

**Verbeteren van efficiëntie, bruikbaarheid en schaalbaarheid
in een veilig, in mogelijkheden gelimiteerd 'Web der Dingen'**

**Improving Efficiency, Usability and Scalability
in a Secure, Resource-Constrained Web of Things**

Floris Van den Abeele

Promotoren: prof. dr. ir. J. Hoebeke, prof. dr. ir. I. Moerman
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen



Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. B. Dhoedt
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017 - 2018

ISBN 978-94-6355-038-3
NUR 986
Wettelijk depot: D/2017/10.500/73



Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Informatietechnologie

Promotoren: prof. dr. ir. Jeroen Hoebeke
prof. dr. ir. Ingrid Moerman

Juryleden: em. prof. dr. ir. Daniël De Zutter, Universiteit Gent (voorzitter)
prof. dr. ir. Jeroen Hoebeke, Universiteit Gent (promotor)
prof. dr. ir. Ingrid Moerman, Universiteit Gent (promotor)
prof. dr. ir. Eli De Poorter, Universiteit Gent (secretaris)
prof. dr. Danny Hughes, KU Leuven
dr. dipl.-ing. Matthias Kovatsch, ETH Zurich
prof. dr. Steven Latré, Universiteit Antwerpen
prof. dr. ir. Sofie Van Hoecke, Universiteit Gent

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Informatietechnologie
Technologiepark-Zwijnaarde 15, 9052 Gent, België

Tel.: +32-9-331.49.00
Fax.: +32-9-331.48.99



Proefschrift tot het behalen van de graad van
Doctor in de ingenieurswetenschappen:
computerwetenschappen
Academiejaar 2017-2018

Dankwoord

“Waarlijk, rechters, hoewel ik wens dat ik met elke deugd vereerd zou worden, is er niets dat ik hoger acht dan dankbaar zijn en dit te laten blijken. Daar deze deugd niet alleen de grootste, maar ook de bron van alle andere deugden is.”

– Marcus Tullius Cicero (54 v.C.)

Een doctoraatsproefschrift komt tot stand met de hulp en input van vele anderen. Dat is niet anders voor mijn proefschrift en ik wil dan ook eenieder bedanken die hieraan heeft bijgedragen. Allereerst wens ik Ingrid te bedanken, aangezien zij mij de kans bood om zowel mijn masterproef als mijn doctoraatsonderzoek aan te vatten bij wat destijds IBCN heette. Inhoudelijk ben ik Jeroen het meest erkentelijk en ik wens hem dan ook te bedanken voor de vele discussies en de voortreffelijke begeleiding en samenwerking gedurende de laatste vijf jaar. Naast Jeroen, ben ik mijn (ex-)collega's Isam, Girum, Cristian en Jetmir dankbaar voor hun adviezen en inzichten in de technische materie. Dit onderzoek is tevens het resultaat van de samenwerking met vele thesisstudenten, deze wens te bedanken voor hun samenwerking: Pieter, Kobe en Stef, Steven en Ruben, Tom, Mathieu, Maarten en Tomas. Tot slot, wil ik Eli, Steven, Danny, Sofie en Matthias erkennen voor hun inzet als jurylid tijdens de evaluatie van dit onderzoek. Allen hebben zij bijgedragen aan de kwaliteit van dit proefschrift.

Ik heb mijn tijd bij de onderzoeksgroep IDLab steeds als bijzonder verrijkend ervaren. Dit kwam voornamelijk door een uitzonderlijke mix van (jonge) persoonlijkheden en culturen. De collega's uit het voormalige 3.17 wens ik te bedanken voor de aangename werksfeer: Wim, Daan, Dries, David, Jen, Wei, Tom, Michael, Isam, Girum, Pieter W., Enri, Tarik en Elnaz. Ook na de verhuis naar Zwijnaarde bleef het aangenaam werken met Cristian, Adnan, Jen, Merima, Matteo, Andy, Amina, Michael, Enri en Mathias als bureaugenoten en Abdulkadir, Bart, Felipe, Jetmir, Irfan en Vasileios als collega's. Naast deze collega's wens ik ook Bram, Frederik, Marlies, Thijs, Sander en Sofie te bedanken voor de leuke momenten en interessante discussies.

Tot slot wens ik vrienden en dichte familie te bedanken voor hun luisterend oor en steun. Wij worden allen deels een product van onze omgeving en ik prijs mezelf dan ook gelukkig dat ik op een warme omgeving kan rekenen. In het bijzonder wens ik Albijn en Tim te bedanken voor de zes jaren ten huize 'Club Plumion'. Hanne en Elisabeth wens ik te bedanken voor hun adviezen en steun in moeilijke tijden. Lukas wil ik graag bedanken voor de vele heuglijke herinneringen, welke

ik blijvend koester, en om mij bewust te maken van de broosheid van het leven. Tot slot, wil ik mijn ouders, Antonia & Erik, bedanken voor hun onvoorwaardelijke liefde. Jullie steun en aanmoediging hebben er mede toe bijgedragen dat dit proefschrift succesvol is afgerond.

Sint-Martens-Latem, september 2017
Floris Van den Abeele

Table of Contents

Dankwoord	i
Samenvatting	xix
Summary	xxiii
1 Introduction	1
1.1 Background	1
1.1.1 Internet of Things	2
1.1.2 The resource-constrained Internet of Things	4
1.1.3 TCP/IP for the resource-constrained Internet of Things . .	6
1.1.4 Web of Things	7
1.1.5 Internet standards for web technology in constrained de- vices and networks	9
1.1.5.1 IPv6 over Low Power Wireless Personal Area Networks	9
1.1.5.2 The Constrained Application Protocol	10
1.1.6 Low Power Wide Area Networks	13
1.2 Research challenges	14
1.2.1 Efficient resource utilization in an open Web of Things . .	14
1.2.2 Holistic security in an open Web of Things	15
1.2.3 Heterogeneity in the resource-constrained Internet of Things	15
1.2.4 Usability of constrained devices	15
1.2.5 Scalability of emerging LPWA technologies	16
1.3 Outline	16
1.4 Research contributions	20
1.5 Publications	23
1.5.1 Publications in international journals (listed in the Science Citation Index)	23
1.5.2 Publications in other international journals	24
1.5.3 Publications in international conferences (listed in the Science Citation Index)	24
1.5.4 Publications in other international conferences	25
1.5.5 Contributions to standardization bodies	26
1.5.6 Patent applications	26

References	28
2 Sensor function virtualization to support distributed intelligence in the Internet of Things	33
2.1 Introduction	34
2.2 The need for distributed intelligence	36
2.2.1 Generic IoT system	36
2.2.2 Open challenges	37
2.2.2.1 Act in time	37
2.2.2.2 Work offline	39
2.2.2.3 Serve many	39
2.2.2.4 Move and sleep	39
2.2.2.5 Monoglot	40
2.2.3 Distributed intelligence	40
2.3 Sensor Function Virtualization for the Internet of Things	42
2.4 IETF protocol stack for the Internet of Things	47
2.5 SFV in the unconstrained domain	49
2.6 SFV in the constrained domain	53
2.7 Related work	53
2.8 Conclusions	55
References	57
3 Secure Service Proxy: A CoAP(s) Intermediary for a Securer and Smarter Web of Things	61
3.1 Introduction	62
3.2 Overview of CoAP and DTLS	65
3.2.1 The Constrained Application Protocol (CoAP)	65
3.2.2 Datagram Transport Layer Security (DTLS)	66
3.2.3 DTLS in constrained environments	69
3.3 Problem statement and research goals	70
3.3.1 End-to-end security in LLNs	70
3.3.2 Complex application features in LLNs	72
3.3.3 Problem statement: illustration in a smart building use case	73
3.4 The Secure Service Proxy	74
3.4.1 Motivation of approach	75
3.4.2 Secure Service Proxy: design	76
3.4.3 Secure Service Proxy: implementation	79
3.4.3.1 Virtual devices and endpoints	79
3.4.3.2 Implemented application layer adapters	80
3.4.3.3 Adapter chain management: interface	82
3.4.3.4 Authenticating (D)TLS clients on the SSP	83
3.4.3.5 Key management between SSP and constrained devices	84
3.5 Related work	84
3.6 Evaluation: results and discussion	87

3.6.1	Terminating end-to-end-security at the SSP	87
3.6.1.1	Simulation setup	88
3.6.1.2	Results	89
3.6.2	Aggregating multiple CoAPs clients at the SSP	91
3.6.2.1	Experiment setup	91
3.6.2.2	Results	93
3.7	Conclusions	96
	References	99
4	Improving User Interactions with Constrained Devices in the Web of Things	103
4.1	Introduction	104
4.2	Problem statement and research goals	104
4.3	User friendly interactions	106
4.3.1	Requirements	106
4.3.2	Approach	106
4.3.3	Design	108
4.3.4	Device mapping, discovery and naming	109
4.4	Evaluation	110
4.4.1	Evaluation setup	110
4.4.2	Functional evaluation	111
4.4.3	Interface responsiveness: load times	112
4.5	Related work	113
4.6	Conclusion	114
	References	116
5	Integration of heterogeneous devices and communication models via the Cloud in the constrained Internet of Things	119
5.1	Introduction	120
5.2	Case study: logistics and transport	122
5.3	Problem statement and research goals	123
5.4	Background: Embedded web services via CoAP	126
5.5	Cloud platform for supporting heterogeneous devices and communication models	128
5.5.1	The access layer: providing access to heterogeneous devices and communication models	129
5.5.2	The abstraction layer: a homogeneous RESTful interface for constrained devices	131
5.5.3	Machine to Machine communications	133
5.6	Evaluation	133
5.6.1	Virtual device abstraction: scalability and latency	134
5.6.2	Communication models: push vs pull	136
5.6.3	Proof of concept: real world deployment	138
5.7	Related work	143
5.8	Conclusion	145

References	147
6 Scalability analysis of large-scale LoRaWAN networks in ns-3	151
6.1 Introduction	153
6.2 Background: LoRa, LoRaWAN and ns-3	154
6.3 Problem statement and approach	157
6.4 LoRaWAN ns-3 module	158
6.4.1 LoRa PHY error model	158
6.4.1.1 LoRa PHY baseband implementation	158
6.4.1.2 LoRa PHY BER simulations	161
6.4.2 LoRaWAN PHY layer	161
6.4.3 LoRaWAN MAC layer	162
6.4.4 LoRaWAN class A end device ns-3 application	165
6.4.5 LoRaWAN gateway ns-3 application	165
6.4.6 LoRaWAN Network server	165
6.5 Scalability analysis of LoRaWAN networks	166
6.5.1 Assigning LoRa spreading factors to end devices	167
6.5.2 Unconfirmed vs confirmed upstream data	169
6.5.2.1 Single gateway LoRaWAN network	169
6.5.2.2 Multi gateway LoRaWAN networks	172
6.5.3 Downstream data traffic	174
6.6 Related work	176
6.7 Discussion	179
6.8 Conclusion	180
References	181
7 Conclusions and perspectives	185
7.1 Summary and conclusions	187
7.2 Outlook	189
A Integrating LoRaWAN networks into the Web of Things via device virtualization	193
A.1 Introduction	193
A.2 RESTful Web services for data sharing and control of LoRaWAN end devices	194
A.3 Binary data encoding over LoRaWAN	195
A.3.1 Proof of concept demonstration	196
A.4 Conclusion	197

List of Figures

1.1	Application domains of the Internet of Things (image courtesy of the Internet of Things (IoT) course at Ghent University)	3
1.2	Resource-constrained sensing devices from the Wireless Sensor Network (WSN) research community.	5
1.3	In the Web of Things (WoT), applications interact via RESTful requests and responses over the Internet regardless of the underlying network technology (image courtesy of ‘Building the Web of Things’)	7
1.4	In the Web Thing Model, Web Things are categorized into four different levels depending on their functionality (image courtesy of http://model.webofthings.io/)	9
1.5	Internet protocols and REST architecture for the resource-constrained Web of Things	11
1.6	Common architecture of Low Power Wide Area Networks (LPWANs): network gateways offer LPWAN-specific APIs in order to integrate LPWAN devices into applications and services	14
1.7	Situating the various chapters of this dissertation in a resource-constrained IoT network architecture	19
2.1	A generic Internet of Things system	36
2.2	Five IoT scenarios mapped to the generic IoT system from figure 2.1	38
2.3	The concept of distributed intelligence	41
2.4	Architecture for sensor function virtualization in the Internet of Things	43
2.5	Infrastructure at different locations throughout the Internet works together to provide sensor function virtualization in the Internet of Things	44
2.6	Three different integration strategies for cloud-based SFV	45
2.7	IETF protocol stack for low power and lossy networks in the Internet of Things	47
2.8	A CoAP request/response message exchange showing resource discovery and data retrieval	48
2.9	Two SFV modules providing emulated resources (EMU) and a mirror server abstraction (MS)	50

2.10	SFV at a trusted gateway provides DTLS termination, access control and caching	52
3.1	Anatomy of a typical CoAP request and response.	65
3.2	The full DTLS handshake.	67
3.3	In a smart building scenario, there is a wide variety of different users. Constrained devices are unable to offer all necessary security and application features to cater to these users. In the approach followed by this work, unconstrained systems (e.g., border routers (BRs)) assist by offering these missing features. CBOR: Concise Binary Object Representation, ACL: Access Control List.	73
3.4	Secure Service Proxy: design.	77
3.5	Cooja network topology: four CoAP(s) servers (6, 7, 8, 9) are located two hops away from the RPL border router.	88
3.6	Transaction times and energy usage of the CoAPs servers for the three gateway configurations (End-to-End (E2E), first Termination (TER1), n -th Termination (TER)) and the Plain Text CoAP reference case (PLT).	90
3.7	Representative RPL network topology: the node under study, node #50, is situated two hops from the border router, node #152.	91
3.8	Evaluation setup: a variable number of CoAPs clients observe one of two resources on either the virtual host (SSP) or the sensor node (E2E)	93
3.9	Total energy expenditure for node #50 per five seconds interval for end-to-end (E2E) CoAPs observation versus CoAPS observation through the Smart Service Proxy (SSP)	94
3.10	Number of exchanged packets for node #50 per five seconds interval for end-to-end (E2E) CoAPs observation versus CoAPS observation through the Smart Service Proxy (SSP)	95
3.11	Notification loss ratios as measured at the CoAPs clients for end-to-end (E2E) CoAPs observation versus CoAPS observation through the Smart Service Proxy (SSP)	97
4.1	Our approach serves users web interfaces of embedded web services on constrained devices.	107
4.2	System overview	107
4.3	Evaluation setup: 6LoWPAN Zolertia Z1's (red) and 802.11 WLAN nodeMCUs (gray) as constrained devices. Raspberry Pi as application proxy. Red arrows indicate CoAP exchanges, black arrows HTTP exchanges. Solid lines indicate IPv6 datagrams, dashed lines IPv4.	110
4.4	Device discovery via the RD endpoint lookup interface	111
4.5	Rendering .well-known/core of a constrained device	112

4.6	Different templates are rendered depending on the resource type of the target CoAP resource: e.g. ibcn.temp and ibcn.light are shown here.	113
4.7	CDFs of load times for different proxy configurations	114
5.1	Isolated vertical platforms hinder cross-vendor service delivery . .	122
5.2	Problem statement: As each domain of the IoT comes with its own set of IoT devices, protocols, standards, data formats and connectivity options, service providers are forced to integrate a multitude of different technologies when developing cross-domain services.	123
5.3	A typical request/response exchange between a CoAP client and server	126
5.4	CoAP mirror server: clients and sleeping endpoints can communicate in an asynchronous fashion	127
5.5	Virtual devices in the cloud represent their real heterogeneous counterparts	128
5.6	The access and abstraction layers of the design enable uniform access to heterogeneous constrained IoT devices.	129
5.7	Two Raspberry Pi's and an 802.15.4 wireless sensor network operating 6LoWPAN form the evaluation setup for our cloud platform	131
5.8	Cumulative distribution function of response times for confirmable CoAP POST requests	135
5.9	Left: stacked bar plot of median energy usage per category. Right: box plot of total energy usage.	137
5.10	Sum of packets received and transmitted for different communication models	138
5.11	Proof of concept: components and setup	139
5.12	(1)+(2): Crane tracker updates access layer mapping. (3)-(6): CoAP request to device abstraction (3) triggers corresponding access layer request (4).	141
5.13	Control and management dashboard: the user is presented with a list of devices and resources. The user can access data collected from resources, retrieve a new resource representation or update a resource.	141
5.14	The dashboard presents a table with collected data of a waste bin tracker to the user	142
6.1	Architecture of LoRaWAN networks (image courtesy of Semtech)	156
6.2	Downlink receive window timing for LoRaWAN class A end devices	156
6.3	LoRaWAN ns-3 module overview: class A end devices, gateways and the network server	159
6.4	Block diagram of LoRa PHY baseband implementation: sender, AWGN channel and receiver	160
6.5	Curve fits used for the LoRa PHY error model in ns-3.	163
6.6	Finite state machine of the LoRaWANPhy class in ns-3	164

6.7	The LoRaWANMac FSM consists of three states for gateways and seven states for class A end devices	164
6.8	Positions for one (cross), two (circles) and four (rectangles) gateways in ns-3 simulations	167
6.9	Packet delivery ratios for various spreading factor assignments strategies	168
6.10	Spreading factor allocation to end devices for PER strategy (0.01)	169
6.11	PDR for unconfirmed (UNC), $NbRep = 4$ unconfirmed (4UNC) and confirmed (CON) upstream messages in a single gateway LoRaWAN network.	170
6.12	PDR for unconfirmed (UNC), $NbRep = 4$ unconfirmed (4UNC) and confirmed (CON) upstream messages in a two gateway LoRaWAN network.	173
6.13	PDR for unconfirmed (UNC), $NbRep = 4$ unconfirmed (4UNC) and confirmed (CON) upstream messages in a four gateway LoRaWAN network.	174
A.1	LoRaWAN end devices are abstracted as virtual CoAP servers to facilitate data exchange and control in the WoT	194

List of Tables

1.1	An overview of the targeted research challenges per chapter in this dissertation.	20
3.1	The proxy offers a number of functionalities, called adapters, that are hosted on virtual devices. The list of adapters that were implemented at the time of this work are shown in this table.	81
6.1	LoRa PHY parameters for BER simulations	161
6.2	Exponential curve fit parameters for the LoRa PHY error model in ns-3	162
6.3	caption	171
6.4	End devices at a specific data rate for LoRaWAN networks with one, two and four gateways	172
6.5	Packet delivery ratios of downstream data messages	175
6.6	Packet delivery ratios of upstream messages in the presence of downstream data	176

List of Acronyms

0-9

6lo	IPv6 over Networks of Resource-constrained Nodes
6LoWPAN	IPv6 over Low power WPAN

A

ACL	Access Control List
API	Application Programming Interface

B

BMS	Building Management System
------------	----------------------------

C

CBOR	Concise Binary Object Representation
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments

D

DI	Distributed Intelligence
DNS	Domain Name System
DTLS	Datagram Transport Layer Security

G

GPRS	General Packet Radio Service
GUI	Graphical User Interface
GW	Gateway

H

HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation and Air Conditioning

I

IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv6	Internet Protocol version six
IPSO	Internet Protocol for Smart Objects
IT	Information Technology

J

JSON	JavaScript Object Notation
-------------	----------------------------

L

LAN	Local Area Network
LLN	Low-Power and Lossy Network

LoRaWAN	LoRa Wide Area Network
LPWA	Low Power Wide Area
lpwan	IPv6 over Low Power Wide-Area Networks
LPWAN	Low Power Wide Area Network

M

M2M	Machine-to-Machine
MAC	Medium Access Control
Mbps	Megabits per second
MQTT	Message Queuing Telemetry Transport
MTU	Maximum Transmission Unit

N

NFV	Network Function Virtualization
------------	---------------------------------

O

OS	Operating System
-----------	------------------

P

PKI	Public Key Infrastructure
PSK	Pre-Shared Key

Q

QoS	Quality of Service
------------	--------------------

R

RAM	Random Access Memory
------------	----------------------

RD	Resource Directory
RDC	Radio Duty Cycle
REST	Representational state transfer
RFC	Request For Comments
ROLL	Routing Over Low power and Lossy networks
ROM	Read Only Memory
RPK	Raw Public Key
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks

S

SCHC	Static Context Header Compression
SDO	Standards Development Organization
SFV	Sensor Function Virtualization
SOA	Service-oriented Architecture
SoTA	State-of-The-Art
SSP	Secure Service Proxy

T

TCP	Transmission Control Protocol
------------	-------------------------------

U

UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier

W

WAN	Wide Area Network
WG	Working Group

WoT	Web of Things
WPAN	Wireless Personal Area Network
WSAN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network
W3C	World Wide Web Consortium
WWW	World Wide Web

Samenvatting

– Summary in Dutch –

Het Internet der dingen (IoT) belooft tal van bedrijfsprocessen te hervormen door (alledaagse) objecten te verbinden, data over onze omgeving te verzamelen en beslissen te nemen en uit te voeren op basis van deze gecapteerde data. Via ‘big data’ analyse, verwachten we nieuwe informatie te kunnen afleiden uit de stormvloed aan data afkomstig van intergeconnecteerde dingen. Deze nieuwe informatie zal ons in staat stellen om onze omgeving beter te doorgronden en zal daardoor leiden tot beter geïnformeerde beslissingen. Verder wordt er verwacht dat het Internet der dingen zal leiden tot nieuwe bedrijfsmodellen omwille van de goedkope monitoring en aansturing aan de hand van intergeconnecteerde objecten. In fabrieken wordt er verwacht dat geconnecteerde machines de automatisatie verder zullen opvoeren en de efficiëntie van productieprocessen zullen verbeteren. In de verzekeringssector wordt er verwacht dat intergeconnecteerde objecten zullen leiden tot verbeterde veiligheid, beveiliging en verliespreventie. In de gezondheidszorg worden er geconnecteerde producten verwacht die nieuwe gezondheids- en fitnessdiensten zullen aanbieden met het oog op het verlagen van de kost van gezondheidszorg en het verhogen van de kwaliteit van de aangeboden zorg. Naast deze voorbeelden, belooft het IoT ook tal van andere sectoren te beïnvloeden: transport en goederenvervoer, slimme gebouwen en domotica, nutsvoorzieningen, landbouw en vele anderen.

Vandaag worden de meeste geconnecteerde producten ontworpen voor één specifieke verticale markt waardoor hun werking beperkt wordt tot een gesloten ecosysteem. Vaak staan producenten zelfs weigerachtig tegenover integratie met externe systemen, omdat zij dit percipiëren als risicovol voor hun concurrentiepositie. Deze fragmentatie van het IoT is een groot probleem omdat het horizontale platformen, die kunnen communiceren, samenwerken en programmeren over allerlei toestellen ongeacht hun producent, model of toepassingsdomein, verhindert om de IoT markt te betreden. De naadloze interoperabiliteit vereist voor zulke horizontale platformen, is vitaal om het succes en het potentieel van het IoT op lange termijn te verzekeren en te realiseren. De gefragmenteerde situatie van het IoT is in vele opzichten vergelijkbaar met het begin van het internet, toen vele pakketswitchingtechnologieën elkaar bekwamen voor marktaandeel.

Het Web der dingen (WoT) werd voorgesteld als een oplossing voor dit fragmentatieprobleem. Het neemt de succesvolle webstandaarden, technologie en hulpmiddelen van het hedendaagse wereld web (WWW) en past deze toe

op het interconnecteren van dingen. In het WoT model stellen dingen hun diensten en data ter beschikking via RESTful programmeringsinterfaces (APIs) en dit ongeacht hun producent, toepassingsdomein of connectiviteitstechnologie. Doordat het bouwen van RESTful APIs door middel van de traditionele TCP/HTTP protocollen moeilijk en soms zelfs onhaalbaar is voor toestellen met gelimiteerde mogelijkheden, heeft de Internet Engineering Task Force (IETF) het Constrained Application Protocol (CoAP) gestandaardiseerd. Dergelijke toestellen zijn doorgaans gelimiteerd in hun grootte, kostprijs, rekenvermogen, geheugencapaciteit, connectiviteit en voedingsbron. CoAP maakt het mogelijk om RESTful ingebouwde web services te implementeren in IoT toestellen met gelimiteerde mogelijkheden op een gestandaardiseerde manier. De term Constrained RESTful environments (CoRE) (gelimiteerde RESTful omgevingen) verwijst naar het onderdeel van het IoT dat Representational State Transfer (REST) toepast op intergeconnecteerde dingen met gelimiteerde mogelijkheden.

Hoewel de standaardisatie van CoAP een belangrijke stap is naar het verwezenlijken van het in mogelijkheden gelimiteerde WoT, heeft deze ook aanleiding gegeven tot een aantal nieuwe en interessante vragen tijdens het inzetten van CoAP in gelimiteerde netwerken en toestellen. Dit proefschrift bestudeert uitdagingen en adoptiebarrières gerelateerd aan de integratie van in mogelijkheden beperkte toestellen in het WoT. Zodoende is het hoofddoel van dit doctoraatsonderzoek het oplossen van openstaande problemen en het wegnemen van adoptiebarrières van open web standaarden in CoRE. Door deze technische uitdagingen het hoofd te bieden, hoopt dit onderzoek bij te dragen aan de adoptie van open webstandaarden in het in mogelijkheden gelimiteerde IoT. Enkel door te bouwen op open standaarden en systemen kan het IoT evolueren van de gefragmenteerde toestand van vandaag naar een toekomst waar alles met elkaar verbonden zal zijn.

Een eerste verzameling uitdagingen betreft de impact van alles met elkaar te verbinden in een open WoT op de werking van systemen met gelimiteerde mogelijkheden. Gezien het IoT blijft groeien, moet men opletten dat het open model van het WoT de werking van CoRE niet verhindert. Inderdaad, bij een in omvang toenemend IoT zal de belasting op toestellen en netwerken met gelimiteerde mogelijkheden ook toenemen waardoor de noodzaak van het efficiënt aanwenden van de gelimiteerde mogelijkheden nog belangrijker zal worden. Gekoppeld aan deze groei, is de nood aan alomvattende en schaalbare beveiligingsmechanismen in CoRE die authenticatie, autorisatie, afscherming van gevoelige informatie en vertrouwelijkheid aanbieden. Sommige van deze beveiligingsaspecten zijn moeilijk te realiseren in CoRE omwille van de gelimiteerde mogelijkheden. Dit proefschrift presenteert Distributed Intelligence (gedistribueerde intelligentie, DI) als een oplossingsmethode voor deze technische uitdagingen. DI herkent dat rekenkracht kan verdeeld worden over het internet en dat hierdoor gelimiteerde systemen kunnen worden bijgestaan in hun werking. Meer specifiek stelt de virtualisatie van gelimiteerde toestellen op krachtigere systemen ons in staat om gelimiteerde toestellen uit te breiden met nieuwe functionaliteiten. Dit bevordert de beveiligingsmechanismen en het efficiënt aanwenden van de beperkte mogelijkheden. Dit werk presenteert de Secure Service Proxy (SSP) die door middel van het inzet-

ten van adapters op virtuele toestellen er in slaagt om gelimiteerde toestellen uit te breiden in mogelijkheden. Adapters zijn modulaire blokken van functionaliteit die RESTful aanvragen en antwoorden verwerken via CoAP. De evaluatie van de SSP toont aan dat de proxy schaalbare authenticatie en verfijnde autorisatie brengt naar gelimiteerde toestellen, terwijl het de kost van sessie gebaseerde beveiliging in termen van energieverbruik en communicatievertraging vermindert. De evaluatie toont ook dat de SSP erin slaagt om de belasting op de gelimiteerde systemen te verminderen en de capaciteit van gelimiteerde netwerken te verhogen door het combineren van resource-aanvragen.

Een andere groep van adoptiebarrières wordt veroorzaakt door de beperkte bruikbaarheid van gelimiteerde toestellen in CoRE. Voor eindgebruikers toont deze beperkte bruikbaarheid zich in de afwezigheid van gebruikersinterfaces ter ondersteuning van de interactie met gelimiteerde toestellen. Dit proefschrift presenteert een methode voor het genereren van grafische gebruikersinterfaces door middel van web templates. Deze methode stelt gebruikers in staat om gelimiteerde toestellen te bevragen en aan te sturen aan de hand van welbekende web technologie in hun (mobiele) webbrowser. Uit het standpunt van softwareontwikkelaars betreft de beperkte bruikbaarheid het ontbreken van verwachte eigenschappen op gelimiteerde toestellen. Dit gebrek manifesteert zich bijvoorbeeld in de afwezigheid van semantische beschrijvingen van web services en de beperkte aanpasbaarheid van gelimiteerde toestellen doordat het updaten van firmware vaak omslachtig of onmogelijk is. Om deze problemen op te lossen werd de hierboven vermelde virtualisatie techniek toegepast, bv. voor het emuleren van CoAP observe en het aanbieden van semantische beschrijvingen. Door het virtualiseren van RESTful resources, is het mogelijk om ontbrekende functionaliteit toe te voegen en bestaande functionaliteit te updaten zonder de nood aan een kostelijk updatemechanisme op de gelimiteerde toestellen.

De diversiteit in connectiviteitstechnologieën, communicatiemodellen en applicatiemodellering vormt de basis voor de derde kwestie behandeld in dit proefschrift. Aangezien legacy technologieën nog vele jaren zullen blijven bestaan in het IoT, is het noodzakelijk om deze technologieën te integreren in CoRE. Verder bestaan er binnen CoRE ook meerdere mogelijkheden voor het modelleren van communicatie (bv. push vs. pull) en applicaties (bv. het mirror server model). Deze diversiteit is hinderlijk voor softwareontwikkelaars die horizontale diensten wensen te ontwikkelen die (gelimiteerde) geconnecteerde objecten combineren uit meerdere systemen. Dit proefschrift presenteert een cloud platform ter bevordering van de integratie in services van heterogene (gelimiteerde) IoT toestellen en communicatiemodellen door middel van een uniforme, op open standaarden gebouwde toestelabstractie. Een driedelige evaluatie toont aan dat het platform slaagt in het verminderen van communicatievertraging, in het verbergen van heterogene communicatiemodellen en dat het platform eenvoudig kan geïntegreerd worden in externe systemen.

