Q_{\text{max}} - Q_{\text{thr}2}$ is our best \textit{a priori} estimate of the impact of the CoAP responses that will follow the CoAP requests that are currently admitted to the network.} As for Griping and Deaf, BP is only applied to CoAP requests.

For the border router, whenever its layer 3 queue becomes full, it rejects any further incoming CoAP request by issuing a 503 Service Unavailable error message.

![Figure 31: Received response rate $\lambda_{rx}$, and round-trip time $D$ vs. the offered request load $\lambda_{tx}$.](image)

7.6.2 System Parameters

- **Offered request load** $\lambda_{tx}$ defines the rate at which each CoAP client, placed in the external Internet network, sends CoAP requests toward a specific CoAP server placed within the constrained IoT network. Note that a server corresponds to an IoT leaf node in our simulation scenario, see Fig. 25.

The definition of the remaining system parameters $Q_{\text{thr}}$, $Q_{\text{max}}$, $N_{\text{retx}}$ remains the same as that of Section 7.5.1, the new threshold $Q_{\text{thr}2}$ has been set to $Q_{\text{thr}} + 5$ packets.

7.6.3 Performance Metrics

To compare the performance of the proposed L3Ds, the following performance metrics have been considered.
- **Received response rate** $\lambda_{rx}$ defines the average per server (running on a leaf node) rate of CoAP responses that are correctly received by the border router and is measured in correctly received CoAP responses per second per server.

- **Round trip-time** $D$ defines the average lapse of time (seconds) spent at the border router waiting for a CoAP response to an accepted CoAP request.

- **Loss probability** $P_{loss}$ defines the percentage of CoAP responses that are not received by the border router, although the corresponding CoAP requests have been accepted into the constrained network.

- **Rate of rejects** $R$ defines the average per client rate of CoAP requests that are not accepted into the network by the border router (issuing an HTTP 503 status code, as per our discussion above) and is measured in terms of rejected CoAP requests per second per client.

- **Transmission Overhead** $N_{tx}$ represents the average number of layer 2 packets that are transmitted in the network for the successful end-to-end bidirectional exchange (from the border router to a leaf node and back to the border router) of a single CoAP request and response pair. This metric accounts for the layer 2 packets that are sent to carry CoAP requests and responses as well as layer 3 BP control messages, such as those sent by *Gripping* and *Fuse* for the explicit signaling of a congestion event.

![Graph](image)

Figure 32: Transmission overhead $N_{tx}$ vs. the offered request load $\lambda_{tx}$.

### 7.6.4 Results for Bi-directional CoAP Traffic

Simulations have been run over the topology of Fig. 25, where a CoAP server has been deployed on each of the 9 leaf nodes; the border router hosts a CoAP proxy, which accepts CoAP requests from 9 CoAP clients placed in the external Internet network. Each
CoAP client emits Non-Confirmable (NON)\(^{10}\) requests at a constant rate \(\lambda_{tx}\) toward a CoAP server running on a leaf node. CoAP requests and responses have a fixed layer 3 size of 12 bytes and 115 bytes, respectively, including 6LoWPAN/UDP headers. The duration for each simulation run is 500 seconds, and the queue size of all nodes is 31 packets. The simulation points on the following graphs have been obtained averaging over 180 independent simulation runs.

7.6.5 Impact of varying \(\lambda_{tx}\)

As a first result, Fig. 31 shows \(\lambda_{rx}\) and \(D\) as a function of \(\lambda_{tx} \in \{1, \ldots, 15\}\) req/client/s. The remaining simulation parameters are \(Q_{\text{max}} = 31\) packets, \(Q_{\text{thr}} = 20\) packets, \(Q_{\text{thr2}} = 25\) packets and \(N_{\text{retx}} = 15\).

For Fig. 31, we note that the general behavior of all metrics is similar to that observed for unidirectional traffic, see Fig. 26. The main difference is that in this case \(\lambda_{\text{sat}}\) is nearly halved due to the presence of CoAP bidirectional exchanges, whereby two packets (CoAP request and response) must be handled by the network for each accepted CoAP request. In fact, although CoAP requests and responses differ in size, their cost in terms of overall time spent, including retransmissions, is nearly the same and this is due to the dominating effect of MAC layer tasks such as the time required to gain access to the channel, back off times, etc., which do not depend on the data frame size.