In zijn voortdurende groei, breidt het WoT uit naar nieuwe connectiviteitstechnologieën. Low Power Wide Area Networks (LPWANs) zijn een voorbeeld van een dergelijke technologie die belooft om connectiviteit voor enorme hoeveel-

heden (gelimiteerde) apparaten te voorzien. Terwijl de lpwan IETF werkgroep het gebruik van IPv6 en CoAP over LPWANs standaardiseert, is het belangrijk om de vermeende voordelen van deze nieuwe technologieën kritisch te bestuderen. Meer bepaald onderzoekt dit proefschrift de bewering dat LoRa Wide Area Networks (LoRaWANs), een populaire LPWAN technologie, bidirectionele communicatie kan voorzien voor vele duizenden toestellen per gateway. Door middel van de gedetailleerde modellering van LoRaWAN in een netwerksimulator, werd de schaalbaarheid van LoRaWAN onderzocht. Uitgebreide simulaties tonen dat de radio-duty cyclus beperkingen van gateways nefast zijn voor het afleveren van berichten met bevestiging in grootschalige netwerken, dit omwille van het ontbreken van tijdige downstream bevestigingen. Hoewel het verhogen van het aantal gateways de aflevering kan verbeteren, blijven de strenge radio-duty cyclus beperkingen van gateways problematisch. Omwille van deze reden dient er karig omgesprongen te worden met berichten met bevestiging in grootschalige LoRaWAN netwerken.

Samengevat presenteert dit proefschrift verscheidene methodes ter bevordering van de integratie van gelimiteerde WoT toestellen in applicaties en diensten. Het onderzoek bestudeerde de onderwerpen van efficiëntie, bruikbaarheid en schaalbaarheid van gelimiteerde RESTful omgevingen. Dit proefschrift kan beschouwd worden als een kookboek met omvangrijke recepten voor het oplossen van operationele problemen in gelimiteerde RESTful omgevingen. Het distributed intelligence concept en de virtualisatie aanpak zijn de hoekstenen van dit werk. Het proefschrift omschrijft en bewijst hun waarde in de context van CoRE door het filteren van (ongewenst) verkeer, het combineren van resource-aanvragen, het toevoegen van virtuele resources en andere functionaliteit, het verbeteren van authenticatie en autorisatie en het efficiënter inzetten van beperkte middelen. Bijgevolg heeft dit doctoraatsonderzoek een bescheiden maar significante bijdrage geleverd aan het doel van een open en veilig WoT waar vele, heterogene (gelimiteerde) toestellen naast elkaar bestaan en succesvol samenwerken.

Summary

The Internet of Things (IoT) promises to revolutionize numerous business processes by interconnecting (everyday) objects, collecting information about our environment and taking and enacting decisions based on this information. Through big data analysis, it is expected that new information can be drawn from the flood of data produced by interconnecting things. Such information will enable us to more thoroughly understand our environment and to make more informed decisions. Additionally, it is anticipated that the IoT will lead to new business models due to the low cost monitoring and actuation offered by connected products. In manufacturing, connected machines are expected to further improve automation and boost efficiency. In insurance, connected things are expected to lead to improved safety, security and loss prevention. In healthcare, connected products will offer new health and fitness services thereby aiding in reducing cost. The IoT is expected to impact many other domains as well, including transportation and logistics, building automation and smart home, utilities, agriculture and many others.

Today, many connected products are designed to operate in a specific vertical market and to operate in a closed ecosystem. Often vendors are even reluctant to support integration with third parties, as this is perceived as a loss in competitive position and therefore in value. This fragmentation is a major issue as it hinders horizontal platforms from entering the market. A major benefit of these horizontal platforms is the integration of a variety of different devices, regardless of the model, the manufacturer and the application domain of the device. The seamless interoperability required for such horizontal platforms, is vital to the long term success and effectiveness of the IoT. In many ways, the fragmented state of the IoT is comparable to the early days of the Internet where many packet switched networks were competing for market share.

The Web of Things (WoT) has been proposed as a solution to this problem of fragmentation. It takes the successful web standards, technology and tools of today's World Wide Web (WWW) and applies them to interconnecting things. In the WoT paradigm, things expose their services and data over RESTful APIs regardless of their make, vendor, application domain or connectivity technology. As building RESTful APIs with the traditional TCP/HTTP protocols has proved challenging and often infeasible for resource-constrained devices, the Internet Engineering Task Force (IETF) has standardized the Constrained Application Protocol (CoAP), which brings RESTful embedded web services to resource-constrained IoT devices. Typically such devices are subject to limitations on their size, financial cost, processing power, memory size, connectivity and energy source. The

result is referred to as Constrained RESTful Environments (CoRE), which is the subset of the IoT that focuses on resource-constrained devices employing the Representational state transfer (REST) paradigm.

While the standardization of CoAP is a crucial stepping stone to realizing the resource-constrained WoT, it has also introduced new and interesting questions when deploying CoAP in resource-constrained networks and devices. This dissertation studies challenges and adoption barriers related to the integration of resource-constrained devices in the WoT. As such the main goal of this PhD has been to solve open issues and remove barriers for the adoption of open standards in CoRE. By providing technical solutions to these problems, this dissertation hopes to increase the adoption rate of open web standards in the resource-constrained IoT. Only by relying on open standards based systems can the IoT move from its fragmented state today to a future where everything is interconnected.

A first set of challenges relates to the impact of connecting everything to everything in an open WoT on the operation of resource-constrained systems. As the growth of the IoT continues, one has to be careful that the open nature of the WoT does not impede the operation of CoRE. Indeed, as the IoT grows, the load on resource-constrained systems will grow as well and the need for efficient utilization of the limited resources will grow more stringent. Coupled to this growth, is the need for holistic and scalable security mechanisms in CoRE that offer authentication, authorization, shielding of sensitive information and confidentiality. Some of these security aspects are difficult to obtain in CoRE due to the limited resource available. This dissertation presents Distributed Intelligence (DI) as a method for solving these technical challenges. DI recognizes that processing may be distributed throughout the Internet and as such may assist resource-constrained systems. More specifically, device virtualization enables non-constrained systems to extend resource-constrained devices with new features that assist in efficient resource utilization and holistic security. The Secure Service Proxy (SSP) presented in this work deploys adapters on virtual devices in order to extend resource-constrained devices with new features. Adapters are modular blocks of functionality that operate on RESTful requests and responses (via CoAP). The evaluation of the SSP shows that the proxy brings scalable authentication and fine-grained authorization to resource-constrained devices, while reducing the costs of session-based security in terms of power consumption and communication delay. It also shows that the SSP manages to reduce resource consumption and increase network capacity by combining resource requests.

Another group of adoption barriers stems from the limited usability of resource-constrained devices in CoRE. For end users, this limited usability manifests itself in the lack of a User Interface (UI) for interacting with resource-constrained devices. This dissertation presents a Graphical User Interface (GUI) rendering agent by means of web templates, thereby enabling users to interact with constrained devices via well-known web technology through their (mobile) browser. From the point of view of developers the limited usability pertains to missing features on resource-constrained devices that are taken for granted on conventional web systems. For example, resource constraints make it difficult to offer semantic descrip-

tions of web services and hinder devices from adapting to long term changes as firmware updates might be cumbersome. To address these challenges, the aforementioned device virtualization technique was applied to offer new functionality (e.g. emulate resource observation and offer semantic resource descriptions). This dissertation applied resource virtualization to implement missing features and to update the functionality of certain resources over time without the need for expensive update mechanisms on resource-constrained devices.

The diversity in connectivity technologies, communication models and application modeling techniques forms the third issue addressed in this dissertation. As many (legacy) technologies will continue to exist in the IoT, it will be needed to integrate these (legacy) technologies with CoRE. Additionally, within CoRE there also exists some freedom in modeling communication (e.g. push vs pull) and structuring applications (e.g. the mirror server model). This diversity is troublesome for service developers wanting to integrate resource-constrained devices from multiple systems. To address this problem, this dissertation presents a cloud-based platform to facilitate the integration of heterogeneous constrained IoT devices and communication models into services by means of a uniform, open standards-based device abstraction. A threefold evaluation demonstrates that the platform improves latency, is effective at hiding heterogeneous communication models and is easy to integrate into services developed by third parties.

In its continuous growth, the WoT is expanding to new network technologies. Low Power Wide Area Networks (LPWANs) are an example of such a new technology and promise to bring low data rate connectivity to vast amounts of (resource-constrained) devices. While the lpwan IETF working group is standardizing the use of IPv6 and CoAP over LPWANs, it is important to evaluate the benefits claimed by these LPWANs. Specifically, this dissertation studies the claim that LoRa Wide Area Networks (LoRaWANs), a popular LPWAN technology, may provide bidirectional connectivity to many thousands of devices per gateway. Via detailed modeling in the ns-3 network simulator, this dissertation has studied the scalability claims of LoRaWAN. Extensive simulations have shown that the radio duty cycle restrictions of gateways are detrimental to the delivery ratio of confirmed messages, due to the inability to send timely downstream acknowledgments. While increasing the gateway density does show a significant increase in packet delivery ratios of confirmed messages, it can not compensate completely for the stringent duty cycle restrictions of gateways. As a result, confirmed messages should be used only sparsely in large scale networks.

To conclude, this dissertation presents several methods for improving the integration of resource-constrained WoT devices into applications and services. The research focuses on the topics of efficiency, usability and scalability of resource-constrained RESTful environments. This dissertation can be considered as a cookbook with extensive recipes to address operational concerns of Constrained RESTful Environments. Specifically, the distributed intelligence concept and the device virtualization approach are the two cornerstones of this work. This dissertation describes and shows the value they can add in (secure) CoRE by filtering (unwanted) traffic, combining resource requests, adding virtual resources and other function-

ality, improving authentication and reducing resource usage. In doing so, this PhD made a modest but significant contribution to the goal of an open, secure WoT, where many, heterogeneous (resource-constrained) devices co-exist and interoperate.

1

Introduction

“I think it’s much more interesting to live not knowing than to have answers which might be wrong. I have approximate answers and possible beliefs and different degrees of uncertainty about different things, but I am not absolutely sure of anything and there are many things I don’t know anything about, such as whether it means anything to ask why we’re here. I don’t have to know an answer. I don’t feel frightened not knowing things, by being lost in a mysterious universe without any purpose, which is the way it really is as far as I can tell.”

– Richard Feynman (1918 - 1988)

1.1 Background

This section introduces the technical background and terminology that are used throughout this dissertation. As the Internet of Things (IoT) has become a broad concept, it is clarified and defined in alignment with the research presented in this dissertation. Next, the subset of the IoT that focuses on resource-constrained systems and networks is introduced. The background then continues to discuss the Web of Things (WoT) and the Internet standards for the resource-constrained WoT which are relevant to this dissertation. Finally, Low Power Wide Area Networks (LPWANs), a novel network technology, are introduced.

1.1.1 Internet of Things

The Internet of Things term is traditionally attributed to Kevin Ashton who, while working on supply chain management at Procter & Gamble (P&G), coined the term in 1999. As part of a supply chain management system, Ashton equipped items with RFID tags and coupled the RFID system to the Internet. While the term was only introduced in 1999, already nine years earlier in 1990 - just one year after the introduction of the World Wide Web (WWW) - John Romkey created a toaster that could be turned on and off over the Internet using TCP/IP. Of course back then the Internet was *very much* smaller than it is today and repeating the same exercise today would probably lead to your toaster being targeted by malicious parties within minutes ¹. In the beginning of the previous decade, the IoT started gaining mainstream media attention with the proposition of an RFID alternative to barcodes, (at the time futuristic) Internet refrigerators and the advent of small sensors equipped with microchips and RF transceivers. Noteworthy in the research community, was the first European IoT conference which was held in March 2008.

According to Cisco, one of the largest vendors in IT and networking products, the IoT was born somewhere between 2008 and 2009 when more things were connected to the Internet than people. In industry, a group of companies founded the IPSO alliance in 2008 to promote the use of the Internet Protocol (IP) in networks of smart objects. The year 2011 saw the public launch of IPv6, the successor to the IPv4 protocol, by large companies such as Facebook, Google and Akamai on January 12th: the World IPv6 Day. Since the beginning of this decade, big players such as Intel, Cisco, Google and Amazon have jumped on the IoT bandwagon by launching new company departments, acquisitions and other investments. Today, the strategy of every big tech company takes the IoT into consideration. This is primarily motivated by the large potential for growth in this market. A global IoT market report by Gartner [1] forecasts the economic value-add (which represents the aggregate benefits that businesses derive through the sale and usage of IoT technology) to be \$1.9 trillion across sectors and the incremental revenue stemming for IoT suppliers to exceed \$300 billion in 2020. The biggest sectors are expected to be manufacturing, healthcare and insurance. The report expects an accelerated growth in the number of connected things, reaching 26 billion IoT units by 2020². Anticipated innovations include improved safety, security and loss prevention in the insurance industry as well as new business models based on real-time data. The healthcare sector is anticipated to benefit significantly from a large range of health and fitness services supported by the IoT. Finally, connected sensors are expected to lead to value creation in utilities, transportation and agriculture by improving efficiency.

¹<http://www.zdnet.com/article/iot-devices-can-be-hacked-in-minuteswarn-researchers/>

²A 2015 forecast by Gartner [2] has corrected the number of connected things to 20.8 billion connected devices by 2020.

Apart from the three sectors briefly discussed above, IoT products are expected to be found across a wide range of different application domains. An overview is presented in figure 1.1. Buildings (homes, offices) will be equipped with IoT technology in order to reduce the consumption of resources (e.g. electricity, heating), to improve the experience of people (e.g. via automation) and to reduce operational expenditures (e.g. by more efficient processes, such as predictive maintenance). In smart cities, IoT technology can provide monitoring and actuation for a city's infrastructure (power, water and road networks) in order to increase the efficiency of these networks and can improve the quality of life of the city's citizens (e.g. by pollution and noise monitoring, traffic control, parking monitoring, etc.). Utility companies will rely on smart metering and smart grids to provide detailed monitoring of energy production (by wind, solar, etc.) and consumption (in homes, offices, vehicles, factories) in order to balance supply and demand as society continues to move to renewable, but volatile sources of energy production³. In healthcare, connected medical products may be used to monitor a patient's status remotely (thereby reducing load on medical personnel and infrastructure) and to assist elderly in living independently at home. 'Quantified self' is another example of eHealth where smart wearables allow to monitor one's life style, habits and health.

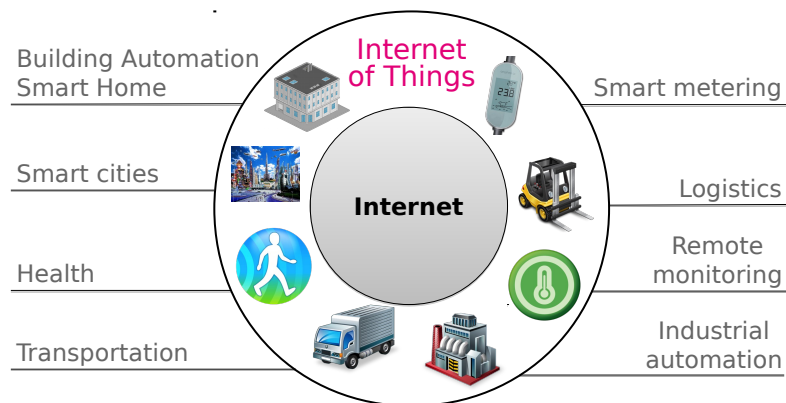


Figure 1.1: Application domains of the Internet of Things (image courtesy of the IoT course at Ghent University)

In the transport and logistics sector, connected trackers provide tracking and tracing of goods. Similarly, the status of the vehicular fleet and other equipment, the behavior of drivers and maintenance can be tracked more accurately in order to improve efficiency and reduce operational costs. Connected and autonomous

³In February 2017, the Flemish made the controversial announcement to start rolling out smart electricity meters in 2019 over a period of fifteen years.

cars promise to increase road safety, while reducing traffic congestion and pollution. In manufacturing, connected products and increased automation enable factories to monitor and collect real-time information from production lines and supply chains. This in turn allows for more informed and faster decisions in order to optimize production or to perform predictive maintenance. In agriculture, soil and crop sensors monitor environmental conditions and allow more effective and efficient fertilization. In animal farms, animal trackers aid in detecting and preventing diseases by monitoring the health status of livestock. Such initiatives help to increase a farm's output, while lowering costs by reducing waste, labor and livestock sickness. Finally, in the retail sector, retailers can shape their outlets and advertisements to meet the expectations of their customers by tracking shopping behavior and individual product sales via product monitoring.

It is clear that the IoT presented above is a diverse field and that many different incarnations will co-exist. Since its inception in 1999, the IoT has continuously grown in scope to the point where today it has become an ambiguous term with different meanings to different parties. Indeed, there exist over thirty definitions of the IoT stemming from research reports, white papers, books, etc ⁴. In the interest of clarity, this dissertation narrows down the IoT by adopting the following definition proposed by Morgan Stanley in [3]:

“The Internet of Things is the combination of sensors, actuators, distributed computing power, wireless communication on the hardware side and applications and big data/analytics on the software side.”

This definition was chosen because, apart from big data, it highlights the technical aspects of the IoT that are relevant to this dissertation. According to this definition, sensors quantify our surroundings and transmit their measurements wirelessly to interested applications via the Internet. Applications transform data into knowledge about the physical world, which is used to make decisions and drive a wide range of processes (as briefly discussed for the different application domains above). Apart from monitoring, our environment may also be affected via actuators, which are connected to the Internet. The term distributed computing power in the definition refers to the ability that IoT software may reside on any Internet connected system, be it a cloud-based system, a gateway system or a system deployed in the fog. This dissertation further narrows down the scope of the IoT by focusing on the resource-constrained WoT. These terms are discussed in the following two sections.

1.1.2 The resource-constrained Internet of Things

The work in this dissertation focuses on the resource-constrained Internet of Things. Computing systems belonging to this category are designed specifically for par-

⁴The interested reader may find a list on <https://www.postscapes.com/internet-of-things-definition/>

particular applications, which set them aside from other general-purpose computing systems (such as mobile phones and desktop computers). As these systems fulfill one specific function, hardware and software are often designed together to be proficient at that one specific function. Due to their close integration with our environment and their connectivity, they are also known as networked embedded systems. Despite their specialized nature, some degree of flexibility of network embedded systems is still required as bug and security fixes and adapting to changes in their environment are essential in order to guarantee their long-term operation.

In the IoT, our surroundings will be permeated with sensors and actuators. In this vision, networked embedded systems have to be produced at a massive scale and low cost. In order to keep unit cost low, the amount of resources available to these systems is limited which leads to the term ‘resource-constrained’ systems. RFC 7228 [4] lists the following limitations as common constraints: limited processing power, limited maximum code complexity, limits on the size of state and buffers, limits on available power and limited user interface and accessibility in deployment. This standard also introduces the term ‘constrained nodes’ to refer to resource-constrained networked embedded systems. This dissertation uses the terms ‘constrained nodes’ and ‘constrained devices’ interchangeably. When



Figure 1.2: Resource-constrained sensing devices from the WSN research community.

these constrained devices form a communication network, this network in turn also exhibits constraints such as unreliable communications, low and fluctuating throughput, unstable topologies, asymmetric link characteristics and others. Such networks are known as ‘constrained-node networks’, but this text often refers to such networks as ‘constrained networks’. Finally, a classification of constrained devices based on their characteristics is also presented in RFC 7228. The work in this dissertation focuses on class 1 devices, which are defined as having data sizes (RAM) in the order of 10 KiB and code sizes (ROM) in the order of 100 KiB. While these devices are unable to house a traditional, full-fledged Internet protocol stack, they do have sufficient resources to run specialized Internet stacks with lightweight protocols that have low memory footprints and low parsing complexity. As a point of reference, modern notebooks ship with working memory sizes ranging between 4 and 16 GiB, which is roughly six orders of magnitude larger than the data size at the disposal of class 1 devices. Class 1 devices are interesting as they are characterized by having a very low power usage and very low cost. Indeed, RFC 7228 mentions that gains made available by increases in transistor

count and density due to Moore's law are more likely to be invested in reductions of costs and power requirements rather than into continual increases in computing power. Minimizing power usage is crucial, as energy storage elements (e.g. batteries) make up a considerable fraction of the cost of class 1 devices.

1.1.3 TCP/IP for the resource-constrained Internet of Things

In terms of communication protocols, the baseline for the IoT is the Internet Protocol. IP has proven to be an open, interoperable, scalable, ubiquitous and stable technology with billions of connected devices since its inception in the nineteen-eighties. The plethora of services running over the IP today is a testament to the versatility of the protocol. The end-to-end design of the IP differs from other (legacy) technologies where complex, vendor-specific gateways map applications to the Internet. More recently, in 2011, IANA, the body in charge of the IPv4 addressing space, allocated the last two unreserved /8 address blocks to regional bodies. Since then four of the five regional bodies have also exhausted their allocation space, with AFRINIC estimated to reach exhaustion in 2018. With the standardization of version six of the IP (IPv6), the address space has been hugely extended from 4.3 billion to 3.4×10^{38} addresses. For comparison, the number of grains of sand on all beaches in the world is estimated to be *only* 7.5×10^{18} , the number of stars in the universe is estimated to be in the order of 10^{22} and on earth we would be able to assign approximately 667×10^{38} addresses per square meter of surface area. Clearly, one does not have to worry about address exhaustion in IPv6 in the near future. While the adoption rate of IPv6 has been slow in the past, it has accelerated in recent years now that IPv4 has reached address exhaustion.

In networked applications, the Transmission Control Protocol (TCP) is a common protocol for transporting a stream of octets between applications running on Internet hosts (e.g. all web traffic runs over TCP). Early work [5] [6] identified the inability of TCP to distinguish between losses due to congestion and other losses as troublesome in wireless networks, where lossy links might be caused by bit errors, link breakage or handoffs rather than congestion. This leads to degradation in TCP throughput in wireless networks. Despite these issues [7], machine-to-machine communications via TCP/IP is commonly used in cellular GPRS networks today⁵. Despite its popularity in cellular M2M networks, the throughput of TCP is lower than that of User Datagram Protocol (UDP) in low-power wireless networks where the bidirectional communication required by TCP may be hindered by multi-hop and asymmetric links [8]. Work by Adam Dunkels et al. [9] [10] has illustrated the feasibility of implementing a highly optimized TCP/IP stack on memory-constrained sensor nodes and indeed Dunkel's implementation, uIP, remains one of the most popular network stacks in wireless sensor and actuator

⁵Note that some countries have plans to phase out their 2G cellular networks.

networks today [11]. In the same work [10], the authors identify the following issues with TCP/IP in sensor networks: very large header overhead for small packets, bad performance over links with high bit-error rate (e.g. wireless links) and energy consumption at every hop of the retransmission path in the end-to-end retransmissions. Some of the issues with TCP/IP in wireless (sensor) networks have been the focus of later work at the Internet Engineering Task Force (IETF), as will be discussed in section 1.1.5.

1.1.4 Web of Things

On the Internet, the TCP/IP stack is often used in tandem with the Hypertext Transfer Protocol (HTTP) when developing web-based applications. The success of web services on the Internet has drawn researchers to apply similar techniques for developing applications in wireless sensor and actuator networks [12] [13]. These works demonstrated that web services may be used to build highly-interoperable systems where multiple applications may share a common sensor network. The authors also highlighted that a key challenge in using web services on resource-constrained sensor nodes is the energy and bandwidth overhead of the structured data formats used in web services. Specifically, TCP/IP and structured XML data increases the number of messages as well as the message size and hence the energy consumption [13]. The authors suggest a number of improvements to mitigate some of these issues, but the header overhead and verbose data formats remain an issue in low-power networks.

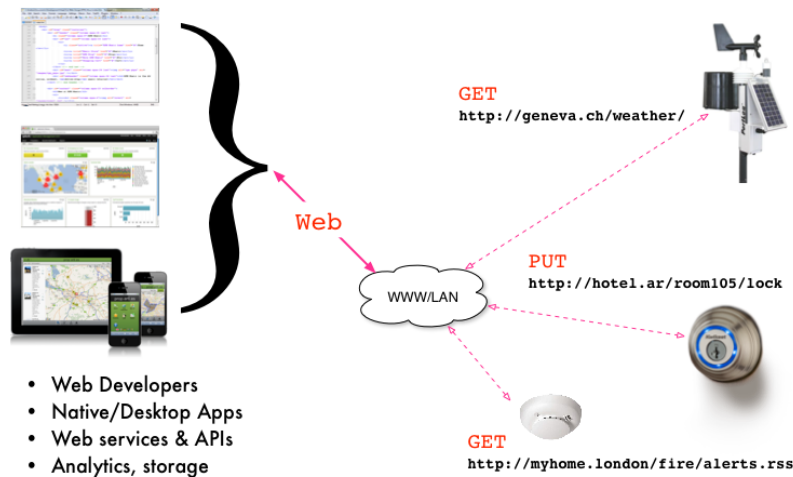


Figure 1.3: In the WoT, applications interact via RESTful requests and responses over the Internet regardless of the underlying network technology (image courtesy of 'Building the Web of Things')

Around the same time, applying the Representational state transfer (REST) principle [14] for integrating physical things into the Web has been proposed in [15] [16] and [17], leading to the term the ‘Web of Things’. In the WoT philosophy, resources on things are available through standard Web mechanisms. Figure 1.3 provides an example where a weather station and a fire alarm expose RESTful resources with unique URIs that can be queried to retrieve their readings. Similarly, a door lock offers a resource for manipulating its locked status. The author of [15] argues that the big advantage of the WoT is that things can be treated like any other Web resource, which allows them to be used in different contexts and applications. This is in stark contrast to exposing real-world data and functionality through proprietary Application Programming Interfaces (APIs), which leads to tightly-coupled systems and often limits interactions to a very specific set of potential users [16] [17]. In the WoT, things can be used in a much more open way, similar to web services on the traditional Internet. The authors of [16] state that “In the WoT, popular web languages (e.g. Python, PHP, node.js) can be used to easily build applications involving smart things and users can leverage well-known Web mechanism (e.g. browsing, searching, caching and linking) to interact with and share these devices”. The versatility of the WoT is illustrated on the left in figure 1.3, where different types of applications employ web technology to communicate with things. These works laid the foundation for the Web of Things, with the caveat that the HTTP/TCP/IP protocols common at the time may be too chatty and too complex and that traditional data formats may be too verbose for use in low-power networks.

Today the WoT is an active field of research, with e.g. annual international workshops [18] and the recently chartered World Wide Web Consortium (W3C) WoT Working Group (WG) ⁶. The WoT WG is in the process of standardizing a WoT architecture, a Web Thing description and a programming interface for running scripts on Web Things. Additionally, the WG is chartered to provide examples of Thing descriptions for common platforms and protocols. Some members of the WG are also working on a model of a Web Thing ⁷ that describes methods for integrating Things in the Web with a focus on HTTP, WebSocket, JSON and JSON-LD. This model, shown in figure 1.4, represents a Web Thing as an onion, where each shell adds additional requirements. Extended Web Things additionally support the REST API and data model defined in the Web Thing Model. As a result, Extended Web Things can be automatically included in complex systems. The Web Thing Model suggests implementing semantic extensions via JSON-LD and schema.org. While the Web Thing Model document focuses on HTTP, the document states that other RESTful protocols, such as CoAP (see further), may be used as well.

⁶<https://www.w3.org/2016/12/wot-wg-2016.html>

⁷<http://model.webofthings.io>

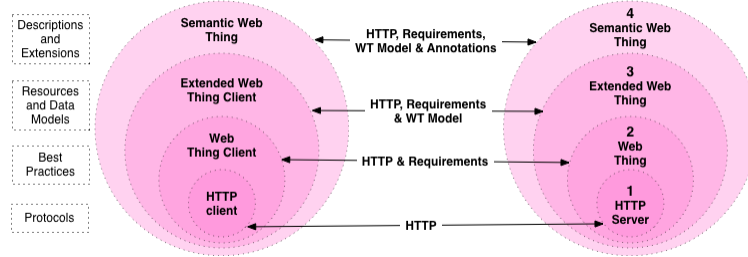


Figure 1.4: In the Web Thing Model, Web Things are categorized into four different levels depending on their functionality (image courtesy of <http://model.webofthings.io/>)

1.1.5 Internet standards for web technology in constrained devices and networks

As detailed in the previous sections, the large overhead of TCP/IP headers in wireless sensor networks leads to wasting energy in low-power networks. Additionally, the end-to-end retransmission scheme in TCP meant that some segments would be transmitted more than once over low-power links, further increasing energy usage. Furthermore, with the introduction of IPv6 the problem of header overhead became even more severe now that the IP header grew from 20 to 40 bytes to accommodate the 128 bit addresses. In order to address these issues in a global forum, various initiatives were started at the IETF, i.e. the Internet standardization body.

1.1.5.1 IPv6 over Low Power Wireless Personal Area Networks

In 2005, the IPv6 over Low power WPAN (6LoWPAN) WG was chartered to investigate the use of IPv6 in IEEE 802.15.4 Wireless Personal Area Networks (WPANs). Specifically, the WG defined packet formats for the ‘Transmission of IPv6 packets over IEEE 802.15.4 WPAN Networks’ in Request for Comments (RFC) 4944 [19] and 6282 [20]. These standards define an adaptation layer which implements framing, header compression, address generation and fragmentation [21]. The header compression scheme can in some cases compress the IPv6 header down to only four bytes and was later extended to also include UDP header compression. The fragmentation scheme satisfies the IPv6 minimum MTU requirement of 1280 octets. Furthermore, the WG optimized the IPv6 neighbor discovery procedure for use in low-power WPANs in RFC 6775 [22]. A comprehensive technical overview of 6LoWPAN is presented in [23]. In the research community Durvy et al. presented the IPv6 compatible successor of uIP, named uIPv6, and released it as open source [24]. Within the year, uIPv6 was extended with an implementation of 6LoWPAN named sicslowpan [25]. In [26] the author

shows how 6LoWPAN header compression reduces the UDP/IPv6 header overhead to 7 bytes for link-local communication and to between 12 and 28 bytes for global communications⁸. Thread, the home automation protocol launched by Google, Samsung, ARM and others in 2014, builds on top of 6LoWPAN. Atmel, a large semiconductor manufacturer, has been shipping 6LoWPAN as part of their ultra-low power wireless hardware platforms since 2013. ARM has been shipping 6LoWPAN as part of the networking stack of their mbed platform for Cortex-M microcontrollers since 2014. Note that the 'IPv6 over Networks of Resource-constrained Nodes (6lo)' IETF WG is continuing the work of the 6LoWPAN WG for other link-layer technologies than IEEE 802.15.4 WPANs.