Also, we note that \(\lambda_{tx}^{\text{Static}}\) equals \(\lambda_{tx}^{\text{Static}}\) up to about \(\lambda_{\text{sat}} = 5\) req/client/s, beyond which the response rate starts decreasing to a floor of about 3 req/client/s. This behavior is due to the so called congestion collapse event, similar to that observed in the early days of the Internet (see [82, 97]). The congestion collapse is caused by the border router accepting more requests than those that can be served by the network, which is given by \(\lambda_{\text{Static}}^{\text{sat}}\). For the delay, we note that \(D\) grows until \(\lambda_{rx}\) stabilizes.

Notably, \(\text{IdealBP}\), \(\text{Griping}\), \(\text{Deaf}\) and \(\text{Fuse}\) are not subject to the congestion collapse of \(\text{Static}\) but their throughput performance stabilizes as soon as \(\lambda_{tx}\) increases beyond \(\lambda_{\text{sat}}\). Moreover, their delay remains stable even with \(\lambda_{tx}\) larger than \(\lambda_{\text{sat}}\). This occurs because the border router acts as a proxy by rejecting traffic as soon as its outbound queue toward the constrained network becomes full.

For the rejection rate \(R\), similarly to what observed for Fig. 27 (unidirectional flows), \(\text{Static}\) starts rejecting packets when \(\lambda_{\text{rx}}^{\text{Static}}\) is approximately 10 req/client/s, which is about twice \(\lambda_{\text{sat}}\). The remaining L3Ds react to an increasing \(\lambda_{tx}\) by rejecting packets as soon as the offered traffic increases beyond \(\lambda_{\text{sat}}\), with \(\lambda_{\text{sat}}\) halved with respect to that of Fig. 27. As for the unidirectional traffic scenario, \(\text{IdealBP}\) shows no losses, \(P_{\text{loss}}^{\text{IdealBP}}\) converges to about 0.5\%, \(P_{\text{loss}}^{\text{Deaf}}\) and \(P_{\text{loss}}^{\text{Fuse}}\) both stabilize around \(10^{-5}\).

Overall, it is worth noting that \(\text{Fuse}\) obtains nearly the same throughput as \(\text{Griping}\) but has the same \(P_{\text{loss}}\) performance as \(\text{Deaf}\). This is due to the combined effect of \(\text{Griping}\)'s explicit signaling, which effectively limits the send rate within the network, and the fact that all requests are deterministically rejected by the border router when its queue length increases beyond \(Q_{\text{thr2}}\), which helps preventing congestion events.

\(^{10}\)NON requests do not have application-layer retransmissions; we chose to use this kind of requests, since our objective is the layer 3 evaluation of the congestion control performance.
In Fig. 32 we look at the transmission overhead $N_{tx}$. As for the unidirectional traffic scenario, IdealBP shows a smaller transmission overhead than the other schemes for $\lambda_{tx} \leq \lambda_{sat}$. Static presents the highest $N_{tx}$, which increases for increasing $\lambda_{tx}$ until it hits a maximum, which occurs at around $2\lambda_{sat}$, when the corresponding $R$ starts increasing. The remaining back pressure schemes effectively limit the maximum overhead and in particular we note that Deaf performs quite well here, in contrast to its unsatisfactory overhead performance for unidirectional traffic. The reason for this is that in this case the border router is the only source of data traffic and sends its packets directly over the bottleneck link of the network. In this case, Deaf’s exponential backoff mechanism effectively keeps the overhead at a small value. This is in contrast to what happens for the unidirectional scenario where: i) there are multiple sources competing for the channel (multiple leaf nodes), ii) the considered tree topology is such that these multiple sources all insist onto the same routers and the data traffic is ultimately conveyed to a single border router (from many nodes to one), leading to an increasing congestion status as the data gets closer to the border router. Thus, in the unidirectional upstream case these facts result in a much more congested network and Deaf’s exponential backoff alone is ineffective.

The good performance of Deaf for bidirectional CoAP flows makes it suitable to add BP functionalities to current CoAP/6LoWPAN protocol stacks, without requiring the definition of further BP messages. In fact, in spite of its simplicity this scheme effectively avoids the network collapse and also leads to a reasonably small traffic overhead.

### 7.6.6 Impact of varying $N_{retx}$

Fig. 33 shows $P_{loss}$ by varying $N_{retx}$ in $\{0, 1, ..., 15\}$. The remaining simulation parameters are $\lambda_{tx} = 20$ req/client/s (the system is congested), $Q_{max} = 31$ packets, $Q_{thr} = 20$ packets and $Q_{thr2} = 25$ packets.

As observed in Section 7.2, BP requires an adequate number of hop-by-hop retransmissions.
to work. In fact, IdealBP obtains no substantial advantage over Static when no retransmissions are allowed, whereas even a very small number of retransmissions ($N_{retx} \leq 3$ for the considered setup) is sufficient for it to effectively relieve network congestion (see the sudden drop of $P_{loss}^{\text{IdealBP}}$ as $N_{retx}$ grows).

Griping and Deaf require as well an adequate number of retransmissions to effectively work. $P_{loss}^{\text{Griping}}$ drops quickly and then stabilizes to a floor of about 0.2%. $P_{loss}$ monotonically decreases for increasing $N_{retx}$ for both Deaf and Fuse, although the latter requires a higher number of retransmissions due to the delayed BP control implied by the new threshold $Q_{thr2} > Q_{thr}$. As in the unidirectional scenario, a high number of retransmissions does not negatively impact $N_{tx}$, which remains stable for all devices with $N_{retx} > 3$.

### 7.6.7 Impact of varying $Q_{\text{max}}$

Memory requirements have a strong relevance for constrained devices. In particular, the available memory and its management limit the queue length in actual implementations, e.g., see [94].

Figs. 34 and 35 show $\lambda_{rx}$ and $P_{loss}$ by varying $Q_{\text{max}}$ in $\{3, 6, \ldots, 60\}$. The remaining simulation parameters are: $Q_{thr} = \lceil (2/3)Q_{\text{max}} \rceil$ packets, $Q_{thr2} = Q_{thr} + 5$ packets, $N_{retx} = 15$ and $\lambda_{tx} = 20$ req/client/s.

![Figure 34: Received response rate $\lambda_{rx}$ vs. the maximum queue length $Q_{\text{max}}$.](image)

The throughput $\lambda_{rx}$ of Static remains stable around 3 res/client/s and is only marginally affected by $Q_{\text{max}}$. For IdealBP, Griping and Fuse, $\lambda_{rx}$ converges to about 5.2 res/client/s for $Q_{\text{max}} \geq 15$ packets. Thus, besides improving reliability, BP control also makes it possible to roughly double the throughput performance. We also observe that Deaf has a throughput performance that is roughly from 5 to 10% worse than that of the other BP schemes. The reason for this is inherent in how Deaf reacts to congestion events. In fact, Deaf detects congestion by stopping the transmission of the layer 2 acknowledgments associated with layer 3 CoAP requests. This has the twofold effect of slowing down the send rate of
CoAP requests, while occupying the channel with their retransmissions. However, these retransmissions prevent the senders from exploiting the channel for other useful traffic such as the transmission of CoAP responses, whose correct delivery would contribute to a higher throughput performance. The performance gap between *Fuse* and the other BP schemes decreases for increasing $Q_{\text{max}}$, as $Q_{\text{th}}$ also increases, leading to a less frequent activation of BP control policies and of the just discussed inefficiencies in terms of channel utilization (waste of channel resources).

For $P_{\text{loss}}$ from Fig. 35 we see that *Static* is unaffected by $Q_{\text{max}}$, whereas the reliability performance improves for all other L3Ds for increasing $Q_{\text{max}}$. This occurs because a larger $Q_{\text{max}}$ implies that network queues have more room to absorb traffic bursts and this makes buffer overruns less likely to occur. We also observe that *Deaf* has a slightly smaller $P_{\text{loss}}$ than *Fuse* as its smaller BP threshold $Q_{\text{th}} < Q_{\text{th}2}$ implies a prompter reaction to congestion events.

### 7.7 Results for Asymmetric Topologies and Cross-Traffic

#### 7.7.1 Asymmetric node deployment

To assess the validity of the discussed congestion control techniques for asymmetric node deployments, the reference topology of Fig. 25 has been modified by moving 2 nodes from subnet S1 to subnet S3. The new node count for each subnet is as follows: S1 has a single node, S2 has three nodes, and S3 has five nodes.

Fig. 36(a) shows the obtained $P_{\text{loss}}$ for each of the subnets for the symmetric topology of Fig. 25, whereas Fig. 36(b) shows the performance metric obtained for the asymmetric topology discussed above. The simulation parameters are $\lambda_{\text{tx}} = 15 \text{ req/client/s}$, $Q_{\text{max}} = 31$ packets, $Q_{\text{th}} = 20$ packets, $Q_{\text{th}2} = 25$ packets and $N_{\text{retx}} = 15$.