1.1.5.2 The Constrained Application Protocol

In 2010, the Constrained RESTful Environments (CoRE) IETF WG was chartered to provide a framework for resource-oriented applications intended to run on constrained IP networks. Targeted applications included monitoring of simple sensors (e.g. temperature sensors), controlling actuators (e.g. light switches, Heating, Ventilation and Air Conditioning (HVAC) control) and device management [27]. Key design principles included embracing the REST philosophy, mandatory support for the UDP transport protocol, compact header and message formats, asynchronous messaging, multicast support, resource discovery and an application-independent mapping from CoAP to an HTTP REST API. Since its inception, the WG has published seven RFCs, with more in the pipeline. The WG has been very active with numerous interoperability testing campaigns (known as plugfests) and WG meetings in order to reach a broad consensus on various protocol design issues.

RFC 7252 [28] standardizes the Constrained Application Protocol (CoAP) and defined message models, message formats and serialization, option encoding, message (re)transmission schemes, request/response codes, URI schemes, multicast support, security considerations and HTTP-CoAP cross-protocol proxying. Logically, CoAP is comprised of two layers: a messaging and a request/response layer. The messaging layer deals with UDP and the asynchronous nature of interactions, while request/response layer interactions use method and response codes. As UDP has no built-in reliability nor deduplication, CoAP defines a deduplication and an optional, lightweight retransmission scheme as part of the messaging layer. For improved compactness, the CoAP header is binary encoded as opposed to the US-ASCII character encoding used in HTTP headers. Additionally, options in the CoAP header use a delta encoding scheme to improve the compactness of the CoAP header even further. Similarly to HTTP, CoAP defines request methods and response codes that enable the REST paradigm. It also defines a media type

⁸An uncompressed UDP/IPv6 header is 48 bytes long

registry in order to provide an enumeration of common media types. In terms of security, the CoAP specification mentions both transport (see further) and object security. A thorough overview of CoAP is presented in [23] and [29]. Note that relevant CoAP principles are introduced where necessary in the following chapters.

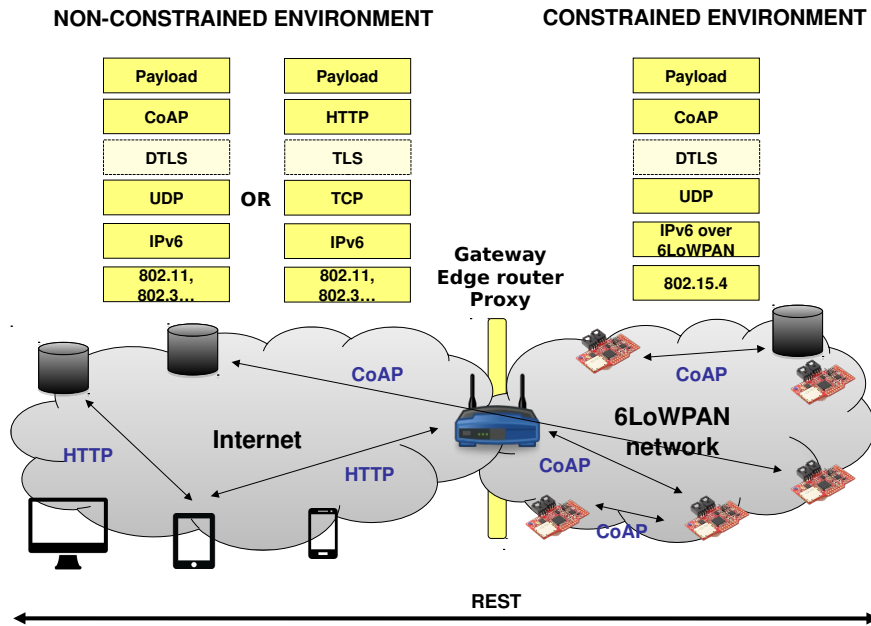


Figure 1.5: Internet protocols and REST architecture for the resource-constrained Web of Things

Figure 1.5 present an overview of the REST architecture resulting from the deployment of the aforementioned Internet standards in a 6LoWPAN network. Constrained devices are running IPv6 via the 6LoWPAN adaptation layer underneath CoAP over UDP or DTLS. A cross protocol-proxy translates between the unconstrained and the constrained world, mapping TCP/HTTP to UDP/CoAP. In parallel, Internet hosts are able to interact directly with constrained devices in the constrained network via CoAP. These Internet hosts can be both unconstrained (e.g. cloud management platform) and constrained systems (e.g. switch activating a light bulb). The use of transport layer security is optional and use case dependent. For constrained networks other than 6LoWPAN, the physical and the adaptation layers would be different, while the other layers remain the same.

Since the standardization of CoAP, a number of interesting extensions have been published by the working group. RFC 6690 [30] defines a web linking format for use in constrained environments. The CoRE link format enables web servers to

describe hosted resources, their attributes as well as relationships between links. The observe protocol extension [31] enables clients to retrieve a representation of a resource and keep this representation updated by the server over a period of time. Due to the absence of such an extension in HTTP, inefficient methods such as HTTP long polling were hacked in later on. Using CoAP block-wise transfer [32], large CoAP messages may be split into multiple messages thereby avoiding (potentially expensive) lower layer fragmentation or circumventing limitations on buffer sizes. RFC 8075 [33] provides guidance for implementing a cross-protocol that translates the HTTP protocol to the CoAP protocol. Specification [34] defines how CoAP should be used in a group communication context. Finally, PATCH and FETCH, two new request methods that allow partial resource manipulation and access respectively, are defined in RFC 8132 [35].

At the time of this dissertation the CoRE WG has adopted a total of ten WG drafts, which are in various stages of the standardization process. One mature draft is the CoRE Resource Directory [36], which is a service that hosts descriptions of resources hosted on other servers thereby supporting resource and device lookups. The ‘Reusable Interface Definitions’ draft is another long-time draft that defines a set of resource interface descriptions applicable for use in constrained environments [37]. In an effort to introduce concise data formats, the WG has adopted the ‘Media Types for Sensor Measurement Lists’ draft [38]. A similar effort is the standardization of the Concise Binary Object Representation (CBOR) [39], which follows a JSON data model and has an extremely low code size and fairly small messages sizes. Drafts that were recently adopted by the WG include a publish/-subscribe broker over CoAP, simple congestion control for CoAP and link bindings in CoAP (similar to bindings in ZigBee).

Kuladinithi et al. present the open source libcoap library and compare its performance to that of various HTTP clients in [40]. An experimental evaluation performed over a GPRS network showed that UDP-based protocols outperform TCP-based protocols in constrained networks due to using a lower number of messages when retrieving resources. In the case of UDP, CoAP packets were 18% smaller than HTTP packets while CoAP adds reliability and request/response matching when compared to HTTP over UDP. In [41] Ludovici et al. present TinyCoAP, a coap implementation optimized for TinyOS, and compare its performance to that of TCP/HTTP and UDP/HTTP. In terms of memory occupation TinyCoAP uses slightly more RAM than the TCP/HTTP implementation (due to at compile time memory allocation), while ROM usage is significantly lower. The authors show that TinyCoAP outperforms TCP/HTTP in terms of delay and energy usage, even when persistent TCP connections are used. In terms of delay and energy usage, the performance of UDP/HTTP is close to UDP/CoAP but as previously mentioned UDP/HTTP is lacking reliability, deduplication and request/response matching. In [42] Kovatsch et al. present a system architecture for CoAP-based

IoT cloud services. Surprisingly, the Californium cloud framework shows 33 to 64 times higher throughput than state-of-the-art HTTP web servers common in conventional cloud services. The authors state that the low overhead of CoAP significantly improves backend service scalability for vast numbers of connected devices. In industry, ARM's mbed platform has added CoAP support in recent years. Thethings.io, an IoT platform for deploying scalable and flexible IoT solutions, also supports CoAP endpoints as things.

Historically, CoAP has been deployed mostly over conventional wireless technologies such as IEEE 802.11 (known as Wi-Fi), IEEE 802.15.4 and cellular M2M networks. As the IoT continues to broaden in scope, new wireless connectivity technologies emerge which has triggered the community to apply the principles of CoRE to these new networking technologies. The next subsection considers one example of such a new and promising network technology: LPWANs.

1.1.6 Low Power Wide Area Networks

In an effort to bring affordable connectivity to low-power devices spread over very large geographical areas, new network technologies known as Low Power Wide Area Networks (LPWANs) are gaining considerable momentum. LPWANs promise to bring Machine-to-Machine (M2M) communication to use cases where conventional cellular and short range wireless technologies were unsuited due to cost, range or power limitations. Promising applications include smart cities and track&trace logistics. Some of the main differences between LPWANs and WPANs (such as the aforementioned IEEE 802.15.4) are the longer range, the lower data rates, the smaller maximum packet sizes and the lower power requirements common in LPWANs. Additionally, most LPWA technologies adopt a model where the network infrastructure (gateways, backend systems) is in the hands of a network operator.

One issue with LoRa Wide Area Network (LoRaWAN) and LPWANs in general, is that these technologies do not connect their Things to the Internet. Instead, it is common to define an interface for two-way data access which is very similar to the APIs offered on WSN gateways. This hinders the usability of low-power devices connected via LPWANs in new application and services. Figure 1.6 illustrates the network architecture for popular LPWAN technologies such as LoRaWAN and SigFox. In this architecture services have to integrate LPWAN specific (and sometimes even network specific) APIs in order to communicate with LPWAN devices.

Very recently, interest in bringing IPv6 to LPWA networks has started to grow. Since last year, work is underway at the IETF 'IPv6 over Low Power Wide-Area Networks (lpan)' WG to accomplish just that. This is reminiscent of the efforts of the 6LoWPAN WG to enable IPv6 in IEEE 802.15.4 networks. So far the

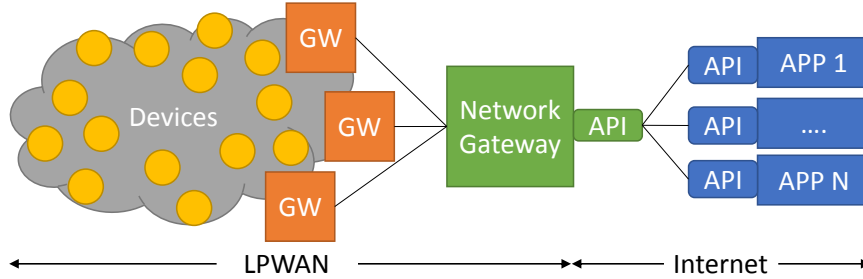


Figure 1.6: Common architecture of LPWANs: network gateways offer LPWAN-specific APIs in order to integrate LPWAN devices into applications and services

lpwan WG has adopted a header compression scheme and fragmentation method for IPv6 and UDP for use in various LPWA networks [43]. The group has also adopted a draft that applies the same compression scheme, named Static Context Header Compression (SCHC), to CoAP [44]. Initial tests show a drastic reduction in packet lengths, as in the best case SCHC is able to compress an IPv6/UDP header to only one byte. One open issue with SCHC is the efficient management of the static context (i.e. compression rules) on the LPWA devices. SCHC will likely only be effective in cases where changes to the static context are very infrequent, e.g. in scenarios where the communication of the LPWA device is limited to a fixed number of Internet hosts.

1.2 Research challenges

This dissertation tackles a number of research challenges, all of which stem from the limitations inherent to resource-constrained devices and networks. When tackling these challenges, the perspective of Constrained RESTful Environments, as sketched in the background section, is taken. As such, the main focus of this dissertation has been to solve a number of open issues faced in CoRE in order to remove barriers to adopting an open and interconnected WoT.

1.2.1 Efficient resource utilization in an open Web of Things

In order for an open WoT, where everything interacts with everything, to be feasible and indeed successful in constrained environments a number of potential pitfalls and problems have to be anticipated and solved. One challenge relates to the efficient utilization of the limited resources in low-power networks. Specifically, one should avoid unnecessarily depleting resources (e.g. computation, energy, radio time) in constrained components where it can be avoided. Inefficient resource usage could hamper the operation of constrained components and the scalability

of constrained networks. Indeed, considering the enormous growth of the IoT one has to be careful that the wanted, open character of the WoT does not compromise its operation in resource-constrained environments. To this end, this dissertation studies a number of solutions to help guarantee the proper operation of constrained environments in an open and growing WoT.

1.2.2 Holistic security in an open Web of Things

Holistic and usable security mechanisms in resource-constrained environments is a second important challenge in an open WoT. Holistic approaches should target all facets of security such as: authentication, authorization, shielding sensitive information, protecting privacy, data confidentiality and other requirements. Some of the mechanisms required to meet these requirements might be difficult to implement on resource-constrained devices alone due to resource limitations. One example is a Building Management System (BMS) where many users, each with different privileges, are present in the building. In such a setting, enforcing individual access policies to the services offered by the building requires scalable authentication and fine-grained authorization which is challenging to implement solely on resource-constrained systems. Rather than inventing new security protocols, this dissertation uses the state-of-the-art to build a scalable and comprehensive security solution for use in resource-constrained environments.

1.2.3 Heterogeneity in the resource-constrained Internet of Things

The large scope of the IoT entails that it will comprise many different approaches and technologies: e.g. already today there exists a wide range of connectivity technologies, communication patterns, protocols, application modeling techniques and other technical elements for building IoT products. In an open IoT, this heterogeneity harms interoperability as resource-constrained systems are unable to be compatible with all of the different technologies available. Problems with interoperability between different IoT systems form a major hurdle to the vision of an interconnected and open IoT [45]. This dissertation proposes a method for abstracting a number of heterogeneous aspects of specific IoT technologies, with the aim of improving interoperability. Specifically, the focus lies on different interaction models, employing standardized vs proprietary protocols and dynamic network access.

1.2.4 Usability of constrained devices

A third set of challenges pertains to the limited usability of resource-constrained devices. As devices will measure and potentially impact our environment, users

expect easy methods for interacting with constrained devices. The current trend towards one mobile app per IoT product is problematic as the numbers of IoT products continues to rise. Instead, this dissertation studies how existing web technology can be applied to accommodate user interactions with constrained devices.

Secondly, integrating constrained devices into applications can sometimes be challenging due to missing features on these devices. This is usability from a software development and integration point of view. This dissertation considers ways of extending constrained devices with missing and/or new features in order to facilitate their integration into software. Coupled to this is the need for the functionality of constrained devices to evolve over time. As firmware updates can be costly or even not supported, other methods for deploying new functionality might become needed. This dissertation proposes such a method that outsources certain functions to unconstrained devices while maintaining the end-to-end aspect crucial for an open IoT.

1.2.5 Scalability of emerging LPWA technologies

As new and unproven Low Power Wide Area Network technologies are conquering the market, a critical evaluation of their claims is necessary in order to avoid unexpected deficiencies in future deployments. Additionally, existing open protocols and technologies are finding their way to these new connectivity technologies (cfr. Section 1.1.6). In order to verify such claims, the last research challenge of this work focuses on studying the scalability of LoRaWAN LPWA networks.

1.3 Outline

This dissertation is composed of a number of publications that were realized within the scope of this PhD. The selected publications provide an integral and consistent overview of the work performed. The different research contributions are detailed in Section 1.4 and the complete list of publications that resulted from this work is presented in Section 1.5. Compared to the original publications, minimal adjustments have been applied in order to correct linguistic issues, fix formatting issues or to further clarify the content. Within this section we give an overview of the remainder of this dissertation and explain how the different chapters are linked together.

Chapter 2 starts by introducing the need for distributed computing and communication in the IoT. We argue that the IoT requires a mix of local and remote processing in order to realize the wide range of requirements posed by different IoT applications. The result from deploying such distributed computing and communication is termed Distributed Intelligence (DI). Identified open challenges targeted by DI are timely processing, offline availability, scalability in terms of users and

devices, mobility of IoT systems and heterogeneity in protocols and data formats in resource-constrained systems. Another important challenge targeted by DI is protecting the privacy of users in the IoT, as local processing and storage may help protect sensitive information. The chapter continues by introducing Sensor Function Virtualization (SFV) as a method for realizing DI. SFV offloads certain functionality from constrained devices to unconstrained systems such as gateways, the cloud and other (in-network) computing infrastructure. SFV has similarities with Network Function Virtualization (NFV), which aims to virtualize the functions of many network equipment types in order to move these functions to commodity computing. By replacing network equipment with software, SFV enables cost savings for network operators. The main difference between SFV and NFV is their focus: NFV focuses mostly on networking functions while the focus of SFV is primarily on transport and application functions. A system architecture for SFV is presented where the desired functionality is split into modules. This modular approach facilitates at run-time management and deployment and distributed deployments over multiple systems. After illustrating the architecture for a number of deployment scenarios, the chapter is concluded by providing examples of DI in both the constrained and the unconstrained domain for CoRE.

After the introduction of DI, the concept of SFV in the unconstrained domain is elaborated further by proposing the Secure Service Proxy (SSP) in chapter 3. By combining ‘node virtualization’, where constrained devices are virtualized at the network layer, and a reverse proxy approach, the SSP is able to enhance constrained devices with a wide range of functionalities. Such functionalities are implemented as adapters, which are modular blocks of functionality that may be instantiated on virtual devices and that process RESTful requests and responses. Adapters for access control, caching, static resources, proxying and response rewriting have been implemented. Additionally, virtual devices can be extended with security primitives that overcome security-related issues common in low-power networks such as the weak authentication and limited scalability provided by pre-shared key and raw public key cipher suites. This approach is reminiscent of SSL brokers common in large web-facing infrastructures, albeit with different motivations. Two evaluations were performed to validate the effectiveness of the presented SSP in the context of CoRE. The results show that the SSP is able to provide scalable security to constrained devices while simultaneously reducing the resource usage of the devices. Secondly, the results also show that by extending virtual devices the SSP is able to provide new functionalities and to improve the performance of constrained devices and networks.

Chapter 4 studies how user interactions with constrained WoT devices can be improved. By combining the reverse proxy approach of the SSP, an HTTP/CoAP proxy, web templates and extensive response rewriting the presented system is able to generate user-friendly web User Interfaces (UIs). The main goal is to provide

IoT users on a mobile device with an intuitive UI for interacting with constrained devices embedded in their environment. Important considerations included minimal impact on constrained devices and mobile devices, minimal configuration and easy discovery and relatively easy to build UIs. Employing open web standards guarantees that the presented system is applicable across a wide range of constrained and mobile devices, while the ubiquity of web technology guarantees the ease of building UIs. The approach was demonstrated for web linking and for temperature sensing and lighting control. Additionally, the use of non-blocking template rendering led to a significant increase in UI responsiveness, which is important in constrained networks where network delays are often significant.

Chapter 5 tackles the challenge of integrating heterogeneous IoT technologies faced by service developers looking to add value on top of multiple IoT systems. This work stemmed from the observation that there exists a wide variety of connectivity options, protocols and communication models in track & trace IoT systems for logistics and transport. By abstracting the underlying technology specifics by means of a uniform, open standard-based RESTful interface hosted in the cloud, the integration of heterogeneous IoT technologies into services is improved without burdening (constrained) IoT devices nor service developers. A functional evaluation shows that the cloud platform is able to abstract two different communication models (push and pull). Additionally, a proof of concept dashboard was built on top of the cloud platform where device management, data access and control were implemented for logistics tracking, waste bin tracking and environmental monitoring systems.

To address the poor integration of LoRaWAN networks into the WoT, appendix A presents an extension of chapter 5 that represents LoRaWAN end devices as virtual CoAP servers, which host resources for publishing upstream messages and collecting downstream messages. By adding structure to the binary LoRaWAN payloads, one LoRaWAN message could be deserialized into multiple CoAP resources and vice versa. While this integration work has not been published, it did lead us to question the scalability of LoRaWAN, particularly in the presence of downstream traffic. Hence, the premise for the final chapter 6 was born.

Chapter 6 takes a step back from CoRE and studies the emerging low-power, long-range connectivity technology LoRaWAN. Specifically, claims regarding the scalability of large-scale LoRaWAN networks are assessed by modeling LoRaWAN in a network simulator. The results show that LoRaWAN networks are limited by interference as their size approaches thousands of devices. This may be somewhat mitigated by increasing the gateway density, which leads to a significant rise in the packet delivery ratio in saturated networks. The results also indicated that the downstream capacity in LoRaWAN networks is severely limited due to radio duty cycle restrictions on the gateways. Simulations revealed the detrimental effect of this limited downstream capacity on the packet delivery ratio of confirmed

upstream messages.

Figure 1.7 presents a graphical overview of the various chapters in this dissertation. Distributed intelligence spreads functionality over various non-constrained

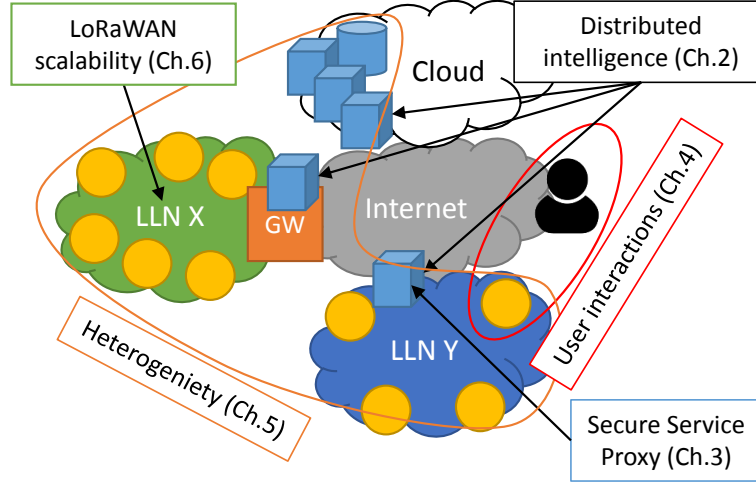


Figure 1.7: Situating the various chapters of this dissertation in a resource-constrained IoT network architecture

systems throughout the network. The SSP is one example of DI that employs device virtualization for realizing SFV. In the figure it is shown on the network edge of an Low-Power and Lossy Network (LLN) that deploys the CoRE RESTful protocol stack as sketched in the previous section. The topic on interactions between users and resource-constrained devices is circled in red. The fifth chapter studies the integration of heterogeneous devices and communication models in the resource-constrained IoT by means of the cloud. The final chapter focuses on the scalability of a specific type of LLN: LoRaWAN.

Finally, table 1.1 lists the research challenges from section 1.2 and indicates which challenges are targeted per chapter. Together with the table of contents, it may aid in browsing this dissertation.

Table 1.1: An overview of the targeted research challenges per chapter in this dissertation.

	Ch.2	Ch.3	Ch.4	Ch.5	Ch.6	App.A
An open Web of Things: - Efficient utilization of resources - Scalable and fine-grained security	•	• •				
Heterogeneity in the resource-constrained IoT: - Hide heterogeneity and improve interoperability		•		•		•
Usability of constrained devices: - Better user interactions - Extending feature set - Evolving over time		• •	•			
Emerging LPWA networks: - Standardized interactions - Scalability					•	•

1.4 Research contributions

In Section 1.2, the problems and challenges for an open Web of Things in Constrained RESTful Environments (CoRE) are formulated. They are tackled in the remainder of this dissertation. This PhD dissertation introduces the concepts of Distributed Intelligence and Sensor Function Virtualization as the two main methods for tackling the identified challenges. By applying and studying these concepts in the context of CoRE, the dissertation shows how DI and SFV enable an open WoT. Specifically, the following topics are studied: efficient resource utilization, extending constrained devices with new functionality, scalable authentication and authorization, user interactions with constrained devices and heterogeneous devices and communication models. Through thorough analysis, the dissertation quantifies the impact of the developed concepts for each of the aforementioned topics. Evaluation shows that DI improves resource utilization by caching responses, filtering traffic, combining resource requests and re-using security sessions. Furthermore, SFV is indeed able to extend constrained devices with new functionality in the form of new resources and user interfaces. This dissertation also applied sensor virtualization to develop a platform for integrating heterogeneous devices and communication models. Finally, the adoption of open web standards in LPWANs prompted this dissertation to study how the request-response pattern,

which is common in RESTful web services, is handled by LPWANs. The study shows the severe limitations of LPWANs for handling downstream traffic, which is problematic for a bi-directional protocol such as CoAP.

The following list presents the research contributions of this dissertation on a per chapter basis:

- Analysis of the need and enablers for distributed computing and communication in the constrained IoT (Ch. 2).
 - Identification of open challenges in distributed IoT systems that may benefit from distributed intelligence.
 - Introduction of SFV as an enabler for distributed intelligence.
 - Application of SFV in both the constrained and unconstrained domains of CoRE.
 - Illustration of how SFV may solve open challenges in distributed IoT systems.
- Design and implementation of a CoAP(s) proxy for a secure and smart Web of Things (Ch. 3).
 - Design of the adapter chain concept for deploying modular functionality on virtual devices.
 - Design of a RESTful interface for managing virtual devices and their adapter chains.
 - Implementation of the proxy using the CoAP++ framework, which was realized using Click Router [46], a C++ based modular framework that can be used to realize any network packet processing functionality.
 - Evaluation of the security termination method in the Cooja WSN network simulator.
 - Evaluation of CoAP observe aggregation on the w-iLab.t testbed ⁹.
- Methods for improving user interactions in the constrained WoT (Ch. 4).
 - Formulation of five requirements for improving user interactions with embedded web services in low-power networks.
 - Design of an application layer proxy that renders web interfaces to facilitate interaction with embedded web services on constrained devices.
 - Small-scale Wireless Sensor and Actuator Network (WSAN) evaluation of functionality and interface responsiveness.

⁹<http://doc.ilabt.iminds.be/ilabt-documentation/wilabfacility.html>

- Development of a cloud-based platform for integrating heterogeneous devices and communication models in the constrained IoT (Ch. 5).
 - Design and implementation of a cloud-based software architecture based on virtual device abstractions.
 - Evaluation of scalability and latency of virtual device abstraction approach.
 - Evaluation of integrating two different communication models via the platform.
 - Real world proof of concept demonstration of how the platform integrates heterogeneous IoT technologies in the logistics and transport sector.
- Critical assessment of the scalability of large-scale LoRaWAN LPWA networks (Ch. 6).
 - Construction of a complex baseband model of the LoRa physical layer.
 - Construction of a LoRa PHY error model from BER simulations for different spreading factors and coding rates.
 - Modeling LoRaWAN networks in the ns-3 discrete event network simulator:
 - * Integration of the LoRa PHY error model.
 - * Implementation of the LoRaWAN MAC layer for class A end devices.
 - * Implementation of a rudimentary LoRaWAN network server.
 - Evaluation of LoRaWAN scalability for various network sizes and traffic loads: confirmed versus unconfirmed messages, upstream versus downstream traffic and the impact of multiple gateways.

1.5 Publications

The research results obtained during this PhD research have been published in scientific journals and presented at a series of international conferences. The following list provides an overview of the publications during my PhD research.

1.5.1 Publications in international journals (listed in the ISI Web of Science ¹⁰)

1. Isam Ishaq, Jeroen Hoebeke, **Floris Van den Abeele**, Jen Rossey, Ingrid Moerman and Piet Demeester. *Flexible unicast-based group communication for CoAP-enabled devices*. Published in the special issue on ‘Wireless Sensor Networks and the Internet of Things’ in *Sensors*, Volume 14, Issue 6, p.9833–9877, 2014.
2. **Floris Van den Abeele**, Jeroen Hoebeke, Girum Ketema Teklemariam, Ingrid Moerman and Piet Demeester. *Sensor function virtualization to support distributed intelligence in the internet of things*. Published in *Wireless Personal Communications*, Volume 81, Issue 4, p.1415–1436, 2015.
3. Femke De Backere, Femke Ongenae, **Floris Van den Abeele**, Jelle Nelis, Pieter Bonte, E. Clement, M. Philpott, Jeroen Hoebeke, Stijn Verstichel, Ann Ackaert and Filip De Turck. *Towards a social and context-aware multi-sensor fall detection and risk assessment platform*. Published in *Computers in Biology and Medicine*, Volume 64, p.307-320, 2015.
4. **Floris Van den Abeele**, Jeroen Hoebeke, Ingrid Moerman and Piet Demeester. *Integration of heterogeneous devices and communication models via the cloud in the constrained internet of things*. Published in the special issue on ‘Leveraging the Internet of Things: Integration of Sensors and Cloud Computing Systems’ in the *International Journal of Distributed Sensor Networks*, Volume 11, Issue 10, 2015.
5. Femke De Backere, Femke Ongenae, Frederic Vannieuwenborg, Jan Van Ooteghem, Pieter Duysburgh, Arne Jansen, Jeroen Hoebeke, Kim Wuyts, Jen Rossey, **Floris Van den Abeele**, Karen Willems, Jasmien Decancq, Jan Henk Annema, Nicky Sulmon, Dimitri Van Landuyt, Stijn Verstichel, Pieter Crombez, Ann Ackaert, Dirk De Grooff, An Jacobs and Filip De Turck. *The OCareCloudS project: toward organizing care through trusted cloud*

¹⁰The publications listed are recognized as ‘A1 publications’, according to the following definition used by Ghent University: A1 publications are articles listed in the Science Citation Index Expanded, the Social Science Citation Index or the Arts and Humanities Citation Index of the ISI Web of Science, restricted to contributions listed as article, review, letter, note or proceedings paper.

services. Published in Informatics for Health & Social Care, Volume 41, Issue 2, p.159-176, 2016.