![Figure 35: Loss probability $P_{\text{loss}}$ vs. the maximum queue length $Q_{\text{max}}$.](image)

Date: June 11, 2013
Version: Final
As expected, the reliability performance for asymmetric topologies is slightly impacted, especially for those subnets containing a larger number of nodes (in our example scenario S2 and especially S3). For medium size subnets, such as S2, the performance of *Fuse* is close to that of *Deaf* and both achieve small error rates. However, as the number of nodes increases (S3) the *Fuse* device is to be preferred due to its higher robustness: here the addition of *Griping* ’s explicit BP messages leads to extra benefits in terms of $P_{\text{loss}}$. The remaining performance metrics $\lambda_{\text{rx}}$, $R$, $D$ and $N_{\text{tx}}$ are almost unaffected.

### 7.7.2 Layer 3 devices for cross-traffic

In what follows, we study the impact of cross-traffic due to the simultaneous presence of multiple border routers. Note that this is an unlikely scenario for 6LoWPAN networks adopting RPL [92], as in this standard multiple and distinct RPL sub-trees will be created for each of the border routers so as to minimize the interference among them. Nevertheless, some residual interference is still possible in practice and in the remainder we evaluate its impact.

For the networking scenario we still consider the network topology of Fig. 25, where a first border router BR1 (node BR in the figure) sends CoAP requests to nodes 1, 5, 6, 7 and 8; the CoAP request rate from BR1 is indicated as $\lambda_{1,\text{tx}}$. In addition, node 9 has been replaced with a second border router, BR2, which sends CoAP requests to nodes 2, 3 and 4; the CoAP request rate from BR2 is indicated as $\lambda_{2,\text{tx}}$. CoAP clients and servers are configured as specified in Section 7.6.4. Further, we define three groups of nodes as follows: $G_1 = \{1, 5, 6\}$, $G_2 = \{7, 8\}$ and $G_3 = \{2, 3, 4\}$ where $G_1$ and $G_2$ contain the nodes that send their CoAP responses to BR1, whereas the nodes in $G_3$ send CoAP responses to BR2.

For this scenario, routers R1 and R4 deal with CoAP requests flowing in opposite directions and the devices defined in Section 7.6.1 deliver poor performance in this case. Next, we redefine our practical layer 3 BP devices to effectively handle this situation.

- **Static** This device remains unchanged with respect to that of Section 7.4.
• **Griping** This device follows the same rules explained in Section 7.4 and further refined in Section 7.6.1. Thus, flow control is only applied to CoAP requests, following the algorithm of Section 7.4. However, in order to effectively address the presence of CoAP requests coming from different border routers, the **Griping** device at the receiver additionally implements the following rule. Whenever, upon the reception of a CoAP request, the local layer 3 queue occupancy $Q$ is larger than the pre-defined threshold $Q_{thr}$, the **Griping** device at the receiver acts as follows: 1) it calculates the fraction $\eta \in [0, 1]$ of packets in the local layer 3 queue that are directed toward the same next hop (same layer 3 address) as that of the current CoAP request, and 2) an explicit BP message is sent back to the originator of the CoAP request only if $\eta$ is larger than or equal to a pre-defined threshold $\eta_{thr}$. $\eta_{thr} = 0.15$ has been selected for the results in this section.

(Comment. This additional rule is implemented at the **Griping** receiver to avoid that one or more CoAP flows directed toward a certain set of border routers will take all the available bandwidth, whereas the remaining flows will starve. This rule effectively achieves this, as those flows which are given limited bandwidth (roughly less than $\eta_{thr}$) are not further slowed down through explicit BP messages. Flow control, through the reduction of the transmission rate is instead implemented for those flows that get the largest portion of the link capacity.)

• **Deaf** This device follows the same rules explained in Section 7.4 and further refined in Section 7.6.1. However, the following probabilistic rule at the receiver has been implemented to handle cross-traffic to multiple WSN sinks. The acknowledgement flow at layer 2 is stopped upon the reception of a CoAP request when the following conditions are verified: C1) the layer 3 queue occupancy $Q$ is larger than $Q_{thr}$, as defined in Section 7.6.1, C2) if C1 is verified, the transmission of the returning layer 2 ACK for the layer 2 packet associated with the current CoAP request is canceled with probability $p = \eta^{1/N_{retx}^{L2}}$, where $N_{retx}^{L2}$ is the number of retransmissions allowed at layer 2 and $\eta$ is the fraction of packets in the local layer 3 queue that are directed toward the same next hop (same layer 3 address) as that of the just received CoAP request.

(Comment. With this new probabilistic rule, whenever a CoAP request flow directed toward a certain layer 3 destination takes a fraction $\eta \in [0, 1]$ of the local layer 3 queue, the probability that $N_{retx}^{L2}$ subsequent layer 2 ACKs for a CoAP request belonging to this flow are denied is $\eta = p^{1/N_{retx}^{L2}}$. This implies that the probability that the layer 2 flow is stopped for a certain CoAP request is $\eta$. Note that a stopped layer 2 ACK flow is perceived by the **Deaf** transmitter as an implicit BP indication. The rationale behind this is that flows should be penalized according to the fraction of link capacity assigned to them.)