6. Girum Ketema Teklemariam, **Floris Van den Abeele**, Ingrid Moerman, Piet Demeester and Jeroen Hoebeke. *Bindings and RESTlets: a novel set of CoAP-based application enablers to build IoT applications*. Published in the special issue on 'Intelligent Internet of Things (IoT) Networks' in Sensors, Volume 16, issue 8, 2016.
7. Jetmir Haxhibeqiri, **Floris Van den Abeele**, Ingrid Moerman, Jeroen Hoebeke. *LoRa Scalability: a Simulation Model Based on Interference Measurements*. Published in Sensors, Volume 17, Issue 6, 2017.
8. **Floris Van den Abeele**, Ingrid Moerman, Piet Demeester and Jeroen Hoebeke. *Secure Service Proxy: A CoAP(s) Intermediary for a Securer and Smarter Web of Things*. Published in Sensors, Volume 17, Issue 7, 2017.
9. **Floris Van den Abeele**, Jetmir Haxhibeqiri, Ingrid Moerman and Jeroen Hoebeke. *Scalability analysis of large-scale LoRaWAN networks in ns-3*. Submitted to the IEEE Internet of Things Journal, May 2017.

1.5.2 Publications in other international journals

1. Isam Ishaq, David Carels, Girum Ketema Teklemariam, Jeroen Hoebeke, **Floris Van den Abeele**, Eli De Poorter, Ingrid Moerman and Piet Demeester. *IETF standardization in the field of the Internet of Things (IoT): a survey*. Published in the Journal of Sensor and Actuator Networks, Volume 2, issue 2, pp. 235287, 2013

1.5.3 Publications in international conferences (listed in the ISI Web of Science ¹¹)

1. Isam Ishaq, Jeroen Hoebeke, **Floris Van den Abeele**, Ingrid Moerman and Piet Demeester. *Group communication in constrained environments using CoAP-based entities*. Published in the proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2013), p. 345–350, 21–23 May 2013, Cambridge, Massachusetts, USA.
2. **Floris Van den Abeele**, Jeroen Hoebeke, Ingrid Moerman and Piet Demeester. *Fine-grained management of CoAP interactions with constrained*

¹¹The publications listed are recognized as 'P1 publications', according to the following definition used by Ghent University: P1 publications are proceedings listed in the Conference Proceedings Citation Index - Science or Conference Proceedings Citation Index - Social Science and Humanities of the ISI Web of Science, restricted to contributions listed as article, review, letter, note or proceedings paper, except for publications that are classified as A1.

IoT devices Published in the proceedings of the IEEE Network Operations and Management Symposium (NOMS 2014), 5–9 May 2014, Krakow, Poland.

3. Girum Ketema Teklemariam, Jeroen Hoebeke, **Floris Van den Abeele**, Ingrid Moerman and Piet Demeester. *Simple RESTful sensor application development model using CoAP*. Published in the proceedings of the 39th IEEE Conference on Local Computer Networks (LCN Workshops 2014), p.552–556, 8–11 Sep. 2014, Edmonton, Canada.
4. **Floris Van den Abeele**, Tom Vandewinckele, Jeroen Hoebeke, Ingrid Moerman and Piet Demeester. *Secure communication in IP-based wireless sensor network via a trusted gateway*. Published in the proceedings of the 10th IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2015), p.1–6, 7–9 Apr. 2015, Singapore.
5. Enri Dalipi, **Floris Van den Abeele**, Isam Ishaq, Ingrid Moerman and Jeroen Hoebeke. *EC-IoT: an easy configuration framework for constrained IoT devices*. Published in the proceedings of the IEEE World Forum on the Internet of Things (WF-IoT 2016), 12–14 Dec. 2016, Reston, Virginia, USA.
6. **Floris Van den Abeele**, Enri Dalipi, Ingrid Moerman, Piet Demeester and Jeroen Hoebeke. *Improving user interactions with constrained devices in the Web of Things*. Published in the proceedings of the IEEE World Forum on the Internet of Things (WF-IoT 2016), 12–14 Dec. 2016, Reston, Virginia, USA.
7. Jetmir Haxhibeqiri, Abdulkadir Karagaac, **Floris Van den Abeele**, Wout Joseph, Ingrid Moerman and Jeroen Hoebeke. *LoRa Indoor Coverage and Performance in an Industrial Environment: Case Study*. Accepted at the 2017 IEEE 22st International Conference on Emerging Technologies and Factory Automation (ETFA 2017), 12–15 Sep. 2017, Limassol, Cyprus.

1.5.4 Publications in other international conferences

1. **Floris Van den Abeele**, Jeroen Hoebeke, Isam Ishaq, Girum Ketema Teklemariam, Jen Rossey, Ingrid Moerman and Piet Demeester. *Demo Abstract: Building embedded applications via REST services for the Internet of Things*. Published in the proceedings of the 11th ACM Conference on Embedded Network Sensor Systems (SenSys - 2013), p. 1–2, 11–15 Nov. 2013, Rome, Italy.

2. Femke De Backere, Femke Ongenae, **Floris Van den Abeele**, Jeroen Hoebeke, Stijn Verstichel, Ann Ackaert and Filip De Turck. *Social-aware and Context-aware Multi-sensor Fall Detection Platform*. Published in the proceedings of the Working on Semantic Web Applications and Tools for Life Sciences (SWAT4LS 2013), p. 1–4, 9–12 Dec. 2013, Edinburgh, UK.
3. **Floris Van den Abeele**, Jeroen Hoebeke, Femke De Backere, Femke Ongenae, Pieter Bonte, Stijn Verstichel, T. Carlier, Pieter Crombez, K. De Gryse, S. Danschotter, Ingrid Moerman and Filip De Turck. *OCareClouds: Improving Home Care by Interconnecting Elderly, Care Networks and Their Living Environments*. Published in the proceedings of the 8th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth 2014), p. 1–2, 20–23 May 2014, Oldenburg, Germany.

1.5.5 Contributions to standardization bodies

1. Bert Greevenbosch, Jeroen Hoebeke, Isam Ishaq, **Floris Van den Abeele**. *CoAP Profile Description Format*. Published as IETF CoRE Internet draft draft-greevenbosch-core-profile-description-02, 21 June 2013.
2. Isam Ishaq, Jeroen Hoebeke, **Floris Van den Abeele**. *CoAP Entities*. Published as IETF CoRE Internet draft draft-ishaq-core-entities-00, 17 June 2013.
3. Shitao Li, Kepeng Li, Jeroen Hoebeke, **Floris Van den Abeele**, Antonio J. Jara. *Conditional observe in CoAP*. Published as IETF CoRE Internet draft draft-li-core-conditional-observe-05, 15 October 2014.

1.5.6 Patent applications

1. Jeroen Hoebeke, **Floris Van den Abeele**. *Emulating Functionality for Constrained Devices*. US2015365467. Koninklijke KPN N.V., iMinds VZW, Universiteit Gent. Priority date: 28 December 2012. Publication date: 17 December 2015.
2. **Floris Van den Abeele**, Jeroen Hoebeke, Girum Ketema Teklemariam. *Reducing a Number of Server-Client Sessions*. US2016006818. Koninklijke KPN N.V., iMinds VZW, Universiteit Gent. Priority date: 28 December 2012. Publication date: 7 January 2016.
3. Jeroen Hoebeke, Girum Ketema Teklemariam, **Floris Van den Abeele**. *Binding Smart Objects*. US2017017533. Koninklijke KPN N.V., iMinds VZW, Universiteit Gent. Priority date: 23 December 2013. Publication date: 19 January 2017.

4. **Floris Van den Abeele**, Jeroen Hoebeke. *Crash recovery for smart objects*. US2017005875. Koninklijke KPN N.V., iMinds VZW, Universiteit Gent. Priority date: 23 January 2014. Publication date: 5 January 2017.

References

- [1] Gartner Inc. *Forecast: The Internet of Things, Worldwide*. Technical report, Gartner, 2013. Available from: <https://www.gartner.com/newsroom/id/2636073>.
- [2] Gartner Inc. *Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015*. Technical report, Gartner, 2015. Available from: <https://www.gartner.com/newsroom/id/3165317>.
- [3] Morgan Stanley Research Global. *Blue Paper: Internet of Things - Wearable Devices*. Technical report, Morgan Stanley, 2014. Available from: http://byinnovation.eu/wp-content/uploads/2014/11/MORGAN-STANLEY-BLUE-PAPER_{_}Internet-of-Things.pdf.
- [4] C. Bormann, M. Ersue, and A. Keranen. *RFC 7228: Terminology for Constrained-Node Networks*. Technical report, IETF, 2014. Available from: <http://tools.ietf.org/html/rfc7228>.
- [5] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. *A comparison of mechanisms for improving TCP performance over wireless links*. IEEE/ACM Transactions on Networking, 5(6):756–769, 1997. Available from: <http://ieeexplore.ieee.org/document/650137/>, doi:10.1109/90.650137.
- [6] G. Holland and N. Vaidya. *Analysis of TCP Performance over Mobile Ad Hoc Networks*. Wireless Networks, 8(2/3):275–288, 2002. Available from: <http://link.springer.com/10.1023/A:1013798127590>, doi:10.1023/A:1013798127590.
- [7] R. Chakravorty, J. Cartwright, and I. Pratt. *Practical experience with TCP over GPRS*. In IEEE Global Telecommunications Conference (GLOBECOM 2002), volume 2, pages 1678–1682. IEEE, 2002. Available from: <http://ieeexplore.ieee.org/document/1188483/>, doi:10.1109/GLOCOM.2002.1188483.
- [8] J. W. Hui and D. E. Culler. *IP is dead, long live IP for wireless sensor networks*. Proceedings of the 6th ACM conference on Embedded network sensor systems - SenSys '08, page 15, 2008. Available from: <http://portal.acm.org/citation.cfm?doid=1460412.1460415>, doi:10.1145/1460412.1460415.
- [9] A. Dunkels, J. Alonso, and T. Voigt. *Making TCP/IP viable for wireless sensor networks*, 2003.
- [10] A. Dunkels, J. Alonso, T. Voigt, H. Ritter, and J. Schiller. *Connecting Wireless Sensornets with TCP/IP Networks*. In Lecture Notes in

- Computer Science, pages 143–152. Springer, Berlin, Heidelberg, 2004. Available from: http://link.springer.com/10.1007/978-3-540-24643-5_{-}13, doi:10.1007/978-3-540-24643-5-13.
- [11] A. Dunkels. *Contiki: Bringing IP to Sensor Networks*. ERCIM News, 2009(76), 2009. Available from: <http://ercim-news.ercim.eu/en76/rd/contiki-bringing-ip-to-sensor-networks>.
- [12] D. Yazar and A. Dunkels. *Efficient application integration in IP-based sensor networks*. In Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings - BuildSys '09, page 43, New York, New York, USA, 2009. ACM Press. Available from: <http://portal.acm.org/citation.cfm?doid=1810279.1810289>, doi:10.1145/1810279.1810289.
- [13] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. *Tiny web services*. In Proceedings of the 6th ACM conference on Embedded network sensor systems - SenSys '08, page 253, New York, New York, USA, 2008. ACM Press. Available from: <http://portal.acm.org/citation.cfm?doid=1460412.1460438>, doi:10.1145/1460412.1460438.
- [14] R. T. Fielding and R. N. Taylor. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.
- [15] D. Guinard. *Towards the web of things: Web mashups for embedded devices*. In In MEM 2009 in Proceedings of WWW 2009. ACM, 2009.
- [16] D. Guinard, V. Trifa, and E. Wilde. *A resource oriented architecture for the Web of Things*. In 2010 Internet of Things (IOT), pages 1–8. IEEE, nov 2010. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5678452>, doi:10.1109/IOT.2010.5678452.
- [17] E. Wilde. *Putting things to REST*. School of Information, 2007.
- [18] S. Mayer, D. Guinard, E. Wilde, and M. Kovatsch. *WoT 2016*. In Proceedings of the Seventh International Workshop on the Web of Things - WoT '16, pages 1–4, New York, New York, USA, 2016. ACM Press. Available from: <http://dl.acm.org/citation.cfm?doid=3017995.3017996>, doi:10.1145/3017995.3017996.
- [19] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. *RFC 4944: Transmission of IPv6 packets over IEEE 802.15. 4 networks*, 2007. Available from: <https://tools.ietf.org/html/rfc4944>.

- [20] J. Hui and P. Thubert. *RFC 6282: Compression format for IPv6 datagrams over IEEE 802.15. 4-based networks*, 2011. Available from: <https://tools.ietf.org/html/rfc6282.txt>.
- [21] G. Mulligan. *The 6LoWPAN architecture*. In Proceedings of the 4th workshop on Embedded networked sensors - EmNets '07, page 78, New York, New York, USA, 2007. ACM Press. Available from: <http://portal.acm.org/citation.cfm?doid=1278972.1278992>, doi:10.1145/1278972.1278992.
- [22] Z. Shelby, S. Chakrabarti, E. Nordman, and C. Bormann. *RFC 6775: Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*. Technical report, IETF, 2012. Available from: <https://tools.ietf.org/html/rfc6775>.
- [23] I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. Van den Abeele, E. De Poorter, I. Moerman, and P. Demeester. *IETF standardization in the field of the Internet of Things (IoT): a survey*. Journal of Sensor and Actuator Networks, 2(2):235–287, 2013.
- [24] M. Durvy, N. Finne, A. Dunkels, J. Abeillé, P. Wetterwald, C. O’Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, and N. Tsiftes. *Making sensor networks IPv6 ready*. In Proceedings of the 6th ACM conference on Embedded network sensor systems - SenSys '08, page 421, New York, New York, USA, 2008. ACM Press. Available from: <http://portal.acm.org/citation.cfm?doid=1460412.1460483>, doi:10.1145/1460412.1460483.
- [25] A. Dunkels. *Sicslowpan-internet-connectivity for low-power radio systems*. SICS, 2008. Available from: <https://www.iis.se/docs/SICS{-}Lowpan-report.pdf>.
- [26] J. W. Hui. *An Extended Internet Architecture for Low-Power Wireless Networks - Design and Implementation*. PhD thesis, EECS Department, University of California, Berkeley, sep 2008. Available from: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-116.html>.
- [27] C. Bormann, A. P. Castellani, and Z. Shelby. *CoAP: An Application Protocol for Billions of Tiny Internet Nodes*. IEEE Internet Computing, 16(2):62–67, mar 2012. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6159216>, doi:10.1109/MIC.2012.29.
- [28] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. *RFC 7252: Constrained Application Protocol (CoAP)*, 2014. Available from: <https://tools.ietf.org/html/rfc7252>.

- [29] M. Kovatsch. *Scalable Web Technology for the Internet of Things*. PhD thesis, ETH-Zürich, 2015. Available from: <http://www.vs.inf.ethz.ch/publ/papers/mkovatsc-2015-dissertation.pdf>.
- [30] Z. Shelby. *RFC 6690: Constrained RESTful Environments (CoRE) Link Format*, 2012. Available from: <https://tools.ietf.org/html/rfc6690>.
- [31] K. Hartke. *RFC 7641: Observing Resources in the Constrained Application Protocol (CoAP)*. Technical report, IETF, 2015. Available from: <https://tools.ietf.org/html/rfc7641>.
- [32] C. Bormann and Z. Shelby. *RFC 7959: Block-Wise Transfers in the Constrained Application Protocol*. Technical report, IETF, 2016. Available from: <https://tools.ietf.org/html/rfc7959>.
- [33] A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk. *RFC 8075: Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)*. Technical report, IETF, 2017. Available from: <https://tools.ietf.org/html/rfc8075>.
- [34] A. Rahman and E. Dijk. *RFC 7390: Group Communication for the Constrained Application Protocol (CoAP)*. Technical report, IETF, 2014. Available from: <https://tools.ietf.org/html/rfc7390>.
- [35] P. van der Stok, C. Bormann, and A. Sehgal. *RFC 8132: PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)*. Technical report, IETF, 2017. Available from: <https://tools.ietf.org/html/rfc8132>.
- [36] Z. Shelby, M. Koster, C. Bormann, and P. van der Stok. *CoRE Resource Directory*. Technical report, IETF, 2017. Available from: <https://tools.ietf.org/html/draft-ietf-core-resource-directory-10>.
- [37] Z. Shelby, M. Vial, M. Koster, and C. Groves. *Reusable Interface Definitions for Constrained RESTful Environments*. Technical report, IETF, 2017. Available from: <https://tools.ietf.org/html/draft-ietf-core-interfaces-09>.
- [38] C. Jennings, Z. Shelby, J. Arkko, A. Keranen, and C. Bormann. *Media Types for Sensor Measurement Lists (SenML)*. Technical report, IETF, 2017. Available from: <https://tools.ietf.org/html/draft-ietf-core-senml-07>.
- [39] C. Bormann and P. Hoffman. *RFC 7049: Concise Binary Object Representation (CBOR)*. Technical report, IETF, 2014. Available from: <https://tools.ietf.org/html/rfc7049>.
- [40] K. Kuladinithi, O. Bergmann, and M. Becker. *Implementation of CoAP and its Application in Transport Logistics*. In Proc. IP+ SN, Chicago, IL, USA, 2011.

- [41] A. Ludovici, P. Moreno, and A. Calveras. *TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS*. Journal of Sensor and Actuator Networks, 2(2):288–315, may 2013. Available from: <http://www.mdpi.com/2224-2708/2/2/288/>, doi:10.3390/jsan2020288.
- [42] M. Kovatsch, M. Lanter, and Z. Shelby. *Californium: Scalable cloud services for the Internet of Things with CoAP*. In 2014 International Conference on the Internet of Things (IOT), pages 1–6. IEEE, oct 2014. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7030106>, doi:10.1109/IOT.2014.7030106.
- [43] A. Minaburo, L. Toutain, and C. Gomez. *LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP*. Technical report, IETF, 2017. Available from: <https://tools.ietf.org/html/draft-ietf-lpwan-ipv6-static-context-hc-03>.
- [44] A. Minaburo and L. Toutain. *LPWAN Static Context Header Compression (SCHC) for CoAP*. Technical report, IETF, 2017. Available from: <https://tools.ietf.org/html/draft-ietf-lpwan-coap-static-context-hc-01>.
- [45] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. *Internet of things: Vision, applications and research challenges*. Ad Hoc Networks, 10(7):1497–1516, 2012.
- [46] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. *The Click Modular Router*. ACM Trans. Comput. Syst., 18(3):263–297, aug 2000. doi:10.1145/354871.354874.

2

Sensor function virtualization to support distributed intelligence in the Internet of Things

This chapter argues that a combination of local and remote processing and communication is necessary to meet the broad requirements of diverse Internet of Things (IoT) applications. To this end, the concepts Distributed Intelligence (DI) and Sensor Function Virtualization (SFV) are introduced in this chapter. This chapter then continues to illustrate SFV in the context of Constrained RESTful Environments (CoRE) for both the constrained and unconstrained domain. This chapter lies the conceptional foundation for the work in chapter 3, where SFV in the unconstrained domain is studied further.

Floris Van den Abeele, Jeroen Hoebeke, Girum Ketema Teklemariam, Ingrid Moerman and Piet Demeester

Published in Wireless Personal Communications Volume 81 Issue 4, April 2015.

Abstract It is estimated that - by 2020 - 50 billion devices will be connected to the Internet. This number not only includes TVs, PCs, tablets and smartphones, but

also billions of embedded sensors that will make up the “Internet of Things” and enable a whole new range of intelligent services in domains such as manufacturing, health, smart homes, logistics, etc. To some extent, intelligence such as data processing or access control can be placed on the devices themselves. Alternatively, functionalities can be outsourced to the cloud. In reality, there is no single solution that fits all needs. Cooperation between devices, intermediate infrastructures (local networks, access networks, global networks) and/or cloud systems is needed in order to optimally support IoT communication and IoT applications. Through distributed intelligence the right communication and processing functionality will be available at the right place. The first part of this paper motivates the need for such distributed intelligence based on shortcomings in typical IoT systems. The second part focuses on the concept of Sensor Function Virtualization, a potential enabler for distributed intelligence, and presents solutions on how to realize it.

2.1 Introduction

Today, most of the data available on the Internet is generated by humans. With more and more everyday objects or sensors being connected to the Internet, the amount of data generated by things is going to increase rapidly. It is estimated that by 2020, 50 billion devices will be connected to the Internet, outnumbering the number of human beings on our planet [1]. This vision is commonly referred to as the Internet of Things.

In the Internet of Things, heterogeneous objects will grasp information about our physical world and inject it in the virtual world where it can be used as input to all kinds of services. Depending on the type of service, a decision can be taken to act again upon the physical world. As a result, everything around us will become an integral part of the Internet, capable of generating and consuming information. It is evident that the Internet of Things may have a great impact in a wide range of application domains such as health, manufacturing, building automation, transportation, smart cities, logistics, etc.

In order to connect things to the Internet, they need to be equipped with processing and communication capabilities. In many cases these capabilities are very limited, as these devices may need to run on batteries for several months or years or need to be produced at an extremely low cost. Such devices are often referred to as constrained devices, many of them belonging to Class 1 (approximately 10KiB RAM and 100KiB ROM) as defined by the IETF Iwlg working group [2]. Specific low data rate and low power communication technologies have been designed, such as IEEE 802.15.4. Further, as typical Internet protocols had not been designed for such small footprints, initial efforts to interconnect these devices to the Internet resulted in proprietary protocols and architectures. However, their incompatibility with widely adopted Internet protocols has hampered their uptake and

the realization of the IoT vision.

The last few years, this mindset has been changing and many efforts have been put into the extension of Internet technologies to constrained devices. Most noteworthy are the efforts of the Internet Engineering Task Force, the de facto standardization organization for Internet protocols. Their initial efforts focused on the networking layer and resulted in the integration of constrained devices and constrained networks in the IPv6 Internet. As a next step, they now target the efficient integration of these devices in web services. So far, the IETF has standardized the Constrained Application Protocol (CoAP), which can be seen as an embedded counterpart of HTTP. With these technologies, it has become possible to interconnect tiny objects or networks of such objects with the IPv6 Internet and to build applications that interact with them using embedded web service technology, bringing us one step closer to the realization of the Internet of Things.

From the above description, it becomes clear that it has been a challenge to fit an Internet-compatible protocol stack on devices with very limited capabilities. It required the careful design of tailored communication protocols. One can question how far one can go to further extend these devices with additional functionalities or intelligence typically encountered in more powerful devices, such as access control, preprocessing of data, data formatting, resource visibility, etc. At some point it will become technically infeasible to put additional intelligence on the devices themselves due to their constraints. Consequently, such intelligence needs to be outsourced to more powerful devices such as gateways, routers, neighboring devices, the cloud, etc. In addition, as every IoT application may have different requirements and devices may be very heterogeneous, there will be no single recipe on how to optimally distribute this intelligence.

The optimal placement of functionalities or intelligence is only one aspect of the whole picture. Current networks and radio technologies are very homogeneous, homogeneous in a sense that the resulting communication infrastructure often offers the very same service to every application running on top of it. Looking at the large number of IoT application domains or the variety of IoT applications with heterogeneous requirements within a single application domain, optimal support of IoT applications also requires adaptations of the network behavior. Whenever possible, networking elements should expose the necessary interfaces in order to optimally configure the underlying network behavior.

Combining both aspects brings us to the concept of distributed intelligence¹ in the context of the Internet of Things: depending on the application, user and policy requirements, it is decided where to place the intelligence to operate certain functions and how to optimally configure the communication infrastructure.

In the remainder of the paper, we will first further motivate the need for such distributed intelligence by identifying a number of shortcomings in typical IoT

¹Not to be confused with Distributed Intelligence in the context of AI.

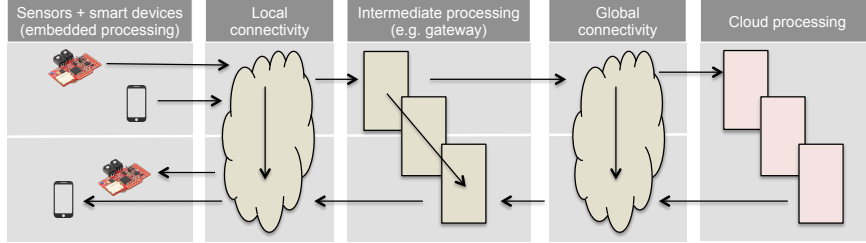


Figure 2.1: A generic Internet of Things system

systems in section 2.2. Section 2.3 presents Sensor Function Virtualization (SFV) as an important enabler for realizing distributed intelligence. Before moving on to examples of SFV, section 2.4 gives a short overview of the IETF IoT protocol stack. This stack is used for examples of SFV in the unconstrained and constrained domain in sections 2.5 and 2.6 respectively. Section 2.7 presents the related work that we have identified in the literature. The paper ends with a number of conclusions and future work in section 2.8.

2.2 The need for distributed intelligence

2.2.1 Generic IoT system

Figure 2.1 shows a high-level representation of a generic IoT system from a communication and processing point of view. On the left there are the embedded devices with some form of processing and communication capabilities. These devices can be very heterogeneous in terms of energy provisioning (energy harvesting, battery powered, mains powered), communication capabilities (IEEE 802.15.4, BLE, IEEE 802.11, etc.), processing power and communication behavior (always connected versus intermittently connected). Examples of such devices are a battery-operated environmental sensor, a Wi-Fi weighing scale, a tracking device with a GPRS module or even a powerful smart phone. In an IoT system, these devices typically collect information about the physical world and, possibly after some (limited) local processing, communicate this information to an external service or another device for further processing.

The information is then transmitted over a local communication network, such as a 6LoWPAN network or a WLAN network, in order to reach the Internet. On its way out, it passes intermediate processing components such as a home gateway, a border router or an access point. In most cases, such a component will mainly perform some protocol translations (e.g. conversion between 6LoWPAN and IPv6, NAT translation, etc.). After that, the information is communicated over a global communication network (i.e. the Internet) until it reaches a cloud service. The

cloud service will process the sensed data, enrich it, combine it with other sources of information and eventually convert it into retrievable knowledge that needs to be stored or actions that need to be performed in the real world. In the latter case, there will be a flow from right to left in figure 2.1, until the action reaches an embedded device such as an actuator.

Of course, many variants of the above generic IoT system exist and in many IoT systems not all of the components shown in figure 2.1 need to be present or involved. Therefore, in figure 2.2, we have mapped a number of realistic IoT use cases to this generic system.

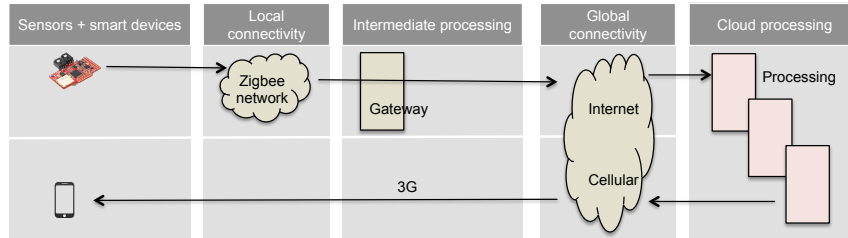
2.2.2 Open challenges

In this subsection we will illustrate through a number of concrete examples, i.e. instantiations of the generic IoT system shown in figure 1, some of the existing limitations of state-of-the-art IoT systems and pinpoint the root cause of these limitations.

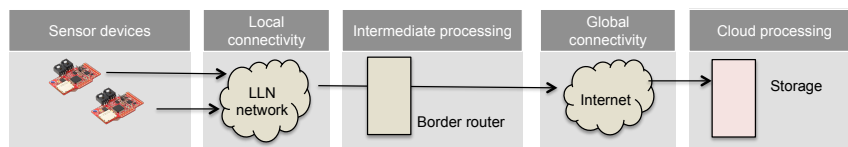
2.2.2.1 Act in time

Many IoT business models adopt a cloud-based approach as partially illustrated in figure 2.2a. Sensor data is pushed to the cloud where it is being processed. If needed, corrective actions are taken. All intelligence is centralized in the cloud, which makes it convenient for the cloud service provider to maintain the system and to roll out new functionalities. However, depending on the application, the complete sensing and actuation cycle might have to be completed within a certain time interval. For instance, turning on a light using a light switch requires a worst case latency of 200ms or better, whereas other home automation applications may live with higher latencies. Consequently, depending on the type of IoT application, the processing functionality may reside either far away from the constrained devices (in the cloud) or must reside nearby in order to meet certain performance requirements. Future IoT systems should be able to cope with these requirements, e.g. by supporting distributed processing that puts the intelligence wherever it is most needed.

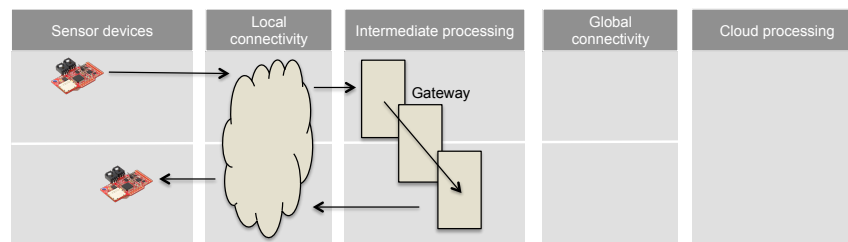
Even when the processing functionality resides close to the sensors and actuators, e.g. on a gateway as shown in figure 2.2c, it may still be possible that the underlying network is not capable of meeting the performance requirements imposed by the IoT application e.g. due to suboptimal routing, inefficient medium access control or competition with other wireless traffic. To tackle this, it should not only become possible to place the intelligence where it is needed, but also to optimally configure the underlying network. Similar observations can be made for cases where for instance synchronization is needed.



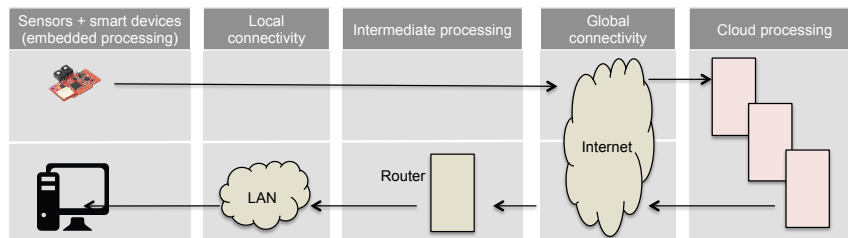
a) Zigbee sensor generates an alarm upon which a user needs to be informed. The intelligence resides in the cloud.