• **Fuse** Since it is defined as the combination of **Deaf** and **Griping**, its behavior arises from the combination of the previously redefined devices.
Figure 37: Average $P_{\text{loss}}$ in the presence of bidirectional CoAP cross-traffic for $\lambda_{tx}^1 = 15$ req/client/s.

**7.7.3 Results for CoAP cross-traffic**

Fig. 37 shows $P_{\text{loss}}$ in the presence of CoAP cross-traffic from BR2 for the following simulation parameters: $\lambda_{tx}^1 = 15$ req/client/s, $\lambda_{tx}^2 \in \{5, 15\}$ req/client/s, $Q_{\text{max}} = 31$ packets, $Q_{\text{thr}} = 20$ packets, $Q_{\text{thr}2} = 25$ packets and $N_{\text{retx}} = 15$. The case where the cross-traffic rate is moderate, i.e., $\lambda_{tx}^2 \leq 5$ req/client/s (see Fig. 37(a)), is reasonable in actual 6LoWPAN/RPL networks where this type of traffic may be due to, e.g., the presence of peer-to-peer traffic. The extreme case $\lambda_{tx}^2 = \lambda_{tx}^1$ (Fig. 37(b)) is instead undesirable and can be avoided by maintaining distinct RPL trees.

In general, $P_{\text{loss}}$ is impacted with respect to that of the previous networking scenarios. From Fig. 37(a), we note that the Deaf and Fuse devices can guarantee an error rate smaller than 1%, which may be adequate in most practical cases. Notably, Fuse shows robustness with respect to an increasing $\lambda_{tx}^2$, being almost unaffected as $\lambda_{tx}^2$ goes from 5 to 15 req/client/s, see Fig. 37(b).

**7.8 Conclusions on Congestion Control Algorithms**

In this section, we have proposed several congestion control techniques for IoT networks. These algorithms have been conceived to add congestion control capabilities to IETF CoAP / 6LoWPAN based protocol stacks and their benefits have been quantified for unidirectional as well as bidirectional CoAP flows, which are typical of Web architectures. Overall, our schemes lend themselves to implementation in existing solutions and perform satisfactorily under a wide range of parameter settings. Among them, Deaf looks particularly suitable for implementation purposes as this schemes requires minimal modifications to existing protocol stacks and does not involve the definition of additional messages. Also, in spite of its simplicity, the throughput performance of this algorithm is only $5 - 10\%$ smaller than that of the best performing scheme, while leading to excellent results in terms of reliability. The best performing scheme is Fuse, which combines Deaf with explicit control messages to detect...
congestion events. This scheme performs well for all the considered metrics and appears as a valid solution for unidirectional and bidirectional traffic. Our future work is devoted to reducing the impact of congestion control on the end-to-end delay, to further evaluating the impact of different topologies on the performance of BP congestion control algorithms, and to the investigation of mechanisms for the self-tuning of key system parameters (i.e., $Q_{thr}$ and $Q_{thr2}$), so that the proposed congestion control algorithms can automatically adapt to any topology, network and traffic flow model. Analytical and experimental evaluations are also interesting future research directions.

8 Conclusions

In this deliverable we have presented our final design of the IOT-A protocol architecture. In doing so we have considered current developments from relevant standardization committees such as IETF and, at the same time, we have extended existing functionalities through novel contributions. Specifically, these new developments are centered on the enhancement of 1) security functionalities, aimed at providing “sufficient” security to the constrained IoT devices, by simplifying and minimizing, as much as possible, the amount of software components and procedures that are required in the IoT devices and the enhancement of 2) transport functionalities for constrained domains, where standard TCP is proved inadequate and UDP, as currently used by IETF CoAP and 6LoWPAN-based stacks, by its own nature is not capable of providing adequate levels of reliability.

Our designs are presented in the deliverable by discussing our choices, comparing the resulting protocols with respect to state of the art solutions, and quantitatively showing the corresponding performance improvements through experimentations and simulations.

References


[20] “DTDPPM Compressor.”

[21] “Xwrt: enhanced XMll compressor.”


[49] “Secure Hash Algorithm (SHA).”

[50] “Zolertia Z1 Platform.”


Date: June 11, 2013
Version: Final


[75] “IT++ Library.”


[91] “Iperf Project.”


[93] “NS-3 Project.”