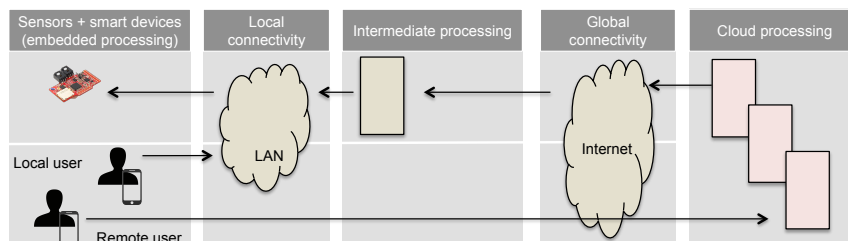
b) Sensors in a 6LoWPAN sensor network monitor their environment. All measurements are sent to the cloud where they are stored.



c) Temperature sensors in a home automation network measure temperature. Based on the measurements, the HVAC system is being triggered. All intelligence resides in the gateway.



d) A tracking device with a SIM card communicates its position to a cloud service over a GPRS connection. The information is stored in the cloud and can be consulted by customers.



e) User 1 (residing in the same network) and user 2 (residing in a remote network) are interacting with a sensor device.

Figure 2.2: Five IoT scenarios mapped to the generic IoT system from figure 2.1

2.2.2.2 Work offline

IoT applications that solely rely on intelligence in a cloud system will completely break upon an interruption of the Internet connectivity (the “global connectivity” in figures 2.1 and 2.2). For instance, one can create a building management system that is fully managed and controlled by the cloud based on locally collected sensor data. However, in case the cloud is unavailable, the building management system should still be able to deliver a minimal service level. Consequently, the core functionalities of an IoT application should reside in the local network. This way, using simplified local processing, it is still possible to have an operational application albeit with reduced functionality. If connectivity is available, more advanced functionalities, e.g. by using externally available data, can be offered to the users. Again, distribution of intelligence and processing is needed to offer a robust IoT system.

2.2.2.3 Serve many

IoT systems such as the one shown in figure 2.2e, may not only scale up to thousands of sensors and actuators, but may also involve a multitude of users, each user taking up a particular role in the overall system. For instance, in a building management system one can have the building owner, facility managers, tenants, cleaning staff, visitors, etc. Depending on their role and the corresponding policies, users should perceive a different system in terms of the data they can see, the actions they can take, etc. However, many of the involved devices are not capable of supporting this level of granularity in terms of visibility, access control, etc. due to their constraints. Even if they would be able to offer some of this functionality such as a security algorithm, it most likely would not scale with the number of users due to resource depletion, i.e. every additional user will require storage of additional state information. Therefore, it should become possible to outsource this functionality to more powerful infrastructure, preferably without impacting the constrained devices. Further, the outsourced functionality ought to be placed at the most optimal location. Note that this depends on the actual location of the user, e.g. a local versus a remote user.

2.2.2.4 Move and sleep

The IoT system shown in figure 2.2d involves tracking devices. These devices are mobile as they move with the object they are tracking. In most cases, as these devices are also battery powered, they are not permanently connected to the Internet. This has two consequences. First of all, the IP address of the device will change over time. Secondly, when users want to interact with the device, e.g. to perform some reconfigurations, the device will mostly be offline. Both aspects complicate the interaction with these devices for end users or external systems. Additional

functionality has to be provided along the communication path in order to hide the sleepy and mobile behavior of the devices, thereby offering a uniform view to the outside world.

2.2.2.5 Monoglot

Most constrained devices do not possess the capabilities for supporting a wide range of data formats or protocols. In many cases, they only speak a single protocol and deliver their data in a particular format or content type. Further, the transferred data is often kept as compact as possible in order to reduce the communication overhead and to save energy. Lengthy, verbose descriptions are out of the question. When an IoT application is implemented as a vertical silo, where devices and cloud are designed to be interoperable, this may not be a problem. However, in more open systems where a variety of devices and services may interact with each other or where IoT devices can be connected to any service provider, this may easily lead to interoperability problems.

Different parties will most likely support different data formats (e.g. JSON versus XML, Fahrenheit versus Centigrade, etc.). When they need to interact, additional intelligence is needed for translating between incompatible data formats, else lack of interoperability will prevent their collaboration. Sometimes, the data generated by constrained devices is not self-descriptive. Therefore, on its path towards e.g. a cloud service, the data may be enriched with additional information to make it completely self-describing. A last example relates to the semantic web. Before semantic reasoning can take place, the sensor data needs to be tagged with additional semantic information. Again, placing such semantic descriptions on the constrained devices themselves might be too complex, so this functionality is better outsourced to more powerful devices. These examples illustrate that by dynamically placing intelligence in network or processing elements, interoperability can be greatly increased.

2.2.3 Distributed intelligence

From the previous examples it becomes clear that in order to optimally support a wide variety of IoT applications and user needs, additional intelligence is needed. This intelligence is not only related to the processing of data, but is also related to security, Quality of Service, network configuration, etc. Further, there is no single place where this intelligence has to be placed or activated. Depending on the situation, it may be spread from the devices themselves up to the cloud, covering all components in the chain shown in figure 2.1. In many cases, the intelligence needs to be distributed over different locations in order to deliver the desired functionality or performance. Further, it involves both processing and networking elements.

This is what we define as distributed intelligence. It is the cooperation between devices, intermediate communication infrastructures (local networks, access networks, global networks) and/or cloud systems in order to optimally support IoT communication and IoT applications. Starting from the application requirements, user needs and policies, intelligence is optimally distributed and activated in order to operate functions such as processing, security, QoS and to configure the communication infrastructure. Through distributed intelligence, the right communication and processing functionality will be available at the right place and at the right time as is illustrated in figure 2.3.

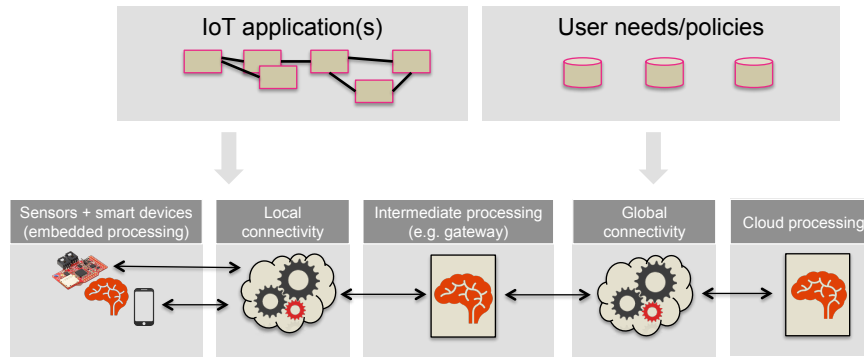


Figure 2.3: The concept of distributed intelligence

Distributed intelligence will enable us to tackle the aforementioned shortcomings of existing IoT systems. It can contribute to increased interoperability, better usage of scarce resources, better application performance, improved user experience, more secure systems, etc. As can be seen from the previous discussion its realization is not straightforward and imposes significant challenges, mainly due to the fact that it is distributed in nature, involves both processing and communication and, communication-wise, pertains to multiple layers in the communication stack.

In the remainder of the paper, we will propose some concrete enablers to support the concept of distributed intelligence and will illustrate how they can work in the context of open IETF-based IoT protocol stacks. Hereby, we focus on the fact that typical IoT devices are constrained in nature and not capable of offering all functionality needed. Therefore, solutions are needed to outsource such functionalities to more powerful components, i.e. to virtualize this functionality as will be explained in the following section. Other interesting aspects of distributed intelligence, such as enabling an optimal configuration of the underlying network are outside the scope of this work.

2.3 Sensor Function Virtualization for the Internet of Things

The previous section illustrated why distributed processing is needed in the Internet of Things and gave a high-level overview of distributed intelligence. This section looks at how distributed processing can be realized while keeping in mind the challenges specific to the IoT domain. To this end, we propose an approach that enables distributed processing by offloading certain functionality from constrained devices to unconstrained infrastructure such as a (virtualized) gateway, the cloud and other (in-network) infrastructure. Hence the term “Sensor Function Virtualization” (SFV), where a sensor is defined more broadly to also include actuators and other types of constrained devices.

As the Internet of Things is expected to include up to 50 billion devices by 2020, any proposed solution for such sensor function virtualization should be able to scale as the number of devices increases. Here, two important points are to be noted. By running (parts of the) sensor function virtualization on cloud infrastructure, we expect our solution to profit from the elasticity provided by these environments: i.e. as the number of devices increases, more resources are automatically allocated by the cloud infrastructure to handle the increased load. Elasticity is a huge benefit for scalability that results from pooling resources in large data centers. Apart from elasticity, sensor function virtualization solutions should also follow a tiered design. Here, multiple tiers work together by each taking care of a part of the sensor function virtualization. Next to a more scalable approach, a tiered design also allows to mitigate some of the issues present in sensor function virtualization that relies solely on cloud-based infrastructure (e.g. high latency and non-functioning devices when Internet is unavailable).

Another important aspect for sensor function virtualization is the heterogeneity of both constrained devices and infrastructure. Constrained devices might be battery powered, might be part of a fixed communication infrastructure, might be mobile, have a large diversity in processing power and so on. Also, different forms of infrastructure that can aid in sensor function virtualization are expected to exist. In some deployments (e.g. typical WSN scenarios) there might be a mains-powered gateway that can assist in sensor function virtualization. In other settings, such gateways are not common but the access network itself might assist in sensor function virtualization (e.g. mobile devices that rely on GPRS communication). Finally, some environments might not provide any opportunities for tiering at all and here one is limited to external infrastructure (such as the cloud). Solutions for SFV have to be flexible in order to deal with these forms of heterogeneity and should try to shield users as much as possible from the heterogeneity of the underlying devices and infrastructure.

A final important aspect is that the actual sensor function virtualization should

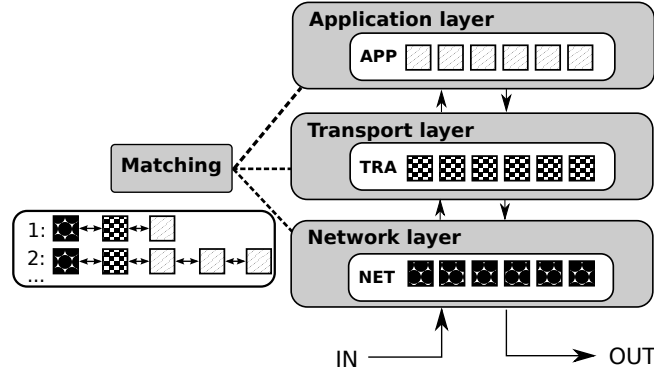


Figure 2.4: Architecture for sensor function virtualization in the Internet of Things

be transparent to end users. This means that any virtual functions that are added to devices should build on top of existing communication interfaces and that changes to protocols running on end hosts should be minimal and preferably non-existent. When SFV enhances physical devices, it should appear as if the constrained device offers the virtualized functions by itself from the user's point of view. When working with entirely virtualized devices, the interface to the user should be the same as the one that is used to communicate with constrained devices. If not, it will be cumbersome for users to discover and use the additional functionality.

Keeping in mind the requirements of the previous paragraphs, our proposed architecture is presented in figure 2.4. The basic principle is that sensor function virtualization is realized by decomposing the desired functionality into smaller modules. In the figure the modules are categorized according to the functionality that they provide (e.g. a module implementing 6LoWPAN compression would fall in the network category). The main benefit of this modular approach is that modules can be added at runtime (much like a plugin-based system) and that it allows to deploy modules over multiple machines thus improving scalability. The input and output data types for all types of modules are network packets. When network traffic arrives at a machine that provides sensor function virtualization, network packets flow up the architecture through a set of modules and down again through (possibly another set of modules) towards an outgoing network interface. The matching component takes care of passing network packets through the correct chain of modules based on configuration information that it stores. When combining all of these small modular functional blocks that are running on a number of machines, the distributed processing and in turn sensor function virtualization for the IoT are realized.

Depending on the functionality that is to be virtualized, one or more locations in the network are suitable for deploying the functionality. For example, if local

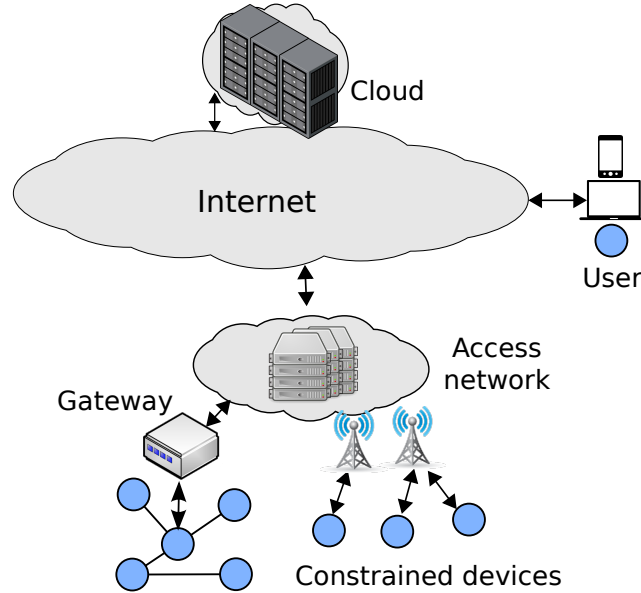


Figure 2.5: Infrastructure at different locations throughout the Internet works together to provide sensor function virtualization in the Internet of Things

operation (i.e. within the same network) is required in case of Internet failure, virtualizing functionality on cloud infrastructure probably is unfeasible. In this case the local gateway will play an important role in sensor function virtualization. If on the other hand, global operation is required at all times then always-online systems (such as the cloud, but potentially also the gateway if it is available) can bridge situations where a constrained device is unavailable for communication (e.g. it might be sleeping to conserve energy, or its GPRS connection might be unavailable). Packaging SFV in modular components that can be (re)deployed at runtime, ensure that our architecture is able to handle this kind of flexibility.

Figure 2.5 gives an overview of the different locations where sensor function virtualization can be realized. The colored disks at the bottom represent constrained devices. Note that mobile device might migrate to a different access network in case of inter-network mobility (only one access network is shown in the figure). The figure also displays potential users on the right hand side (note that a constrained device is also considered as a user). As the Internet - and by extension most standardization efforts by IETF (see section 2.4) - follows a host-centric approach our SFV architecture is designed to be compatible with this point of view. While SFV in effect moves functionality away from Internet hosts to supporting infrastructure, our approach allows extending constrained devices in a way that is transparent to its users. Transparency here means that it appears as if

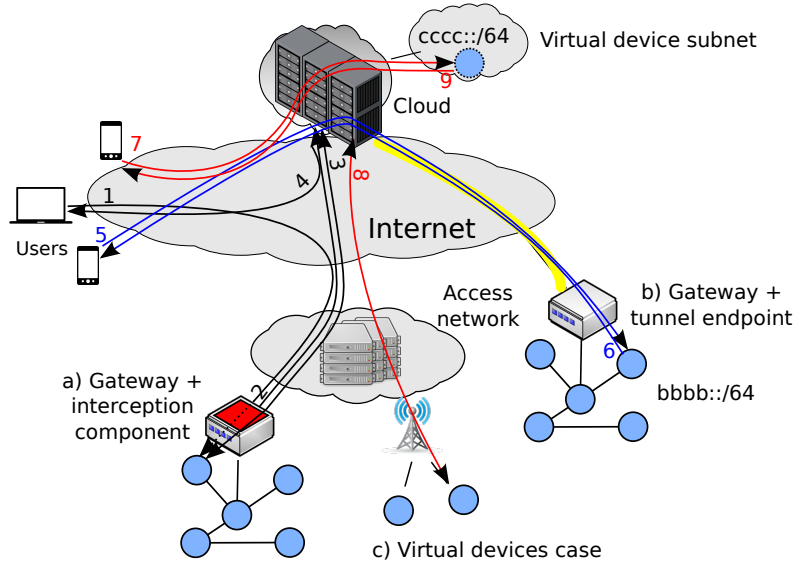


Figure 2.6: Three different integration strategies for cloud-based SFV

the functionality resides on the device itself, while in fact it is offered by supporting infrastructure. This is an important point, as this transparency differentiates our work from most of the cloud-based systems available today that each offer their own integration interface. This transparency means that users do not have to integrate with yet another cloud-based API as SFV functionality appears to be offered by the device itself. One consequence is that in order for a component of our architecture to enhance a constrained device with new functionality it must be able to receive and process requests for this device. For gateways that are on the routing path between a device and its Internet client this is trivial. For cloud-based infrastructure, extra measures are most likely necessary in all cases.

Figure 2.6 gives an overview of the mechanisms that ensure that cloud-based SFV infrastructure - or more generally any infrastructure that is not on the routing path between the user and the constrained device - is able to process requests destined and responses coming from the constrained devices that it handles. This is necessary if the cloud infrastructure should be able to modify requests and responses in order to fulfill its function in a transparent way (e.g. remove unnecessary data, add semantic descriptions, translate between data formats, etc.).

In case a (black arrows), an interception component deployed along the routing path is responsible for forwarding (matching) incoming network traffic for the constrained device (1) towards the cloud infrastructure (2). Once the request arrives in the cloud it can be processed, and a response can be generated immediately or a (modified) request can be sent towards the constrained device (3). In the latter

case, the interception component is also responsible for ensuring that responses pass via the cloud. Once the cloud has processed the response (e.g. to update a cache), it forwards the response to the user (4). Note that response processing by the cloud is optional in certain cases (e.g. in the case of a cache that is only running on the gateway). Depending on the use case, the interception component can be configured by the cloud to stop forwarding traffic of a particular stream after the cloud has processed a number of packets of said stream (e.g. in access control the denial/granting of access is configured into the gateway by the cloud). This avoids unnecessary forwarding of traffic to the cloud. In this case the gateway can also forward responses directly to the user.

In case b (blue arrows), the constrained devices receive an IPv6 address that is globally routable and that is routed via the cloud infrastructure. As a result the cloud is able to process networking traffic destined to constrained devices (5). When the cloud forwards the traffic to the constrained device, then this traffic has to be encapsulated on the public Internet (otherwise it would never reach its destination). The tunnel endpoint can be a gateway (if one is available) or could be the constrained device itself (depending on its processing capabilities). The former is more suitable for WSN-like networks, while the latter is applicable to mobile nodes that are capable of hosting a tunnel endpoint. Note that the tunnel endpoint should also send all returning traffic through the tunnel in this case (6). If not, the cloud infrastructure is unable to process the responses from constrained devices (making e.g. caching impossible).

In case c (red arrows), the cloud-based SFV infrastructure mirrors every constrained device via a virtual device that is allocated a globally-routable IPv6 address from a “Virtual device subnet” that is routed to the cloud. In this case users interact with the virtual device in the cloud (7). Here the cloud fulfills the role of a traditional reverse proxy. This mode of operation is still transparent in the sense that the virtual device offers the same interfaces as the actual constrained devices but in this case the user does have to communicate with an Internet host at a different IPv6 address. The cloud infrastructure processes requests from users destined to virtual devices and forwards these to the actual constrained devices (8). The responses arrive from the constrained device and are sent back to user (9). This case is actually a generalization of case b. Note that this mechanism can also be used as a light-weight alternative to mobile IPv6 for constrained devices where their anchor point on the Internet changes due to their mobility. In this case, users always communicate with the fixed “virtual device” and the cloud takes care of mapping requests to the volatile IP endpoint of the constrained device. Signaling of the volatile IP address can happen via existing interfaces.

2.4 IETF protocol stack for the Internet of Things

As the remainder of this paper will employ CoAP and the IETF IoT stack for illustrating SFV, a concise overview of all protocols involved is presented here. Ishaq et. al present a more elaborate overview of the subject in [3].

The IETF started standardization for the IoT with an adaptation protocol for IPv6 over 802.15.4 communication links. The resulting protocol is defined by RFCs 4944 [4] and 6282 [5] and is more generally known as 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks). 6LoWPAN allows compressing IPv6 and UDP packets by eliding redundant information in IPv6 and UDP headers. 6LoWPAN also provides fragmentation support for large IPv6 packets, thus fulfilling the minimum MTU requirement of 1280 bytes found in IPv6. As a result, deploying IPv6 (with its 40 bytes header) on links with small MTUs (such as 802.15.4 links with a 127B MTU) in a standard compliant matter has been possible since 2007. It is interesting to note that the idea of compression for IPv6 in 6LoWPAN is also being applied to other protocols (such as DTLS [6]).

Around 2008 consensus emerged in the IETF on standardizing a routing protocol for use in low power and lossy networks (LLNs). Due to the unique properties of LLNs (e.g. conserve energy as much as possible, point to multipoint traffic, etc.), their routing requirements differ from what traditional routing protocols considered at that time. The results of the ROLL working group include a number of requirements for specific types of LLNs (home automation, industrial, smart city, etc.) and a routing protocol known as the “IPv6 Routing Protocol for Low-Power and Lossy Networks” (RPL). RPL, standardized in RFC 6550 [7], allows constructing routing trees (known as DODAGs) for LLNs according to an objective function that can be tuned to meet the requirements specific to the type of LLN.

Application (CoAP)
Transport (UDP + DTLS)
Network (IPv6+RPL)
Adaptation (6LoWPAN)
MAC
PHY

Figure 2.7: IETF protocol stack for low power and lossy networks in the Internet of Things

In June 2014 the IETF Constrained RESTful Environments (core) working group standardized an application layer protocol for use in low power and lossy networks. The Constrained Application Protocol (CoAP), as defined in RFC 7252 [8], allows building applications based on the concepts of RESTful web services that are well-

known from the WWW. CoAP can be thought of as a lightweight alternative to HTTP and as the counterpart of HTTP for use in the embedded world (with battery operated devices, unreliable wireless links and low-cost 16 bit microcontrollers). Another important contribution of the working group is RFC 6690 [9] which specifies a format for web linking for constrained web servers. This format can describe hosted resources, provide attributes for resources and define relationships between links. It is designed with a simple parser in mind that is memory efficient and that provides compact web links. Figure 2.8 shows a typical request/response message exchange between a CoAP client and server. First the client discovers which resources are hosted by the server via the “.well-known/core” resource, the response contains a list of links according to the core link format. The second request retrieves a plain-text representation of the temperature resource from the server.

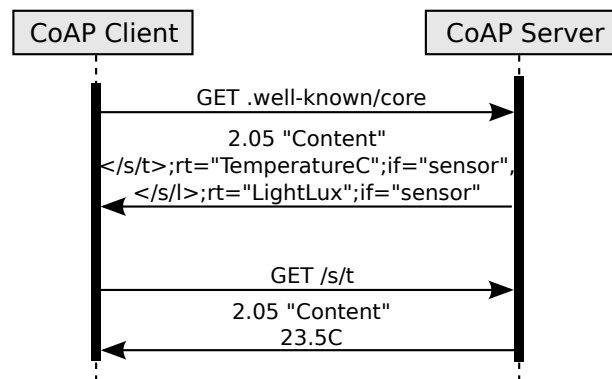


Figure 2.8: A CoAP request/response message exchange showing resource discovery and data retrieval

The CoRE working group identified that security is an important requirement for almost all IoT applications. Therefore it chose to rely on transport-layer security as this is a popular choice in the WWW today. More specifically, RFC 7252 defines a binding for CoAP that specifies how CoAP should be used in conjunction with DTLS. Datagram Transport Layer Security (DTLS, RFC 6347 [10]) provides end-to-end security via TLS over UDP (as opposed to TCP in case of TLS). When employing pre-shared key (PSK) cipher suites, DTLS can rely solely on symmetric encryption for providing end-to-end security. When combined with the small memory footprint of DTLS, this makes DTLS an attractive candidate for use in LLNs.

Finally, the IETF has identified the need for standardizing a set of DTLS parameters for use in LLNs. To this end, the DTLS In Constrained Environments (DICE) working group was formed in 2013. DICE is looking to standardize a LLN profile for DTLS, to overcome some of the issues of DTLS in combination

with multicast and to investigate practical issues surrounding the DTLS handshake in LLNs. DICE explicitly states that it will not alter the DTLS state machine.

Note that for GPRS equipped constrained devices, the 6LoWPAN and RPL standards are less applicable. These types of device do however still benefit from the low communication overhead offered by DTLS/CoAP when compared to TLS/HTTP [11]. An overview of the entire stack is shown in figure 2.7.

2.5 SFV in the unconstrained domain

As mentioned function virtualization enables us to extend constrained devices with new functions without burdening the constrained device itself. This is achieved by offloading (or virtualizing) functionality to more powerful infrastructure (such as gateways, routers, cloud-based infrastructure) in a way that is transparent to the constrained device and its users. This section presents two scenarios that demonstrate SFV according to the architecture presented in section 2.3. Note that these two scenarios serve as illustrations of SFV and that SFV as a technique is not limited to what is shown here (for example SFV at the network layer is not shown here). The scenarios are applied to the IETF IoT stack that was detailed in the previous section.

The first case considers constrained devices that are only intermittently reachable via the Internet. These devices usually sleep for prolonged periods of time in order to keep energy consumption to a minimum. Every once in a while (e.g. after a certain period of time has passed or after an event is triggered) the device wakes up, sends its sensor data to a so-called mirror server, checks whether there is any incoming (configuration) data and goes back to sleep. Users do not interact with the sleepy device, instead they communicate with a mirror server that is always online. In CoAP this kind of functionality is known as a CoRE Mirror Server [12]. One problem with this approach is that mirror servers host mirrored resources on behalf of a multitude of sleepy devices. As a result, the user is exposed to the mirrored resources belonging to all sleepy devices that are interacting with the mirror server. This is problematic when the user in question is unfamiliar with and does not support the mirror server client operation interface. As a solution, SFV hides this client operation interface and instead provides a dedicated CoAP server for every device that is mirrored on the mirror server. This dedicated CoAP endpoint is always online and hides the underlying mirror server's interfaces from CoAP clients.

Apart from hiding the mirror server's client operation interface to clients, this scenario also illustrates how SFV can extend a device by hosting new CoAP resources on behalf of the device. In this case, SFV emulates CoAP resources via the dedicated CoAP endpoint that provides semantic descriptions of the resources offered by the constrained device. Because these descriptions are too large to store

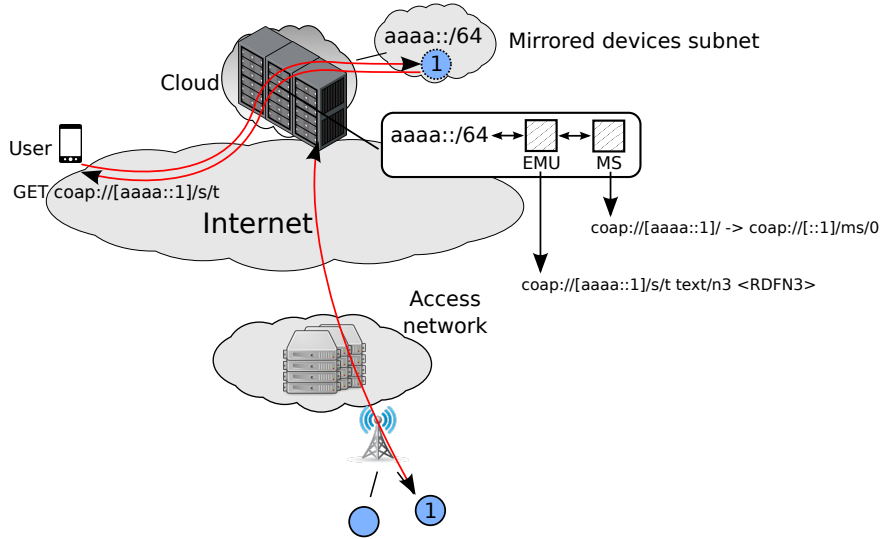


Figure 2.9: Two SFV modules providing emulated resources (EMU) and a mirror server abstraction (MS)

on the device itself, it is more efficient to offer them by means of SFV.

In order to realize this scenario in accordance with the architecture presented in figure 2.4, two SFV modules are needed. The MS module provides the dedicated CoAP endpoint that hides the mirror server's interface. The EMU module implements the CoAP resource emulation. The matching component also has to be configured in order to match requests for CoAP resources on the dedicated endpoint to the two modules. The result is shown in figure 2.9. The constrained device (near the bottom in the figure) wakes up periodically to push its sensor data to the mirror server in the cloud as per [12]. Users send their CoAP requests to the mirrored devices, which reside in a subnet that is routed towards the cloud (aaaa::/64 in the figure). Requests are handled by the SFV matching component in the cloud (the white box in the figure). The matching component passes the requests to the EMU module. If this module does not generate a response, then the request is passed along to the MS module. In the figure, the EMU module is configured to emulate one resource at `coap://[aaaa::1]/s/t` for the content-type "text/n3". This emulated resource provides a semantic description of a temperature sensor in the RDF Notation3 format. The response of the resource (`<RDFN3>`) is shown in the listing below.

```
<coap://[aaaa::1]/s/t>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.oclc.org/NET/ssnx/ssn#Sensor>;
<http://spitfire-project.eu/ontology/ns/obs>
<http://vmuss07.deri.ie:8182/ld4s/res/property/temperature>;
```

```

<http://spitfire-project.eu/ontology/ns/uom>
<http://vmuss07.deri.ie:8182/ld4s/res/uom/centigrade>;
<http://purl.oclc.org/NET/ssnx/ssn#onPlatform> <coap://[aaaa::1]>;
<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#hasLocation> "Testbed iMinds";
<http://purl.oclc.org/NET/ssnx/ssn#featureOfInterest> "iMinds Office" .

```

Listing 1: RDFN3 description for a temperature CoAP resource

If the EMU module does not generate a response, then the request is handled by the MS module. The MS module simply rewrites CoAP requests for mirrored devices to requests for the mirror server. To accomplish this, it stores a mapping between the two. For example, in the figure a request to `coap://[aaaa::1]:s/t` will be mapped to `coap://[::1]/ms/0/s/t`. The user is totally oblivious that its request is rewritten by the module. Responses sent to the user by the MS module use the destination address of the original request as the source address, e.g. for the example responses come from `aaaa::1`.

The second case considers a Wireless Sensor and Actuator Network (WSAN) that deploys the full IETF IoT stack from figure 2.7 and that relies on DTLS for end-to-end security. In this network all constrained devices are accessible via their publicly routable IPv6 address and users interact with the CoAP resources that are hosted on the constrained devices. Optionally, devices employ radio duty cycling to prolong battery life. In such a scenario, DTLS sessions with the sensors and actuators are typically restricted to cipher suites that exclude any asymmetrical encryption due to the high complexity and computational cost of the algorithms involved. Furthermore, transporting and verifying certificates as is necessary in a Public Key Infrastructure (PKI) is also deemed too expensive for sensors and actuators. Instead, the constrained devices rely on symmetrical encryption algorithms (AES being a common choice) and Pre-Shared Key (PSK) cipher suites (TLS_PSK_WITH_AES_128_CCM_8 is a popular suite) due to the relatively simple algorithms (AES co-processors are common) and small amount of keying material that has to be communicated respectively.

One issue in this scenario is that PSK cipher suites do not scale well as the number of clients increases. A sensor and or actuator (sensa) has to share a unique pre-shared key with every client. Obviously, this does not scale. A second issue arises due to the end-to-end encryption (E2EE) itself. As one would expect in E2EE, intermediary systems are unaware of the contents of the requests/responses between a client and the sensa. However, in WSANs processing at the edge of the network by a trusted intermediary can significantly improve response times and battery lifetimes. This consideration applies to techniques such as caching and access control. Ideally, we would want a variant of E2E security where the WSAN gateway (which is considered to be trusted by both the client and the sensa) is able to decipher communications in order to provide caching, access control, etc. To summarize, the low scalability of PSK cipher suites and the inability to perform any processing at the edge of the network (e.g. caching) are two problems inherent to DTLS in WSANs today.

SFV can provide a solution to these two problems by virtualizing asymmetrical encryption and by extending the WSAN gateway up to the application layer (in

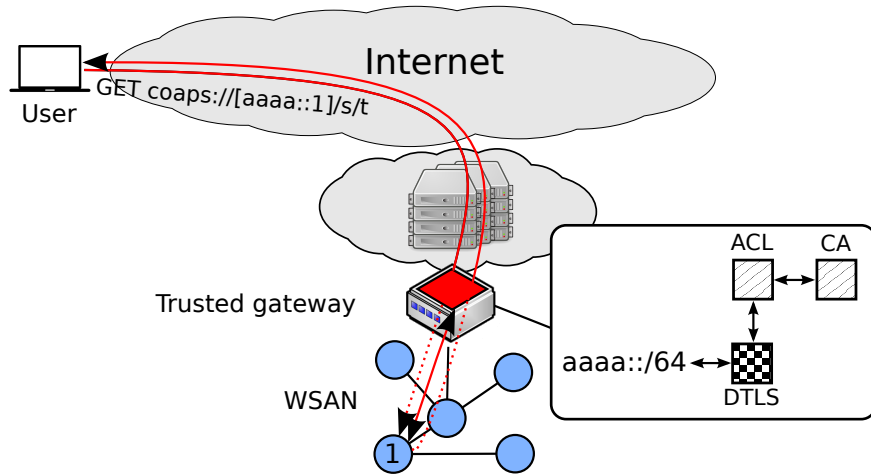


Figure 2.10: SFV at a trusted gateway provides DTLS termination, access control and caching

effect realizing an application-level gateway). SFV on the trusted gateway intercepts traffic and terminates DTLS sessions between clients and sensas. As a result, the DTLS handshake of the client is performed with the trusted gateway. Note that this happens completely transparent to the client, i.e. the client still communicates with the public IPv6 address of the sensa (as the gateway is assumed to be trusted, this is considered secure). The trusted gateway performs a DTLS handshake with the sensa using a PSK cipher suite and all DTLS traffic of the client is sent over this session. This way, the sensa only has to share a unique PSK with the gateway. Furthermore, the gateway can offer a more extensive list of supported cipher suites to the client. This list can include suites that rely on PKI, thereby circumventing the scalability problem of the PSK suite offered by sensas. Finally, by terminating DTLS sessions the trusted gateway is also able to perform edge processing such as caching and access control.

Figure 2.10 presents an overview of this case. Preliminary results show that terminating DTLS and re-using the same DTLS session in the WSN for multiple clients can reduce the energy consumption of the constrained device by more than 59% when compared to traditional end-to-end security where the trusted gateway does not terminate the session. This large reduction in expended energy is primarily due to the fact that the constrained device only has to setup one DTLS session with the trusted gateway for all clients. As a result, the expensive DTLS handshake is only executed once whereas in the traditional end-to-end case the handshake has to be repeated for every separate client. Because the gateway is able to see the contents of the CoAP requests, it can apply access control on a per resource basis.

For example, users that are authenticated with a specific certificate might be able to reconfigure resources (i.e. PUT and POST methods are allowed for specific resources), whereas others are only allowed to retrieve data (i.e. only the GET method is allowed for specific resources). Furthermore, the DTLS termination module can be configured to omit DTLS in the WSN for certain requests that do not require authentication and/or confidentiality in the WSN. For the latter, consider an example where retrieving a temperature value is done in plain-text in the WSN, but is transferred in cipher text over the public Internet. Note that these kinds of advanced scenarios are very hard to realize without a distributed approach due to constraints of the IoT devices. This is where SFV really shows its potential.

2.6 SFV in the constrained domain

In some cases, sensor function virtualization might be required on the constrained devices themselves. For SFV on constrained devices the architecture from figure 2.6 is often not a good fit, as their characteristics differ greatly from what is available on more powerful hardware such gateways and the cloud. Nevertheless, in previous work we have shown that some form of flexible distributed computing can be provided for constrained devices. Other approaches that target reprogrammability of constrained devices can also achieve the desired flexibility at a higher cost due to the expensive operation of transferring a program's memory contents.

Bindings [13] allow a third party to setup direct CoAP-based interactions between sensors and actuators and other devices. One of the benefits is that the third party does not have to be a message broker between the sensor and the actuator and therefore can go offline after setting up the binding. Furthermore, this gives us the flexibility to link sensors to actuators after they are deployed instead of when they are produced.

Another concept that we have presented in previous work [14] are small REST web services that fulfill common tasks such as filtering and (de)multiplexing of sensor data. These web services can be deployed inside the constrained network (on the gateway but also on constrained devices) and are called RESTlets. RESTlets have one or more inputs and zero or more outputs. All in- and outputs are RESTful web services. RESTlets allow decomposing parts of the data processing into smaller blocks that can be deployed close to the data sources and their consumers. Because requests and responses do not have to traverse the entire WSN, response times and energy consumption are lowered in multi-hop WSNs.

2.7 Related work

Thirty-five years after Cerf V. and Kirstein P. argued about the role of gateways in what was then the early Internet, the authors revisit the same question at the advent of the Internet of Things [15]. Very similar to the early Internet, there exist a num-

ber of proprietary protocols for connecting things to the Internet today. The authors argue that there will first be a period of adaptation for connecting legacy technologies to the IoT, followed by period of adoption where the IP will supersede these legacy technologies. In terms of adaptation, the authors introduce repositories as the end-point for interactions of clients with things. These repositories have direct access to sensors to gather data or to set configuration information. Repositories act as a bridge between clients and sensors and apply adaptations where necessary. The authors also mention that repositories can be part of a distributed system. We envision that these repositories are deployment options for our SFV concept. Our work does not only focus on adapting proprietary technologies for Internet integration, SFV also enhances things that already deploy the IETF IP stack. This is necessary in order to bridge the gap in processing power and capabilities between constrained devices and traditional Internet devices.

Varakliotis et al. [16] build further on the vision presented by Cerf V. & Kirstein P. The authors describe which tasks are common for a gateway and decompose these into smaller blocks that are situated at various layers in the protocol stack. These functional blocks are very often required but need not be co-located on one and the same machine, e.g. they could be distributed on multiple servers. The result is a distributed protocol stack, with a minimal controller and external servers that hold most of the functional blocks. The IETF stack is also considered as one of the potential technologies for IoT networks (i.e. the DevNet). Our vision aligns closely to that of Varakliotis et al. However, the assumption that a distributed approach will automatically lead to a stripped gateway with minimal functionality and external servers doing the heavy-lifting does not hold in all cases. Consider an example where things generate large amounts of data that cannot be transported in their raw format to external infrastructure. In such a case, it can be beneficial to do some form of processing locally at the gateway. In contrast to the work of Varakliotis et al., our SFV architecture does not exclude this possibility.

Another interesting work to mention in this context is the IPv6 addressing proxy by Jara et al. In [17] an adaptation proxy is presented for mapping native addressing from legacy technologies to the IPv6 Internet of Things. The authors provide extensive examples for legacy technologies such as X10, EIB, CAN and RFID from the industrial, home automatic and logistics domains. Our SFV concept is an ideal candidate for implementing such an addressing proxy.

In “Moving application logic from the firmware to the cloud” [18] Kovatsch et al. argue that application development should be fully decoupled from the embedded domain. Embedded devices are thin servers that export only elementary functionality by means of REST resources. The approach relies on application servers that house the logic of applications. Our SFV concept can be seen as a distributed version of these application servers. SFV also allows to transparently enhance embedded devices for all clients involved, thus the virtualized functionality becomes available to all parties interacting with the device and not only to the application server. As a result, our work should facilitate easier reuse of application logic than the application servers presented by Kovatsch et al.

PyoT [19] is a programming framework for the IoT that claims to simplify IoT

application development by providing a number of common operations to developers. Supported operations include discovery, monitoring, storage and triggering of devices and their data. The PyoT framework can reside on a gateway or in the cloud. Furthermore, PyoT also integrates with T-Res [20], a framework enabling “in-network processing” in IoT-based WSNs. The existence of the PyoT framework shows that extending constrained devices with common operations can help to accommodate IoT application development. This is also one of the objectives of SFV. T-Res is an alternative approach to SFV in the constrained domain, as discussed in section 2.6, that relies on programmability via a minimal python interpreter suitable for embedded systems (PyMite).

Finally, there are two other works that discuss the role of cloud computing in the Internet of Things. In [21] Pereira et al. present a holistic network architecture for supporting mobility in IoT applications. This service-oriented architecture can be deployed on three levels: on the node itself (for resilience against lost Internet connectivity), on a SOA-enabled gateway (when more processing than what a node can provide is necessary) and on the Internet (where all limitations of processing power are mitigated). In [22], Zhou et al. present a common architecture for integrating the Internet of Things with cloud computing that is named CloudThings. The authors describe a cloud-based Internet of Things platform for developing, deploying, running, and composing Things applications. CloudThings differs from PyoT in that it focuses on providing a development platform (that supports IaaS, PaaS but also SaaS) as opposed to a more rigid development framework. These two works illustrate how distributed processing can extend constrained IoT devices, which is one of the key concepts of SFV.

None of the identified related work considers reconfiguring the network to optimally support an IoT application’s requirements as an integral part of their approach. In our work this is an important aspect of distributed intelligence, as in some cases (cfr. section 2.2.2.1) the network has to be able to adapt in order to fulfill the requirements of the IoT application.

2.8 Conclusions

In this paper we presented some of the limitations that are present in today’s IoT systems. We argued that some of these issues can be overcome by means of distributed intelligence. In distributed intelligence the network and its infrastructure can be reconfigured to meet application requirements and the resources offered by these systems can be leveraged to provide processing and communication at the right place and time according to the requirements of IoT applications. Sensor function virtualization is a technique that enables distributed intelligence via modular functional blocks that can be deployed anywhere in the network infrastructure. This way processing can be placed where it is most needed. Furthermore, SFV can also be used to expose configuration interfaces for existing network functionality (thus enabling the reconfiguration required in distributed intelligence). By means of two extensive examples, we have illustrated some of the benefits that

SFV can bring to the table of IoT application development and distributed IoT systems.

In the future, we plan to evaluate our approach more thoroughly. The reduction in energy consumption and latency in the DTLS termination use case already show that sensor function virtualization can have a significant impact on important performance metrics for constrained devices and IoT applications. However, more experiments are necessary. Furthermore, reconfiguring the communication network (e.g. routing, MAC protocols, etc.) in order to support specific communication patterns required by an IoT application was only briefly discussed. More work is needed to explore what is possible and how well this aligns with the vision of distributed intelligence that was outlined in this paper.

Acknowledgement

The authors would like to acknowledge that part of this research was supported by the COMACOD project. The iMinds COMACOD project is co-funded by iMinds (Interdisciplinary institute for Technology) a research institute founded by the Flemish Government. Partners involved in the project are Multicap, oneAccess, Track4C, Invenso and Trimble, with project support of IWT.

References

- [1] D. Evans. *The internet of things: how the next evolution of the internet is changing everything*, 2011. Available from: [https://www.cisco.com/c/dam/en_{-}us/about/ac79/docs/innov/IoT_{-}IBSG_{-}0411FINAL.pdf](https://www.cisco.com/c/dam/en/_us/about/ac79/docs/innov/IoT_{-}IBSG_{-}0411FINAL.pdf).
- [2] *Light-Weight Implementation Guidance (lwig) - Charter*, 2011. Available from: <https://datatracker.ietf.org/wg/lwig/charter/>.
- [3] I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. Van den Abeele, E. De Poorter, I. Moerman, and P. Demeester. *IETF standardization in the field of the Internet of Things (IoT): a survey*. Journal of Sensor and Actuator Networks, 2(2):235–287, 2013.
- [4] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. *RFC 4944: Transmission of IPv6 packets over IEEE 802.15. 4 networks*, 2007. Available from: <https://tools.ietf.org/html/rfc4944>.
- [5] J. Hui and P. Thubert. *RFC 6282: Compression format for IPv6 datagrams over IEEE 802.15. 4-based networks*, 2011. Available from: <https://tools.ietf.org/html/rfc6282.txt>.
- [6] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. *Lithe: Lightweight secure CoAP for the internet of things*. IEEE Sensors Journal, 13:3711–3720, 2013. doi:10.1109/JSEN.2013.2277656.
- [7] T. Winter and P. Thubert. *RFC 6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, 2012. Available from: <https://tools.ietf.org/html/rfc6550>.
- [8] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. *RFC 7252: Constrained Application Protocol (CoAP)*, 2014. Available from: <https://tools.ietf.org/html/rfc7252>.
- [9] Z. Shelby. *RFC 6690: Constrained RESTful Environments (CoRE) Link Format*, 2012. Available from: <https://tools.ietf.org/html/rfc6690>.
- [10] E. Rescorla and N. Modadugu. *RFC 6347: Datagram transport layer security version 1.2*, 2012. Available from: <https://tools.ietf.org/html/rfc6347>.
- [11] L. Daniel, M. Kojo, and M. Latvala. *Experimental Evaluation of the CoAP, HTTP and SPDY Transport Services for Internet of Things*. In 7th International Conference on Internet and Distributed Computing Systems, pages 111–123, 2014.
- [12] M. Vial. *CoRE Mirror Server*, 2013. Available from: <https://tools.ietf.org/html/draft-vial-core-mirror-server-01>.

- [13] G. K. Teklemariam, J. Hoebeke, I. Moerman, and P. Demeester. *Flexible, direct interactions between CoAP-enabled IoT devices*. In The Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2014), 2014.
- [14] G. K. Teklemariam, J. Hoebeke, F. Van den Abeele, I. Moerman, and P. Demeester. *Simple RESTful Sensor Application Development Model Using CoAP*. In 9th IEEE Workshop on Practical Issues in Building Sensor Network Applications (IEEE SenseApp 2014), pages 552–556, 2014.
- [15] V. Cerf and P. Kirstein. *Gateways for the Internet of Things-An Old Problem Revisited*. In Global Communications Conference (IEEE GLOBE-COM), pages 2641–2647, 2013. Available from: <http://discovery.ucl.ac.uk/1414930/>.
- [16] S. Varakliotis, P. T. Kirstein, A. Jara, and A. Skarmeta. *A process-based Internet of Things*. In 2014 IEEE World Forum on Internet of Things (WF-IoT), pages 73–78. Ieee, mar 2014. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6803123>, doi:10.1109/WF-IoT.2014.6803123.
- [17] A. J. Jara, P. Moreno-Sanchez, A. F. Skarmeta, S. Varakliotis, and P. Kirstein. *IPv6 addressing proxy: mapping native addressing from legacy technologies and devices to the Internet of Things (IPv6)*. Sensors (Basel, Switzerland), 13(5):6687–712, jan 2013. Available from: <http://www.mdpi.com/1424-8220/13/5/6687/htm>, doi:10.3390/s130506687.
- [18] M. Kovatsch, S. Mayer, and B. Ostermaier. *Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things*. ... Mobile and Internet ..., 2012. Available from: <http://ieeexplore.ieee.org/xpls/abs{ }all.jsp?arnumber=6296948>.
- [19] A. Azzara, D. Alessandrelli, S. Bocchino, M. Petracca, and P. Pagano. *PyoT, a macroprogramming framework for the Internet of Things*. In Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014), pages 96–103. Ieee, jun 2014. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6871193>, doi:10.1109/SIES.2014.6871193.
- [20] D. Alessandrelli, M. Petraccay, and P. Pagano. *T-Res: Enabling Reconfigurable In-network Processing in IoT-based WSNs*. In 2013 IEEE International Conference on Distributed Computing in Sensor Systems, pages 337–344. IEEE, may 2013. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6569453>, doi:10.1109/DCOSS.2013.75.
- [21] P. P. Pereira, J. Eliasson, R. Kyusakov, J. Delsing, A. Raayatinezhad, and M. Johansson. *Enabling Cloud Connectivity for Mobile Internet of Things Applications*. 2013 IEEE Seventh International Symposium on

Service-Oriented System Engineering, pages 518–526, mar 2013. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6525570>, doi:10.1109/SOSE.2013.33.

- [22] J. Zhou, T. Leppanen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, and H. Jin. *CloudThings: A common architecture for integrating the Internet of Things with Cloud Computing*. In Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2013, pages 651–657, 2013. doi:10.1109/CSCWD.2013.6581037.

3

Secure Service Proxy: A CoAP(s) Intermediary for a Securer and Smarter Web of Things

The motivation for this chapter stems from the issues we encountered when developing application in a secure, constrained and open Web of Things (WoT). Specifically, issues related to end-to-end security and missing features in Constrained RESTful Environments (CoRE) led to this work applying Sensor Function Virtualization (SFV), introduced in the previous chapter, to try and solve these problems. To this end, the design of a secure, reverse proxy that extends virtual devices with modular functionality is presented in this chapter. We demonstrate how, by combining device virtualization and reverse proxying, the proposed proxy is able to address the aforementioned issues.

Floris Van den Abeele, Ingrid Moerman, Piet Demeester and Jeroen Hoebeke

Published in MDPI Sensors, Volume 17, Issue 7, 2017.

Abstract As the IoT continues to grow over the coming years, resource-constrained devices and networks will see an increase in traffic as everything is connected in an open Web of Things. Performance- and function-enhancing features are

difficult to provide in resource-constrained environments, but will gain importance if the WoT is to be scaled up successfully. For example, scalable open standards-based authentication and authorization will be important to manage access to the limited resources of constrained devices and networks. Additionally, features such as caching and virtualization may help further reduce the load on these constrained systems. This work presents the Secure Service Proxy (SSP): a constrained-network edge proxy with the goal of improving the performance and functionality of constrained RESTful environments. Our evaluations show that the proposed design reaches its goal by reducing the load on constrained devices while implementing a wide range of features as different adapters. Specifically, the results show that the SSP leads to significant savings in processing, network traffic, network delay and packet loss rates for constrained devices. As a result, the SSP helps to guarantee the proper operation of constrained networks as these networks form an ever-expanding Web of Things.

3.1 Introduction

In recent years, the Internet of Things (IoT) has increasingly become a hot topic in industry, academia, the do-it-yourself community and also consumers. Businesses are attracted by the new product opportunities and new sources of revenue that the IoT promises to bring. For example, a 2013 market report on IoT by Cisco Inc. (USA, CA, San Jose) predicts 14.4 trillion USD in created value for the “Internet of Everything” from 2013 to 2022 [1]. Academia is interested in the many new problems and issues that arise when deploying billions of devices on the Internet. These issues include big data analytics, energy efficient communications, large-scale deployments, management of devices, communication protocols, security models, data privacy and many more. An introduction to the research aspect of the IoT is presented in [2]. Finally, consumers are drawn to the IoT because IoT products promise to bring improvements and novel services to their daily lives. Examples of IoT domains include smart home, smart health, smart transportation, smart factory, smart grid and many more [3].

As the Internet of Things continues to grow in scope and in size, the number of available technologies and platforms that promise to enable the IoT keeps increasing. As a family of such technologies, a complete protocol stack was standardized at the Internet Engineering Task Force (IETF) for use with constrained IoT devices in Low-power and Lossy Networks (LLNs) [4]. This suite of protocols defines the communication stack from the network layer up to the application layer. In contrast to the popular alternative ZigBee [5], the IETF protocol stack gives the developer more flexibility to model the network and the application to a specific use-case. For instance, with the IPv6 Routing Protocol for Low-power and Lossy Networks (RPL) [6] the routing can be tuned by employing different objective functions that optimize routes according to the metrics that are relevant to the use case (e.g., minimize hop count, maximize battery lifetime, etc.). Another example of flexibility is found at the application layer, where the REST ar-

chitecture followed by the CoAP protocol allows developers to design their own RESTful resources and to model their behavior. In terms of security, the IETF elected to standardize an End-to-End (E2E) architecture as it is a popular choice on the unconstrained Web today. Therefore, the CoAP standard defines DTLS (i.e., Datagram TLS) as its recommended security method.

Secure Sockets Layer (SSL) and, later, Transport Layer Security (TLS) have been around since the end of the past century and have become very popular protocols for their roles in securing the WWW. Today, (D)TLS has become a flexible protocol where endpoints can negotiate the type of security and where a built-in extension mechanism allows one to add new features to the protocol without touching the base specification. A comprehensive overview of the (D)TLS protocol is presented in the Background Section 3.2. Widespread adoption, a wide range of implementations, an open protocol specification and a high level of interoperability are just a few of the benefits of the TLS protocol. Nevertheless, one should be careful when deploying end-to-end security with DTLS in constrained environments. This issue has been recognized by the IETF, which has formulated guidance for implementing and deploying DTLS in constrained environments in RFC 7925 [7].

Despite the advantages offered by DTLS, E2E security has a number of disadvantages when deployed as-is in LLNs. One issue with E2E security is that it completely blocks out any third party (e.g., intermediate middleboxes) from taking part in the communication. In most traditional Internet deployments, this is a wanted property of E2E security, but in LLNs, it stops intermediary systems from providing services that can improve resource usage and the performance of constrained devices and networks. For example, caching of CoAP responses is not possible when E2E security is applied between the CoAP client and the constrained CoAP server. A second disadvantage of E2E security is that application-layer enhancements cannot be applied by middleboxes, as all communication is enciphered. Thus, access control, admittance control and other similar features cannot be provided at the edge of the LLN. Another known problem with DTLS is its performance in duty-cycled networks, which is common in multi-hop LLNs. Research [8] has shown that the latency introduced by the DTLS handshake can become excessively large in multi-hop duty-cycled networks (up to 50 s for four hops). Vućinć et al. also show that constrained nodes can only store a limited number of DTLS sessions in their memory (e.g., max. three DTLS sessions for a WiS-Mote node). As a result, nodes have to start dropping active DTLS sessions from memory, which can deteriorate battery lifetime and DTLS performance. Finally, end-to-end network addressing reduces the effectiveness of 6LoWPAN compression. This is due to the fact that the IPv6 prefixes for nodes situated on the Internet and the used UDP ports are difficult or impossible to compress on 6LoWPAN. All of these issues are covered in greater depth in the problem statement, cf. Section 3.3.

The goal of this work is to overcome the issues identified with E2E security without losing the benefits offered by such a widely-used protocol as DTLS. To this end, we propose the “Secure Service Proxy” (SSP). It is a reverse DTLS and

CoAP proxy that provides a secure bridge between clients on the Internet and constrained IoT devices in a low-power and lossy network. By employing DTLS on both legs of the communication path, the resulting system can still enjoy most of the benefits offered by the popularity of DTLS without suffering from the disadvantages of E2E security specific to constrained environments (as identified in the previous paragraph). As the SSP operates as a trusted entity in the network, it can also offer network services such as caching, as well as application-layer enhancements. For the latter, this paper employs the concept of node virtualization where a constrained node has a virtual counterpart that resides on the proxy and that offers additional functionality on behalf of the node. This virtualization concept is effective because the SSP is deployed on hardware more powerful than the constrained nodes themselves. As a result, node virtualization can offer new and complex functionality that is unfeasible to offer on the constrained node itself. Examples include support for more complex modes of DTLS (e.g., public key infrastructure and certificate-based suites), translating responses between content formats, offering verbose semantic descriptions for the constrained node, storing large binary blobs (e.g., a picture of the deployment area), keeping historical data, etc.

Our contributions in this paper are as follows. First, we identify and discuss a number of issues with end-to-end security in constrained RESTful environments. We argue that these issues can be overcome by a reverse proxy approach that splits the end-to-end security at the proxy. Secondly, we design and implement such a reverse proxy. Apart from solving the E2E security issues, our developed proxy can also offer additional functionality and services on behalf of the constrained network and the constrained nodes. To our knowledge, this work is the first to study, design, implement and evaluate a reverse proxy for use with end-to-end security in constrained RESTful environments. Finally, by means of a real-world evaluation, we show that our work can significantly improve the operation of constrained networks by reducing power consumption, network latency and network traffic.

The rest of this paper is structured as follows. First, a brief overview of CoAP and DTLS is presented in the next section. Using this overview, a number of issues with deploying CoAP and DTLS in low-power and lossy networks is presented in Section 3.3. This section also lists the research goals of this work. In Section 3.4, our approach to tackling these issues is presented together with the design of the secure service proxy and an overview of the security risks related to breaking end-to-end security. The secure service proxy is aligned to similar work in the literature and the commercial world in Section 3.5. An extensive evaluation of our approach based on both simulations and a real-world wireless sensor network testbed is presented in Section 3.6. Section 3.7 presents the conclusions that are drawn from this work.

3.2 Overview of CoAP and DTLS

3.2.1 The Constrained Application Protocol (CoAP)

RFC 7252 [9] states that the Constrained Application Protocol (CoAP) is a specialized Web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things. The protocol is designed for Machine-to-Machine (M2M) applications such as smart energy and building automation. The main design considerations for CoAP include simplicity, very low overhead, easy translation to and from HTTP and support for multicast.

In CoAP, constrained devices that host applications structure their data and actions as RESTful Web services, also called CoAP resources. CoAP clients send requests to resources in order to retrieve and store data or trigger actions. CoAP defines the same request methods as HTTP: GET, PUT, POST and DELETE. They are used respectively for retrieving data, storing data, toggling an action and removing data. CoAP chose UDP as its transport protocol due to the lightweight nature of UDP (TCP was deemed too verbose due to its connections and too complex to implement in constrained devices). Therefore, CoAP includes a simple reliability layer and deduplication mechanism in order to compensate for the minimalistic nature of UDP. In order to minimize overhead, CoAP uses a binary format for encoding message options in the headers of CoAP requests and responses. As a result, the CoAP message size is significantly reduced when compared to a non-binary encoded protocol, such as HTTP [10], which is important in LLNs where message sizes are typically small and communication is expensive for battery-powered devices.

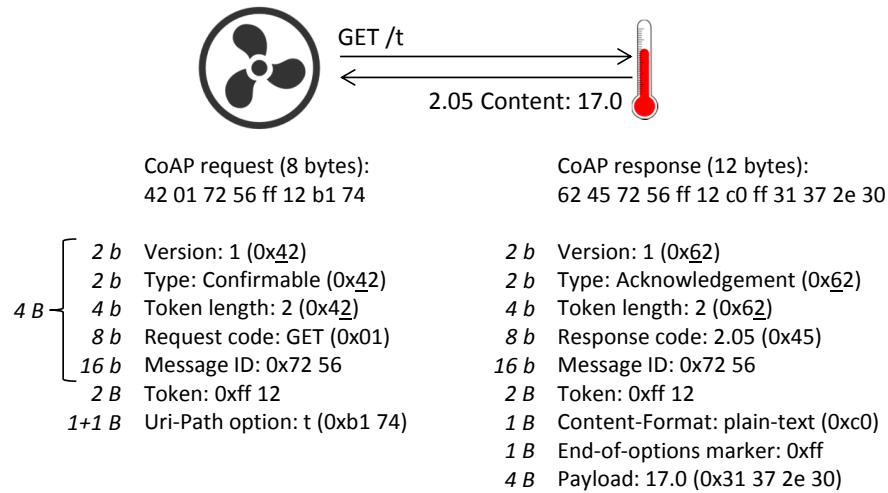


Figure 3.1: Anatomy of a typical CoAP request and response.

An illustration of a typical CoAP request/response exchange is shown in Figure 3.1, where a client (a ventilation unit) retrieves a temperature resource on a

CoAP server. The first elements of the CoAP header are the two-bit protocol version (RFC 7252 standardizes Version 1) and the two-bit message type. By sending a confirmable message, a sender can ask a receiver to acknowledge the reception of a message. This is reflected in the message type of the response, which is an acknowledgment. In most cases (like here), the response message is actually piggy-backed on the acknowledgment message in order to reduce the number of messages. The four-bit token length comes after the message type in the CoAP header, and it represents the length of the optional message token in bytes. The next element of the CoAP header is the eight-bit message code, which consists of a three-bit class and a five-bit subfield. Requests codes are Class 0 codes (e.g., GET is code 0.01), and successful response codes are Class 2 codes (e.g., Content is code 2.05). The final part of the fixed four-byte CoAP header is the two-byte message ID. It is used for deduplication and for confirmable (CON) messages, where acknowledgments echo the message ID of the CON message. The token is used to match a response with a request and can vary in length between zero and eight bytes. After the token come the header options and the payload (if any). In CoAP, header options are assigned unique numbers by IANA and are delta encoded in CoAP messages in order to reduce their encoding size. Every option encoding contains the delta of the option number (relative to the preceding option), the size of the value of the option (in bytes) and the value of the option. Finally, the options and the payload are separated by an end-of-options marker (0xff).

The CoAP Observe option [11] is a CoAP protocol extension that is important for this work. When a client is observing a REST resource on a CoAP server, the server will notify the client of state changes for that resource. This frees the client from polling the resource on the server, which can save resources in LLNs when changes in resource state occur rarely. RFC 7641 [11] also states that intermediaries must aggregate observation registrations: “If two or more clients have registered their interest in a resource with an intermediary, the intermediary **MUST** register itself only once with the next hop and fan out the notifications it receives to all registered clients. This relieves the next hop from sending the same notifications multiple times and thus enables scalability”. Apart from enabling scalability, aggregation also saves resources.

3.2.2 Datagram Transport Layer Security (DTLS)

For security, CoAP standardized end-to-end security and DTLS as its default security mechanism and protocol respectively. The primary motivation for preferring transport-layer security over alternatives such as object security and network layer security is the popularity of TLS on the conventional Web. Datagram TLS is by design very similar to the TLS protocol, and the specification of DTLS is largely written as a set of changes to the TLS specification [12]. However, there are some key differences as DTLS runs over an unreliable datagram transport while TLS runs over a reliable TCP transport. Therefore, DTLS must cope with the unreliable and unordered delivery of packets as available in TLS. To this end, DTLS introduces a simple timeout and retransmission scheme and adds an explicit sequence number

to the Record Protocol (versus an implicit number as available via TCP in TLS). Another difference is that stream ciphers must not be used with DTLS. DTLS also enhanced the handshake protocol with a stateless cookie exchange for denial of service resistance. By forcing DTLS clients to echo the cookie in their second handshake message, malicious clients (e.g., those spoofing IP addresses) can be rooted out, and a DTLS server can avoid wasting resources on bogus handshakes.

DTLS is a session-based protocol in that DTLS endpoints have to set up a session when they want to communicate securely. Negotiation of the security parameters for the session and peer authentication are both performed during the handshake phase of the protocol. After the handshake phase, both endpoints can exchange data with guarantees for confidentiality, endpoint authentication and integrity of the data. To this end, DTLS employs symmetric cryptography for data encryption according to an encryption algorithm and encryption keys that are agreed upon during the handshake. DTLS also guarantees message integrity by means of Hash-based Message Authentication Codes (HMAC). Sessions are typically negotiated on an ad hoc basis, although long-term sessions and resumption of established sessions are possible in DTLS.

TLS introduces the concept of cipher suites; these are named combinations of the authentication and key exchange algorithm, the cipher and key length, the cipher mode of operation, the hash algorithm for integrity protection and the hash algorithm for use with pseudorandom functions.

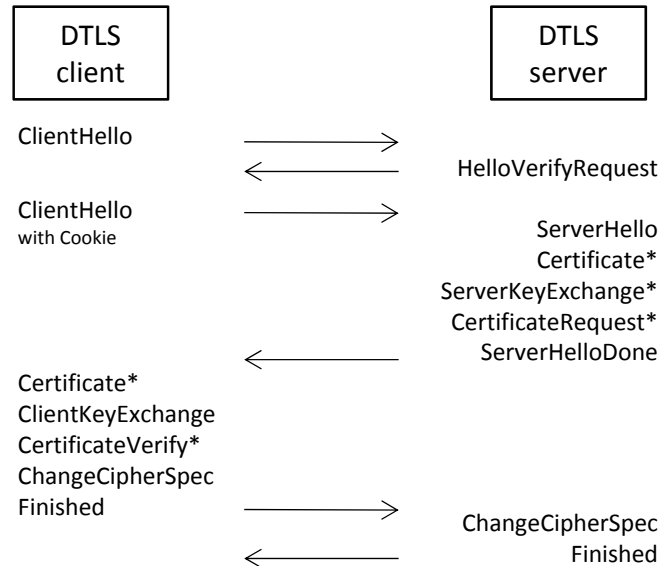


Figure 3.2: The full DTLS handshake.

The DTLS handshake is shown in Figure 3.2. In order to reduce the number of network packets, multiple DTLS messages can be grouped into a single flight of

messages. In the figure, the horizontal arrows correspond to the different message flights. The DTLS client initiates the handshake with the ClientHello message, to which the server replies with a HelloVerifyRequest message. The HelloVerifyRequest message contains the stateless cookie for DoS mitigation and must be echoed by the client in its second ClientHello message. After the server has verified the cookie, it responds with the ServerHello message. The hello messages are used to establish security enhancement capabilities between the client and server [13]. They establish the following attributes: protocol version, session ID (used in session resumption), cipher suite and compression method. Additionally, two random values are generated and exchanged: one for the client and one for the server.

The messages of the remainder of the handshake depend on the negotiated security enhancement capabilities. In the figure, messages marked with an asterisk (*) are optional or situation-dependent messages. The figure shows the message flow for a certificate-based cipher suite where the server replies with Certificate, ServerKeyExchange, CertificateRequest and ServerHelloDone messages. If the cipher suite requires the server to authenticate itself, then the server sends its X.509 certificate in a Certificate message. In cases where the key exchange does not use the server certificate, the server may send a ServerKeyExchange message. For example, in Pre-Shared Key cipher suites (PSK suites are discussed later), the server may send a hint in the ServerKeyExchange message to help the client in selecting which PSK identity to use. Additionally, the server may also send a CertificateRequest message to request a certificate from the client. Finally, a ServerHelloDone message is sent by the server to indicate that the hello-message phase of the handshake is complete.

If the server requested a certificate, the client must provide one in its Certificate message. Next, the client sends a ClientKeyExchange message, the contents of which depend on the chosen key exchange algorithm. In the case of RSA for example, the client chooses a secret and encrypts it with the public key from the certificate of the server and sends the result in the ClientKeyExchange message. Together with the Certificate and ServerKeyExchange messages of the server, the client's Certificate and ClientKeyExchange messages are used for the key exchange. The CertificateVerify message allows the client to prove the possession of the private key in the certificate. In the case of pre-shared key cipher suites, the key exchange of the client consists of a ClientKeyExchange message, which contains the identity of the chosen PSK.

Next, the client sends a ChangeCipherSpec message, which signals that the client has switched to the negotiated cipher spec. The client then immediately sends the Finished message, which contains a hash of the shared secret and all handshake messages. The server must verify the contents of the Finished message in order to detect any tampering of the handshake messages. The Finished message also proves that the client knows the correct shared secret (i.e., the pre-master secret), and any subsequent keying material (master secret, encryption keys and MAC keys) is generated from this pre-master secret. After the server has sent its own ChangeCipherSpec and Finished messages and the client has successfully verified the Finished message, the handshake is completed, and secure communi-

cation of application data can start.

3.2.3 DTLS in constrained environments

There are a number of additional protocol features that are applicable to DTLS in constrained environments, and these are discussed in this subsection. RFC 5116 [14] introduced Authenticated Encryption with Associated Data (AEAD) to TLS, which enables the use of cipher suites that use the same cipher for confidentiality, authenticity and integrity protection. Particularly in constrained environments, AEAD provides the benefit of more compact implementations as only one cipher has to be implemented.

RFC 6655 [15] defines multiple such compact cipher suites that use the widespread AES cipher in the Counter with Cipher Block Chaining-Message Authentication Code (CBC-MAC) Mode (CCM). AES is a popular choice in constrained environments, as it is often accelerated in hardware in modern IoT systems (e.g., the TI CC2538 SoC has an AES accelerator on the same die as the ARM-M3 CPU). Note that the AEAD construct is only supported from Version 1.2 of the DTLS protocol.

RFC 4279 [16] introduces Pre-Shared Key (PSK) cipher suites for TLS. These cipher suites are interesting for constrained devices, as the size of the key exchange is minimal: typically only a PSK identifier in the client key exchange is exchanged. Of course, key management is an important issue in this case, as common cryptography practice dictates that a unique PSK should be allocated for every peer. The ‘TLS_PSK_WITH_AES_128_CCM_8’ cipher suite combines the benefits of PSKs and AES-CCM in that only one cipher is needed (AES), and the key exchange is minimal. This cipher suite is also the mandatory-to-implement PSK cipher suite for DTLS in the CoAP RFC [9]. Furthermore, this suite uses just an eight-byte authentication tag (as opposed to a 16-byte tag), which is more suitable in networks where bandwidth is constrained and messages sizes may be small.

RFC 7250 [17] introduces a new certificate type and two TLS extensions for exchanging Raw Public Keys (RPKs) in DTLS. In this case, a peer has an asymmetric key pair, but it does not have an X.509 certificate; this asymmetric key pair is the RPK. This extension allows the raw public key to be used for authentication, which is beneficial in constrained environments as RPKs are smaller in size than X.509 certificates. Additionally the resulting key exchange is therefore smaller, as well. Of course, the scalability benefits of a Public Key Infrastructure (PKI) are lost when using RPKs.

Finally, RFC 7251 [18] describes the use of AES-CMM elliptic curve cryptography (ECC) cipher suites in DTLS. This type of cipher suites uses the AEAD mechanism to provide confidentiality, authenticity and integrity of application data with just AES, while using Ephemeral Elliptic Curve Diffie–Hellman (ECDHE) as their key exchange and peer authentication mechanisms. ECC is attractive for constrained environments as its smaller key sizes result in savings for power, memory, bandwidth and computational cost [19]. For example, a 256 to 383-bit ECC key is considered comparable in strength to a 3072-bit RSA key by NIST [20]. CoAP

mandates the use of the ‘TLS_ECDHE_ECDSA_WITH_AES_128_CCM.8’ cipher suite for X.509 certificates in constrained environments. This cipher suite uses the secp256r1 or NIST P-256 elliptic curve.

3.3 Problem statement and research goals

When securing communications in LLNs via end-to-end security with DTLS, one should be mindful of a number of potential issues and pitfalls. Some of these issues arise due to the limitations of the constrained devices that secure the communications. For example, in end-to-end security, there is a considerable difference between constrained devices (and their protocols) and powerful Internet hosts (and their protocols) in terms of available resources and design. A second potential issue stems from the DTLS protocol itself, namely the large overhead of the DTLS handshake can be an issue of concern in constrained networks. A third group of issues is related to securing the LLN itself and is the result of deploying end-to-end security in LLNs. Apart from these issues related to end-to-end security in LLNs, there is also the problem of the limited amount of application layer functionality that can be provided by constrained IoT devices. In a world as heterogeneous as the IoT there exists a need for protocol translation, data format mapping, semantic descriptions and many other features that improve the interoperability with IoT devices. Similarly, network access to constrained nodes and LLNs should be as efficient as possible by supporting caching of information, efficient discovery and network edge filtering. These types of functionality are too complex and in some cases impossible for implementation on a constrained device. Clearly, an approach that does not burden the constrained device is needed in this case. The remainder of this section discusses these various issues and problems in more detail.

3.3.1 End-to-end security in LLNs

Constrained devices with a limited power source (e.g., battery powered or energy scavenging devices) should take care to avoid excessive network communications in order not to preemptively deplete the power source. Similarly, constrained networks where the available throughput is in the order of a few kbps should minimize the amount of network communications to avoid congestion. Therefore, chatty or verbose security protocols that communicate excessive amounts of information should be avoided in these situations. As DTLS employs UDP instead of TCP as its transport protocol, it avoids the TCP handshake, which reduces the number of messages exchanged between DTLS clients and servers. However, some options supported by DTLS, as presented in the previous section, may lead to large amounts of network communications. Specifically, certificate-based cipher suites involve sending the certificate of the DTLS server (and peer, depending on the security needs) over the network. These certificates are generally large (i.e., a thousand bytes or more), and therefore, their network communication can be problematic when communication has a large impact on the power source or the net-

work. As a result, these types of devices are unable to offer authentication based on PKI certificates. While raw public keys are significantly more compact than X.509 certificates, they do not offer the same benefits in terms of authentication and scalability.

For devices with limited computational power (e.g., low-cost embedded systems) certain cryptographic primitives may prove too complex for computation by the low cost microcontroller. While hardware acceleration may help to alleviate this issue, it can be an expensive option and might only be available for certain primitives: e.g., AES is often accelerated in hardware, while others are not. Specifically, public-key cryptography methods (e.g., based on large integer factorization or discrete logarithm problems) and key agreement schemes (such as (EC)DH) may be too taxing for constrained microcontrollers. Therefore, the set of cryptographic functions that can be offered by such low cost embedded systems excludes a number of common cryptographic primitives and is typically limited to what can be achieved by symmetric-key cryptography.

Another important limitation in constrained environments is the low amount of available memory (i.e., both volatile and non-volatile memory). For example, according to IETF RFC 7228 [4], Class 1 constrained devices have around 10 kibibyte (KiB) of RAM and 100 KiB of ROM memory. Such a small amount of memory must accommodate an entire networking stack, adequate security mechanisms, peripheral control, the application itself and various other subsystems. This forces a device manufacturer to limit the amount of software that will ship with the device by carefully selecting what is needed. One consequence is that it is impossible for these devices to support a wide range of DTLS extensions and cipher suites (e.g., only one suite might be supported). This also means that verbose operations such as checking certificate revocation lists or performing OCSP [21] checks typically cannot be supported.

Powerful Internet hosts on the other hand may expect constrained devices to support security features similar to those found on the conventional Internet (e.g., with strong authentication and key agreement schemes). As constrained devices cannot support these features (see above), an alternative is to consider third party systems (e.g., middleboxes or off-path systems) that offer such features on behalf of constrained devices. However, in this case, a big issue with conventional end-to-end security is that as the connection is secured end-to-end, a third party is excluded from the communication. Thus, an important question addressed by this work is how third parties can take part in securing (but also optimizing; see later) communications with constrained devices in order to bridge the gap with powerful Internet hosts.

While DTLS can avoid the TCP handshake, it still has to perform its own handshaking mechanism in order to negotiate key exchange and authentication methods. The overhead of this handshake in terms of delay or amount of network traffic can be problematic for some types of constrained nodes and networks. Specifically, previous research has shown that in duty-cycled multi-hop networks, the delay introduced by the DTLS handshake can run up to fifty seconds [8] for four wireless hops. The authors also correctly conclude that the memory for stor-

ing the DTLS session state on constrained nodes is typically limited to a handful of nodes for Class 1 devices. Additionally, other research [22] has shown that ephemeral DTLS sessions with constrained devices should be avoided as their energy expenditure is up to 60% higher when compared to a single DTLS session with a long lifetime. Therefore, one goal of this work is to limit the impact of the DTLS handshake on delay and energy expenditure, while supporting more than just a handful of simultaneous DTLS sessions per constrained device.

The third group of issues stems from naively deploying end-to-end security in (multi-hop) Low-power and Lossy Networks (LLNs) and from allowing unmonitored access to LLNs to malicious users. In these networks, resources are sparse (see above), and care should be taken in order to avoid unwanted depletion of these resources by Denial-of-Service (DoS) attacks. For example, by repeatedly opening and closing DTLS sessions, a malicious user can significantly reduce the lifetime of a battery-powered device. A malicious user could also send large datagrams to the LLN, which will trigger fragmentation that can exhaust the allocated network buffers in the LLNs. Most of these resource-depletion threats can be mitigated by monitoring and restricting access to the LLN at the edge of the network, where an unconstrained firewall or gateway system resides. However, end-to-end security encumbers such systems from authenticating parties (as constrained devices cannot support strong authentication) and therefore restricting access to authorized parties. Here, this work will study how end-to-end security can be reconciled with the need for traffic filtering at the edge of the network and the need for strong authentication.

3.3.2 Complex application features in LLNs

Apart from security issues, there is another important category of problems that relate to the functionality at the application layer for constrained devices, which is targeted by this work. Firstly, the same constraints that prohibit offering extensive security features also apply to implementing application features on the constrained device. This is one of the reasons why the IETF has standardized special purpose protocols and data formats for use in constrained environments (e.g., CoAP and CoRE Link Format (CLF) [23]). However, traditional Internet hosts do not always implement these protocols and data formats. In these cases, a protocol and data format translation should occur that enables the Internet host to communicate with the constrained device (e.g., an HTTP/CoAP proxy and a JSON/CLF mapper). Such a translation has to be performed by an unconstrained third party system (e.g., gateway). Secondly, some types of functionality can be ineffective when they are offered on the constrained device. An example is caching the responses of a constrained server on the device itself, which will not save any network traffic. A second example is the aggregation of observation relationships by intermediaries; clearly, this has to be offered on an intermediary and not on a constrained node in order to have any effect. Note that conventional end-to-end security does not allow for response caching or observation aggregation, as all traffic passing at an intermediary is encrypted. Thirdly, some functionality can be

inefficient when they are implemented on the constrained device. An example is storing verbose semantic descriptions on a constrained device, which will lead to significant amounts of network traffic every time these descriptions are requested. Another example of functionality that is inefficient to offer on constrained devices is access control. Typically, the LLN will have already spent a significant amount of resources delivering the request to its destination where it will end up being discarded. Clearly, discarding this request before the network has wasted its resources is more efficient. For these cases, this work will study how third party systems can support and optimize the operations of constrained devices and LLNs.

3.3.3 Problem statement: illustration in a smart building use case

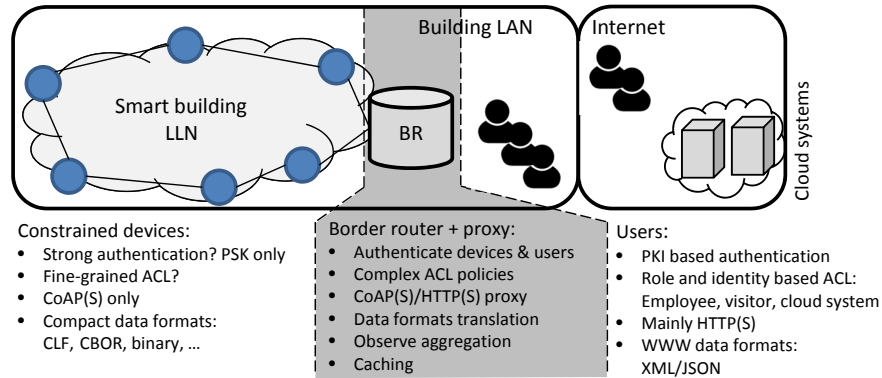


Figure 3.3: In a smart building scenario, there is a wide variety of different users. Constrained devices are unable to offer all necessary security and application features to cater to these users. In the approach followed by this work, unconstrained systems (e.g., border routers (BRs)) assist by offering these missing features. CBOR: Concise Binary Object Representation, ACL: Access Control List.

Figure 3.3 shows a smart building scenario that illustrates the problems targeted by this work. In a smart building most of the building services can be monitored and controlled over the Internet. Such services include for example the management of doors, lighting, climate control (e.g., AC), elevators and the monitoring of presence in certain areas. Smart buildings, such as offices and public buildings, typically have a large variety of users: visitors, cleaning staff, technicians, employees, etc. Similarly, there are also a number of computer systems that interact with the smart building: e.g., systems for HVAC, surveillance, facility management, etc. Each of these actors accesses the services offered by the building according to specific access control rules that depend on the role and or identify of the actor, e.g., the HVAC system can control the air conditioning units, but cannot control the doors. However, the HVAC system might be allowed to monitor the status of a door adjacent to an AC unit without being able to (un)lock it. Considering

the limited resources of constrained devices (see above), managing and enforcing which actions an actor is allowed to perform depending on their role or identity quickly become too complex for the constrained devices. Furthermore, as most constrained devices only support PSK-based authentication, such a system would require management of shared secret keys between every two actors. Limitations on the LLN and the constrained devices also prohibit these devices from offering protocols and data formats that are common to the unconstrained actors, such as HTTP(S) and XML/JSON. The gray center of the figure already hints at our approach detailed in the next section: a proxy offers many of the missing features on behalf of the constrained devices.

Finally, one might question why this work relies on end-to-end security via DTLS at all, when there appear to be many problems in constrained environments according to the discussion above. Our main motivations for doing so is that DTLS is a proven (and secure) standard, is widely available, is commonly used on the Web and is standardized for use with CoAP. Alternatives to DTLS are either proprietary, or still in the process of standardization (e.g., Object Security of CoAP (OSCOAP) [24]), not applicable to constrained environments (e.g., network layer security), or cannot provide the same level of security as DTLS (e.g., physical layer security). Object security specifically can be considered complementary to transport layer security, and while it is not considered in this work, it can be combined with the work presented here (if feasible given the constrained environments under consideration). The Related Work section discusses object security in greater detail. While the literature shows that lightweight network security is feasible in constrained environments (e.g., compressed Internet Protocol Security (IPsec) [25]), it is not considered in this work because CoAP standardized end-to-end security over DTLS as its security mechanism.

3.4 The Secure Service Proxy

The approach followed in this work allocates one reverse CoAP(s) proxy per constrained device. The CoAP specification [9] defines a reverse proxy as “an endpoint that stands in for one or more other server(s) and satisfies requests on behalf of these, doing any necessary translations”, and it also states that “The client may not be aware that it is communicating with a reverse-proxy; a reverse-proxy receives requests as if it were the origin server for the target resource.” The reverse proxy approach enables splitting the end-to-end communication between a constrained device and its client at the proxy with no need for any additional configuration on the client (as mentioned in the CoAP specification). While the resulting communication is no longer end-to-end, indeed the proxy will share DTLS security contexts with both parties and will translate CoAP messages, the resulting system has many benefits and is able to overcome all of the issues that are discussed in the previous section. Additionally, our reverse proxy approach implements a virtual device for every constrained device. This enables the reverse proxy to extend a constrained device (beyond only proxying) by hosting functionality on the cor-

responding virtual device. Finally, by enabling the reverse proxy to be deployed on any system (see design), it is not restricted by the limitations common to constrained IoT devices. In the next subsections, we argue that the benefits of this approach far outweigh the downsides of splitting the end-to-end communication, and we present our design for such a reverse proxy.

3.4.1 Motivation of approach

Our motivation for following a reverse proxy approach consists of two facets: one for the security-related aspects of constrained devices and LLNs and one for the application layer-related aspects of constrained devices. In terms of security, the reverse proxy approach allows one to setup two sorts of DTLS sessions: “lightweight” sessions between the constrained devices and their reverse proxy and fully-featured sessions between the proxy and the clients of the devices. The lightweight sessions employ security primitives that are known to the constrained devices (e.g., pre-shared keys for authentication and key exchange), while the fully-featured sessions can use conventional security methods that are known to the clients: e.g., certificates for strong authentication and Elliptic Curve Diffie-Hellman (ECDH) for the key exchange (including ephemeral key exchanges if perfect forward secrecy is required). Additionally, the reverse proxy can be configured to maintain one long-term session with the constrained device while simultaneously keeping active sessions with multiple clients. This allows one to overcome the small session pool at the constrained devices (due to its limited memory, see above), as well as limit the total number of handshakes performed by the constrained device during its lifetime. As a result, the impact of the DTLS handshake on the LLN and the communication in terms of, e.g., traffic and communication latency is lowered. Finally, the reverse proxy also protects the LLN from a number of resource depletion attacks from attackers on the Internet. By design, a reverse proxy handles all messages for all constrained devices in an LLN from Internet hosts. Thus, the reverse proxy becomes the main traffic entry point for the LLN, and therefore, it can inspect, filter and drop traffic in order to root out traffic from malicious users. Combined with the strong authentication of clients and an access control policy, this proxy can make more informed decisions in regards to filtering traffic when compared to, e.g., a simple Internet firewall.

In terms of the application layer, a reverse proxy is free to process and transform the requests it receives from clients as it chooses. A reverse proxy can improve network access by offering features such as caching, network-edge access control and enforcing congestion control algorithms. Interoperability with other systems can be increased by, e.g., translating between HTTP and CoAP, which is fairly straightforward considering the design goals of CoAP. Translation between different data types (e.g., CoRE link format [23] to JSON) can also boost interoperability. Such a proxy can also implement additional application functionality on behalf of the constrained device. Examples of such functionality include extending the constrained device with semantic descriptions for its resources, a deployment location photo, the weather near the device, etc. Additionally, a proxy can choose

to facilitate adding, configuring and deploying such functionality via a plugin-like system. This greatly eases the management of such functionality at run time by making adding, updating, enabling and disabling such functionality easier.

It is important to reiterate that all of the above is possible without any additional configuration on either the constrained device or the client; nor does the presented approach require any modifications to the standards compliant protocol stacks (e.g., 6LoWPAN/DTLS/CoAP) running on the constrained device and the client. Indeed, the client discovers the Internet endpoint of the constrained device that is hosted on the proxy, and the proxy takes care of mapping every request to the corresponding constrained device. In the scenario presented here, all configuration is limited to the proxy. These last two benefits are an important differentiator from existing work, as will be discussed in the Related Work section.

While the reverse proxy approach offers a number of benefits, it also entails some risks that if ignored might undermine the presented system. One risk is that the reverse proxy presents a single point of failure in terms of security and operation. Indeed, if the reverse proxy were to be compromised then, e.g., all session keys and long-term keying material (pre-shared keys and private keys) could be made public. As the proxy offers a RESTful interface for managing virtual hosts and their keying material, this interface entails a security risk and should therefore be properly hardened against malicious usage (see Section 3.4.3.1 for suggestions). Likewise, if the reverse proxy were to be the target of a resource depletion attack, then the constrained devices hosted by that proxy would become unreachable. On the other hand, as the proxy is deployed on a more powerful system, the proxy is more resilient to resource depletion attacks than constrained devices and networks. A second issue is the introduction of a third party (i.e., the proxy itself) into the trust model by terminating the end-to-end security that must be trusted by both the constrained device, as well as the clients. As all collected data and issued commands pass via the proxy, this can raise privacy concerns when the device or the client does not trust the owner of the proxy. One option to mitigate this privacy risk is to let the owner of the constrained devices operate the reverse proxy on his or her own. To this end, our evaluation shows that a low-cost single board computer (e.g., Raspberry Pi) is capable of hosting the proxy, which enables on-premises deployments. To summarize, the proxy breaks end-to-end security in order to provide additional features, which address operational and performance concerns of resource constrained devices. This work argues that the benefits of terminating the end-to-end security outweigh the security-related risks in the case of ‘Class 1’ resource constrained devices and networks. For less constrained devices and networks, this balance might tip in favor of end-to-end security.

3.4.2 Secure Service Proxy: design

In order to enable our proxy to extend constrained devices with a wide range of functionality, the design adopts the concept of virtual devices. In our design, every virtual device is allocated a dedicated IPv6 address from an IPv6 subnet that is either routed to the proxy or directly connected to the proxy. Every virtual device

endpoints; this is shown in the bottom left of Figure 3.4.

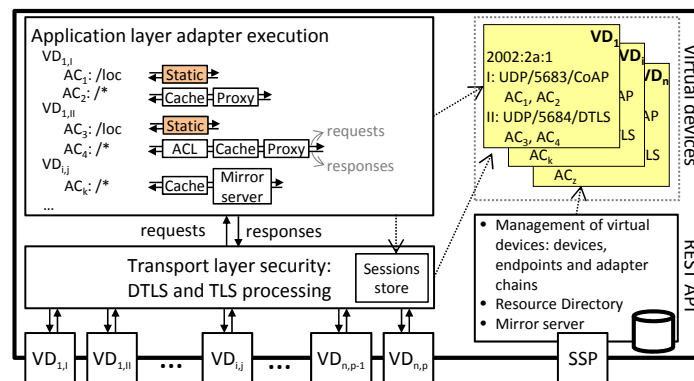


Figure 3.4: Secure Service Proxy: design.

depend on the endpoint involved.

endpoint of the virtual device, as well as the URI of the request.

the chain. Every element of the chain (i.e., an adapter) can either return (a modi-

fied) the request, which will be passed to the next adapter in the chain, or stop the execution of the chain by returning a response. The current implementation allows returning a response from an adapter in a non-blocking (i.e., asynchronous) way, as retrieving a response might involve a lengthy IO operation. Once the response is available, it is passed along the chain in reverse. This allows adapters to process and (if needed) modify the response before it is stored in the virtual device and returned to the client.

Application layer adapters implement the functionality hosted by virtual devices. The idea underlying adapters is to compartmentalize functionality into modules that can be reused by virtual devices. When creating an adapter chain, an instance for every adapter in the chain is created, and every instance is configured according to the parameters exposed by the adapter type (see further). While instances of adapters reside in adapter chains, they can be shared by more than one adapter chain (AC). For example, in Figure 3.4, the same Static adapter instance (colored orange) is shared by AC₁ and AC₃. This is mainly useful when the same functionality should be available for multiple endpoints of the same virtual device (e.g., CoAP and CoAPs) or when an adapter implements functionality that does not require configuration that differs per adapter chain (e.g., a logging adapter that logs all incoming requests for auditing purposes).

The proxy also exposes a networked interface in the form of a REST API to manage virtual devices, which is shown in the bottom right of Figure 3.4. The REST API allows creating and deleting virtual devices and their endpoints, as well as instancing and deleting adapters and defining adapter chains. When creating (D)TLS endpoints, the REST API also allows specifying the cipher suites supported by the virtual device, as well as the keying material (e.g., X.509 certificate or private key). Apart from the management interface, the proxy also hosts a resource directory that contains the hosted virtual devices. Finally, a mirror server is also available to enable resource updates from constrained devices that are asleep for continuous and long periods of time (i.e., sleepy devices). This mirror server can be used by virtual devices to interface with resources from sleepy constrained devices.

Finally, the presented design allows one to deploy the proxy on different locations in the network by varying the IPv6 subnet for the allocation of virtual device IPv6 addresses. We foresee two scenarios. In the first scenario, the proxy resides close to the constrained devices by allocating addresses from a neighboring LAN network to virtual devices. An example would be a home LAN network from which the proxy assigns unused addresses to virtual devices. In the case of a 6LoWPAN network, the proxy can be combined with the border router. This scenario also aligns nicely with the distributed computing concept that is commonly found in fog computing and in in-network processing [26]. In a second scenario, the proxy resides further ‘upstream’ from the constrained devices (e.g., in a data center, the cloud, etc.) and allocates addresses from a special-purpose IPv6 subnet that is dedicated to virtual devices. In this scenario, the routing has to be configured to route this special-purpose IPv6 subnet via the proxy (which is not a problem in most data centers). Both scenarios are complementary and will depend

on the specific needs of the considered use-case: e.g., a proxy in the LAN network means that data stay inside the home network, which may benefit privacy. Similar considerations were previously discussed in the problem statement section.

3.4.3 Secure Service Proxy: implementation

For the implementation of our secure service proxy, we chose to build upon the previous work in our CoAP++ framework (which in turn builds on top of the Click modular router software). This choice provides a great amount of flexibility in how we process the network traffic for the virtual (and constrained) devices, as all routing functions are part of Click and can therefore be configured to our liking. In terms of the (D)TLS implementation, we chose to use the wolfSSL library as this offers the easiest API for managing sessions and integrating into the Click router where most processing happens on network packets.

3.4.3.1 Virtual devices and endpoints

Virtual device endpoints are created and deleted via the management interface. This is a straightforward REST interface that is hosted on the secure service proxy over CoAPs. As this interface handles sensitive information such as keying material, access is restricted to authorized users, which are allowed to manage endpoints and adapter chains.

POST requests with an endpoint description are used to create a new endpoint for a virtual device. The endpoint description contains both the virtual device to which the endpoint belongs, as well as any configuration details describing the endpoint itself. This description is serialized as a JSON object in the payload of the POST request. For a plain-text CoAP endpoint, the configuration details are limited to the UDP transport port of the endpoint. For a DTLS CoAPs endpoint, the configuration also includes information about the supported cipher suites and any parameters for the cipher suites. In the current implementation, the “TLS_PSK_WITH_AES_128_CCM_8” and “TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8” cipher suites are supported for CoAPs endpoints. When creating an endpoint that supports the PSK cipher suite, the pre-shared-key and an (optional) client identity hint have to be specified as parameters. For the elliptic curve DSA suite, the secp256r1 private key and signed certificate have to be provided as parameters. These are both encoded in base64 in the endpoint description. The following listing contains an example POST request that creates a CoAP endpoint for a virtual device hosted under 2001:6a8:1d80:23::1 on port 5684 with an ECC cipher suite.

```
POST /virtualDevices
Content-Format: application/json
{
  "address": "2001:6a8:1d80:23::1",
  "prefixLen": 128,
  "port": 5684,
  "dtls": {
    "supportedCipherSuites": [
```

```

    {
      "cipherSuite": "TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8",
      "parameters": {
        "b64PrivateKey": "QVNO...==",
        "b64Certificate": "LS0t...=="
      }
    }
  ]
}

```

```
2.01 Created /virtualDevices/2001:6a8:1d80:23::1~128~5684
```

The response of the secure service proxy links to a newly-created resource that can be used to delete the endpoint at a later time. This resource is also used for managing the adapter chains that belong to an endpoint, as explained in section 3.4.3.3.

3.4.3.2 Implemented application layer adapters

In terms of application layer adapters, our proxy currently implements the adapters listed in table 3.1. This section describes each of the adapter types in more detail.

The access control adapter applies Access Control List (ACL) rules to the CoAP(s) requests it processes. ACL rules are parsed as JSON objects that assign allow and deny rules to either a username or a role of users. An allow and deny rule consists of a regular expression, which is applied to the request URI, and a list of request methods. In case no matching ACL rule is found, then the default policy of the adapter instance (either accept or deny) is applied. The following JSON serialization of an example ACL rule gives user “bob” full access to the devicename resource, while access to the lock resource is restricted to read only.

```

{ "username": "bob",
  "allow": [{ "uri-regex": "devicename", "methods": ["GET", "PUT", "POST", "DELETE"] },
            { "uri-regex": "lock", "methods": ["GET"] } ],
  "deny": [] }

```

Hosting a virtual resource on a virtual device is the task of the static resource adapter. In order to allow arbitrary content types of the payload, the value of the virtual resource is encoded in base64 in the configuration of the adapter. An example is shown in the next section.

The cache adapter serves and caches responses for requests to virtual devices. The cache adapter calculates a cache key for every CoAP request it handles. When a fresh response matching the cache-key is found, the adapter chain’s execution is halted, and the cached response traverses the adapter chain in reverse. Responses processed by the cache adapter are handled in accordance with Section 5.9 of the CoAP RFC [9]. This means that, e.g., a “2.05 Content” response will be cached, while a “2.04 Changed” response will mark any stored response as not fresh. Cached responses are removed when they expire after their Max-Age option. Note that the cache adapter does not implement the “Validation Model” specified in Section 5.6.2 of the CoAP RFC [9]. When used in conjunction with access control, it is important that all ACL rules are applied before hitting the

Table 3.1: The proxy offers a number of functionalities, called adapters, that are hosted on virtual devices. The list of adapters that were implemented at the time of this work are shown in this table.

Adapter	Functionality	Configuration parameters
Access control	Restrict access to virtual devices depending on client identify, request method and URI.	ACL rules and default policy
Static resource	Host RESTful resources on virtual devices that can be read and modified.	Payload and content type
Cache	Cache and serve previous responses from virtual devices to clients.	Default cache entry lifetime
Congestion control	Enforce congestion control on clients querying virtual devices. Per device and network wide rules are implemented.	Per user CC limits
.well-known/core	Manipulate discovery responses from virtual devices to include functionality hosted by the proxy.	None
Proxy	Proxies requests for the virtual device to a CoAP(s) server (e.g. the constrained device). Also aggregates observe registrations.	CoAP(s) server endpoint
Mirror server	Proxies requests for a virtual device to a mirror server.	Mirror server endpoint and sleepy device anchor point

cache, as the execution of the request leg of the adapter chain will stop when a cache hit is found. The underlying implementation caches responses in memory via a memcached instance.

The congestion control adapter in its current form applies traffic shaping on a per host basis. Currently, it is possible to limit the number of open requests between a client and a specific virtual device and between a client and a group of virtual devices. This group encompasses all virtual devices with an adapter chain that shares the same congestion control adapter instance. Open requests are requests for which a response has not been sent yet. If a client reaches its limit, then the request is dropped until either a response is received or one of the prior requests of that client is removed after a time out period (can be configured). Finally, a client can either be identified by its endpoint address or by its identity derived from the authentication credentials during the (D)TLS handshake.

The .well-known/core (wkc) adapter is responsible for including the functionality that is hosted on the virtual devices in the resource discovery responses of the real constrained device. In the current implementation, the wkc adapter asks ev-

every adapter from all of the adapter chains that are defined for the virtual device to modify the discovery response from the real device. This way, the static resource adapter can add a link to its virtual resource, and the ACL adapter can remove links for resources that the user is not authorized to access. To this end, every adapter type offers a “processDiscoveryResponse” method that is used by the wkc adapter.

The proxy adapter takes a request for a virtual device and issues a new CoAP request to the corresponding actual constrained device. Therefore, an instance of this adapter is configured with the CoAP(s) endpoint of the constrained device. Only the transport layer addresses are changed; the new CoAP request is copied from the output of the previous adapter in the adapter chain (with the exception of the message ID and the token, of course). The proxy adapter will either retrieve a response or generate a time-out; therefore, it always comes last in adapter chains. This adapter will also combine observation registrations when it receives multiple registrations for the same resource on a virtual device. Likewise, it also multiplexes responses from constrained devices to multiple clients in case there is more than one ongoing observation registration.

Finally, the mirror server adapter is a special type of proxy adapter in that it issues CoAP(s) requests to a mirror server instead of the constrained device itself. Apart from the end point of the mirror server, also the handler of the constrained device is configured into the mirror server adapter instance. For instance, a request to the coaps://vd1.iot.test/status resource on a virtual device would be translated to coaps://ms.iot.test/ms/0/status.

3.4.3.3 Adapter chain management: interface

Once an endpoint for a virtual host has been allocated on the proxy, adapter chains can be created and hosted on that endpoint. Building on our previous example, the listing below contains a CoAP request that instantiates an adapter chain, which contains the access control, well-known core rewriting, caching and forward CoAPs proxy adapters. Again, the payload is a JSON object that describes the chain and contains the parameters for the different adapter instances. The adapter chain is created as the default chain via the wildcard character in the chain path. The default chain is executed for requests where no other adapter chains with a matching URI path are found.

```
POST /virtualDevices/2001:6a8:1d80:23::1~128~5684
Content-Format: application/json
{
  "path": "/*",
  "pipeline": [
    {
      "type": "acl",
      "default_access_control_policy": "deny",
      "rules": [
        {
          "username": "fvdabeele",
          "rules": [
            {
              "uri-regex": "regex1",
              "allowMethods": ["*"]
            },
            {
              "uri-regex": "regex2",
              "allowMethods": ["GET"]
            }
          ]
        },
        {
          "username": "*",
          "rules": [
            {
              "uri-regex": "regex1",
              "denyMethods": ["*"]
            },
            {
              "uri-regex": "regex2",
              "allowMethods": ["GET"]
            }
          ]
        }
      ]
    }
  ]
}
```

```

    ]
  },
  {
    "type": "wkc"
  },
  {
    "type": "cache",
    "default_lifetime": 60
  },
  {
    "type": "proxy",
    "scheme": "coaps",
    "addr": "bbbb:1",
    "port": 5684
  }
]
}

2.01 Created /virtualDevices/2001:6a8:1d80:23::1~128~5684/*

```

The second example, shown in the listing below, details how a static resource is created on the endpoint of our virtual host (in this case, it contains a semantic description of the virtual host in the RDF format). The chain also illustrates the linked adapter, which refers to the ACL adapter instance that was created in the previous listing. The link points to the management resource of the adapter instance.

```

POST /virtualDevices/2001:6a8:1d80:23::1~128~5684
Content-Format: application/json
{
  "path": "/rdf",
  "pipeline": [
    {
      "type": "linkedAdapter",
      "link": "/virtualDevices/2001:6a8:1d80:23::1~128~5684/*/*0"
    },
    {
      "type": "static",
      "contentType": 41,
      "value": "PGh0d...=="
    }
  ]
}

2.01 Created /virtualDevices/2001:6a8:1d80:23::1~128~5684/rdf

```

Finally, note that the parameters of existing adapters can be updated via a PUT request to the management resource of the adapter instance. In this case, the payload is a JSON object where the keys are the parameter names. Likewise, adapters and chains can be deleted via their respective management resources.

3.4.3.4 Authenticating (D)TLS clients on the SSP

In order to facilitate the authentication of users and the authorization of user actions, the SSP links client authentication information (e.g., TLS PSK or X.509 client certificate) with users and roles. The current implementation is limited to using TLS primitives for supplying authentication credentials, although in the future, alternatives might be considered (e.g., lightweight application-layer access

tokens). For example, a (D)TLS session that was setup with PSK_1 as the pre-shared key can be linked with $user_A$. Likewise, attributes in a client X.509 certificate that is signed by a party trusted by the SSP can be linked with a specific user, e.g., a certificate issued by CA_A with the common name attribute set to *fydabeele* can be linked with $user_B$. Finally, the proxy also exposes a RESTful interface for managing which credentials belong to which user and the roles of users.

3.4.3.5 Key management between SSP and constrained devices

The SSP contains an in-memory repository of pre-shared keys and corresponding identity hints in order to setup DTLS sessions with resource-constrained CoAPs servers. As this repository contains all of the keying material for the constrained devices known to the proxy, it contains sensitive information and should be handled accordingly. In the current implementation, this repository is initialized when the SSP process is started. A future extension could enable at run-time manipulation of this repository by, for example, specifying keying material when instantiating coaps proxy adapters. Currently, this has not yet been implemented, as in our use cases, this repository does not change frequently and remains stable. In use cases where the repository is more volatile, such an extension could enable better key management.

3.5 Related work

The concept of device virtualization in the IoT is widespread in the literature, though often times under different names such as sensor, thing and object virtualization. Indeed, in [27], the authors present a survey on object virtualization in the IoT stating that “the concept has become a major component of current IoT platforms where it aids in improving object energy management efficiency and addressing heterogeneity and scalability issues”. The authors classify existing architectures as one or many real objects for one or many virtual objects. While the focus in this work has been on one real object for one virtual object, the flexibility of the presented design enables the same adapter to be shared by multiple virtual devices, as well as one virtual device to span multiple physical devices (for example, a virtual device combining all lamps in a room).

There exist numerous works in the literature that study the benefits of using third parties or intermediaries in constrained environments. In order to narrow the scope of this section, only works that are relevant in the context of constrained RESTful environments are discussed here. In [28], Kovatsch et al. discuss moving application logic from firmware to the cloud. According to the vision of the authors, devices are thin servers exposing RESTful resources for data access and actuation, and most of the application logic would reside in the application servers. While our approach also advocates thin servers for devices, deploying the SSP in the cloud is optional. In use-cases where local access is important, the SSP may reside closer to the devices (e.g., deployed in the LAN) in order to meet require-

ments with respect to latency, privacy or availability. Additionally, the SSP may support constrained nodes and applications servers by providing functionality such as caching and more scalable authentication and authorization. The IPv6 addressing proxy presented in [29] is an example of an intermediary system for mapping legacy technologies to the IPv6 Internet of Things. By allocating IPv6 addresses to map to different legacy technologies, the approach is similar to the virtual devices presented in our work. Note that the adapter concept provides the flexibility to map virtual devices to different technologies similar to the work in [29]. While not presented in this work, the SSP has been used to host LoRaWAN end devices as virtual IPv6 CoAP endpoints via an Advanced Message Queuing Protocol (AMQP) publish/subscribe adapter that interfaced with the LoRaWAN network server. The authors in [30] propose to interconnect Web applications based on HTTP and Web sockets with CoAP-based wireless sensor networks via a CoAP proxy. The CoAP proxy focuses on translating between different protocols and closely follows the guidelines outlined in RFC 8075 [31]. The scope of the SSP is broader, as it includes transport security, access and congestion control next to mapping HTTP to CoAP. Finally, note that the forward proxy approach of Ludovici differs from the reverse proxy approach of the SSP. In [32], Mongozzi et al. introduce a framework for CoAP proxy virtualization in order to address the scalability and heterogeneity challenges faced in large-scale Web of Things deployments. The framework installs a reverse CoAP proxy on the sensor network gateway and then applies virtualization so that the proxy can be customized and extended by third parties without modifying the reverse proxy. All interactions of these virtual proxies with smart objects pass via this reverse proxy, which acts as an arbiter for access to the limited resources of the smart objects. The presented approach is interesting as the containerization of the virtual proxies into virtual machines makes them more flexible than the adapter approach followed in the SSP. We have experimented with providing some degree of extensibility by creating adapters from python scripts in the SSP (these scripts could be uploaded via the adapter chain management interface). While this python adapter type provided some degree of customization, the lack of proper process isolation meant that (malicious) scripts could stall the SSP. As such, these python adapters did not make the final SSP design. While the concept of the virtual proxies is interesting, the extent of the work is limited as the focus lies on the virtualization technique, and interesting features such as scalable security and efficient and authorized network access are not considered. Instead, the authors focus on providing service differentiation between multiple virtual proxies. Also note that proxy virtualization is not the same concept as device virtualization, though they can be used to solve similar problems. The same authors of [32] look at the specific problem of proxying CoAP observation efficiently for different QoS requirements in [33]. While the scope of the work is quite different from this paper, the use of a reverse proxy for bundling observation relationships is shared between the two works. Another example of device virtualization in RESTful environment is [34], where the authors assign virtual coap servers to RFID tags. The actual CoAP servers are not running on the tags though. Instead, they reside on RFID readers, which are able to enhance tags with

additional functionality (such as discovery). This work has parallels with the SSP, which enhances constrained devices by means of application layer adapters.

A second category of relevant works in the literature studies the challenges faced by transport layer security in constrained IoT environments. There are a number of works that study the DTLS handshake, as it is a fairly complex and verbose process with significant resources requirements for constrained devices. In [35], Hummen et al. propose a delegation architecture that offloads the expensive DTLS connection establishment to a delegation server, thereby reducing the resource requirements of constrained devices. The delegation architecture also enables more complex authorization schemes, as it has more resources at its disposal. The authors report significant reductions on memory overhead, computations and network transmissions on constrained devices. Our termination method can also provide complex authorization schemes of the virtual device. In Section 3.6.1, we have also reported significant savings in regards to CPU and network resource usage (and consequently, energy usage). While our approach still requires an active DTLS session between the SSP and the constrained device, the number of handshakes during the lifetime of a device is drastically reduced. While the memory requirements are not as low as in [35], they are still lowered as the constrained device can limit the number of simultaneous sessions to one. Finally, note that our approach does not require any changes to the DTLS stack running on the device. The work in [36] focuses on various challenges in deploying DTLS in resource-constrained environments. Similarly to [35], the approach revolves around handshake delegation. The authors adopt the concept of secure virtual things in the cloud where physical things delegate the session initiation to their corresponding virtual thing. As a result, physical things can limit their DTLS implementation to only the record layer protocol, which leads to drastic memory savings. One interesting aspect of the presented architecture is that the physical thing can assume both roles of client and server. Unfortunately, the concept of virtual things is not extended beyond the handshake delegation mechanism. It would be interesting to combine a delegation mechanism with some of the findings presented in our work. A hybrid option would be possible where the delegation mechanism is used for the most constrained devices (requiring a custom lightweight DTLS stack) and where the termination mechanism can be used for devices with sufficient memory (i.e., where a full DTLS stack is feasible) or where the DTLS stack cannot be customized to implement the delegation method.

Object Security of CoAP (OSCOAP) [24] is an IETF Internet Draft standardizing end-to-end security of CoAP options and payload at the application layer. While the specification focuses on the forwarding case when using a forward proxy (which excludes caching), it does include an appendix describing a mode of operation, Object Security of Content (OSCON), which is compatible with caching responses at intermediaries. The draft notes that OSCOAP may be used in extremely-constrained settings, where CoAP over DTLS may be prohibitive, e.g., due to large code size. Nevertheless, the authors state that OSCOAP may be combined with DTLS, thereby benefiting from the additional protection of the CoAP message layer present in DTLS-based security. Note that the standardization ef-

forts focus on the case of a forward proxy, whereas this work focuses on a reverse proxy approach. As such, the trust models are different as the reverse proxy represents the end device from the point of view of the client. Despite the difference in proxy models, the two approaches remain compatible and could strengthen each other. For example, the SSP could implement OSCOAP for cases where clients are employing a forward proxy, which is not trusted by the client. Additionally, it would be interesting for the SSP to support OSCOAP as a lightweight alternative for DTLS to protect communications with constrained devices with severe memory limitations. In such a case, clients would communicate securely with the SSP over DTLS while the communications between the SSP and the constrained devices would be protected either via OSCOAP (e.g., for constrained devices with severely limited memory) or via DTLS (e.g., for constrained devices with sufficient memory).

Finally, in high volume Web environments, transport layer security is often terminated at a proxy deployed close to the Web server(s). The main motivation for terminating TLS is that it enables load balancing, where terminated HTTPS requests are distributed over multiple Web servers. Load balancing increases the availability of the Web deployment, as the outage of one Web server does not affect the service availability in this case. Popular Web proxy software, like nginx and HAProxy, supports different reverse proxy deployment options for terminating TLS. Similarly, the elastic cloud computing platform of Amazon.com, Amazon Web Services, supports TLS termination and load balancing by virtue of its HTTPS listener service. While the main motivation of the SSP for session termination is not load balancing, the SSP does apply termination in order to be able to move computationally-expensive and verbose operations from constrained devices to the proxy, which improves performance. Similarly to high availability TLS proxies, the SSP may reduce key management complexity, as all keying material for public communications is stored on one system.

3.6 Evaluation: results and discussion

This section presents two evaluation scenarios that show the gains attainable by our approach. Such gains include: a decrease in load on constrained devices and the LLN, lower energy usage for constrained devices, an increase in user handling capacity of LLNs, more responsive LLNs, more scalable authentication and better authorization. The evaluation scenarios were chosen to evaluate the impact of the proxy on two specific operational aspects of LLNs: setting up DTLS sessions with constrained devices over multiple wireless hops and observing CoAPs resources on constrained devices from multiple DTLS clients.

3.6.1 Terminating end-to-end-security at the SSP

The first evaluation scenario is geared towards quantizing the impact of splitting end-to-end security at the smart service proxy. More specifically, the goal is to

study the impact of re-using a DTLS session of a constrained CoAPs server on the operation of both the constrained node, as well as the CoAPs client.

3.6.1.1 Simulation setup

Extensive simulations were performed with a nine-node 6LoWPAN network arranged in a cross topology as detailed in Figure 3.5. One node is at the center of the cross and is the RPL border router of the 6LoWPAN network; four nodes are intermediate routers (each located in the middle of one of the four legs of the cross); and the last four nodes are CoAP(s) servers that are located at the four ends of the cross. The border router is connected to the smart service proxy, which is running on the same PC as the Cooja simulator. Finally, an unconstrained CoAP(s) client interacts with the CoAP(s) servers. In the evaluation scenario, the client sends the following sequence of CoAP(s) requests: a .well-known/core discovery request, a sensor measurement request for the “/s” resource and an actuator request for the “/a” resource. The constrained CoAPs servers are running er-coap and Tiny-DTLS (in Contiki) configured to accept the “TLS_PSK_WITH_AES_128_CCM_8” cipher suite with a PSK hint of 15 bytes.

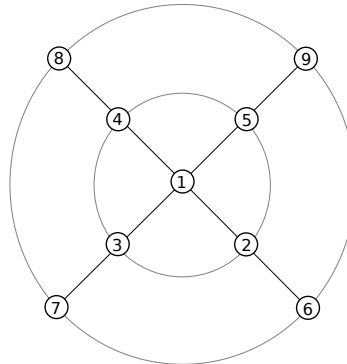


Figure 3.5: Cooja network topology: four CoAP(s) servers (6, 7, 8, 9) are located two hops away from the RPL border router.

The same request sequence was sent to the CoAP(s) servers for one reference case and three different SSP configurations: Plain Text (PLT), End-to-End (E2E), first Termination (TER1) and n -th Termination (TER). In the PLT configuration, all requests are sent over plain text CoAP. This is a reference cases for the other three cases. In the E2E case, all requests are sent over CoAPs without any termination of DTLS sessions at the SSP. In the case of TER1 and TER, all requests are sent over CoAPs, and the DTLS session is terminated at the SSP. For TER1, there does not exist an active DTLS session between the proxy and the constrained node. Therefore, a new DTLS session must be setup between the SSP and the constrained node. For TER, the active DTLS session in the LLN can be re-used, and there is no need to setup a new DTLS session with the constrained node. For

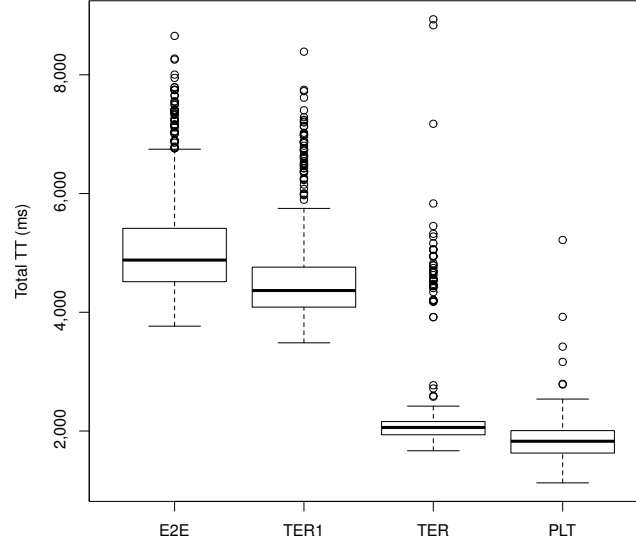
all DTLS cases, the DTLS client always sets up a new DTLS session at the start of a request sequence. It also tears down the existing session at the end of every sequence. As such, this testing scenario represents a large number of DTLS clients that would interact with the constrained CoAPs servers over the lifetime of the constrained node. For each configuration, the request sequence was run four hundred times, i.e., one hundred times per DTLS server. All results were obtained using the default CSMA MAC protocol and ContikiMAC RDC protocol as available in Contiki.

3.6.1.2 Results

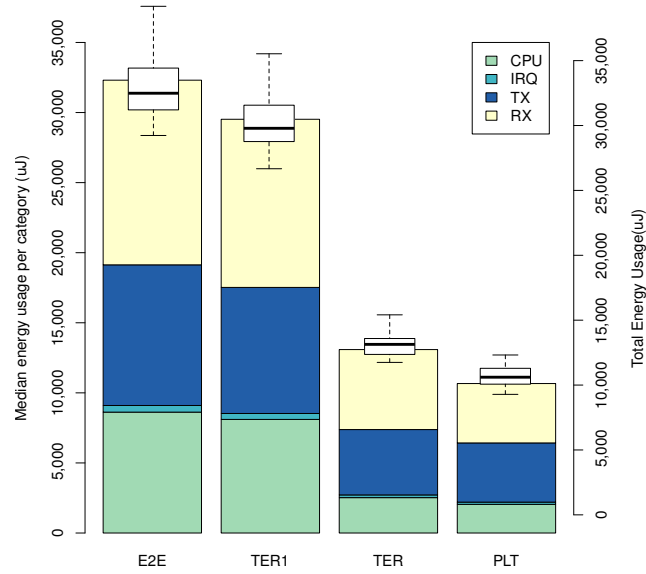
Figure 3.6(a) shows the Total Transaction Time (TTT). This is the time between the start of the DTLS session handshake (i.e., when the first ClientHello message is sent by the client) and the end of the DTLS session (i.e., when the DTLS Finished message is received by the client). There is a significant reduction in TTT between the E2E and the TER configurations: their medians are 4879 ms and 2060 ms, respectively. This is due to the DTLS session re-use in the LLN, which saves, when comparing the median cases, thirteen packets in the LLN, as the DTLS handshake in the LLN can be avoided in the TER configuration. As a result, the TER configuration is able to closely match the reference plain-text case in terms of TTT. The 233-ms difference in the median is caused primarily by the overhead of the additional DTLS headers. More specifically, the overhead triggers 6LoWPAN fragmentation for the large discovery response in the TER case, whereas this fragmentation is absent in the PLT case.

Figure 3.6(b) displays the energy usage for the different configurations. The stacked bar plot shows the median energy usage per category on the constrained device, whereas the box plot shows the total energy usage (to show the dispersion of the measurements). Again, there exist a significant difference between the E2E and the TER configurations: 32,485 J vs. 13,133 J respectively (a reduction by a factor of 2.4). Similarly to the TTT results, this reduction is primarily due to the absence of the DTLS handshake in the LLN. This is confirmed by the bar plot where the energy usage for the RX and TX categories are reduced the most. The energy consumption in the CPU category is also significantly lower, as the CPU is in low-power mode more often and does not have to perform expensive hash calculations when completing the handshake. All in all, the results allow us to conclude that our approach increases the responsiveness of constrained devices (provided there is an active session in the LLN) while reducing the energy consumption for traffic loads with many DTLS sessions (e.g., traffic loads with many parties).

Finally, it is worth pointing out that our approach drastically limits the total number of handshakes that a constrained node will perform during its lifetime. Apart from the savings discussed above, this also has the additional benefit that, in lossy networks, the total number of failed handshakes will be lower. Indeed, Garcia et al. [37] have shown that in lossy networks, the fraction of failed handshakes can vary significantly based on the packet loss ratio: e.g., 30 to 40% of handshakes fail for a packet loss ratio of ~20%. By limiting the total number of handshakes, our



(a) Total transaction times (TTT) for the request sequence



(b) Median energy usage per category (left axis) and total energy usage (right axis).

Figure 3.6: Transaction times and energy usage of the CoAPs servers for the three gateway configurations (End-to-End (E2E), first Termination (TER1), n -th Termination (TER)) and the Plain Text CoAP reference case (PLT).

approach also limits the amount of constrained device resources wasted on these failed handshakes. On the other hand, care should be taken to periodically refresh keying material as needed by the underlying cryptographic primitives in use.

3.6.2 Aggregating multiple CoAPs clients at the SSP

The second evaluation scenario focuses on the impact of the SSP on constrained devices that serve multiple CoAPs clients simultaneously via CoAPs observation. Unlike clear text CoAP observation, notifications for one CoAPs client typically cannot be reused to serve another client due to the confidentiality of the notification in DTLS. However, the SSP presented in this work can, as a reverse CoAPs proxy, observe one CoAPs resource on a constrained device and use these notifications to serve a multitude of CoAPs clients. The presented evaluation considers up to ten CoAPs clients that observe a resource on a constrained device and compares the case of end-to-end observation versus observation via the SSP. Note that one should keep in mind client authorization when using one CoAPs stream of notifications for serving multiple CoAPs clients, e.g., a client that is not authorized to access a resource on the constrained device must also be denied access to that resource via the SSP. To this end, this work presents and implements an access control adapter, which enforces CoAPs resource-specific access control policies.

3.6.2.1 Experiment setup

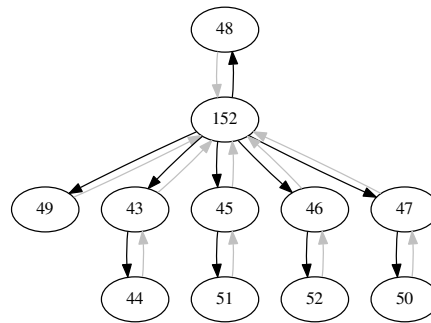


Figure 3.7: Representative RPL network topology: the node under study, node #50, is situated two hops from the border router, node #152.

To quantify the impact of aggregating CoAPs observations at the SSP, a number of experiments were run on a WSN testbed. The experiments consisted of a 6LoWPAN network with ten sensor nodes arranged on a line with six meters of spacing between adjacent nodes. An additional sensor node (Node #152) is situated to the upper left of the line and is connected to a Raspberry Pi 2, where it serves as the RPL border router. The smart service proxy software is running on the Raspberry Pi 2. In order to cope with the changes in the RPL topology

between experiments and over time in the same experiment, Node #50 was selected for testing as it was always located two hops away from the border-router. A representative network topology is shown in Figure 3.7. Note that depending on the experiment, Node #50 might have a different parent than Node #47 (e.g., Node #43 was a common alternative), but in all experiments, there was always an intermediary router between the border router and Node #50.

All wireless sensor nodes employ the msp430f5437 uC with 128 KB of RAM and 256 KB of ROM and the TI CC2520 802.15.4 transceiver. As such, the platform is identical to the WiSMote platform in Contiki in terms of the specifications that are relevant for the presented results. The sensor nodes run a TinyDTLS CoAPs server, which is configured to support three simultaneous DTLS sessions and one simultaneous DTLS handshake. While a binary for four simultaneous sessions could be built, it was not running stably. Attempts to support more than four clients led to a RAM overflow during linking. By default, `er-coap` in Contiki sends one confirmable notification for every twenty notifications. Finally, all sensor nodes in the network are running the default CSMA MAC protocol and ContikiMAC RDC protocol available in Contiki.

For every sensor node, a corresponding virtual host was created on the SSP. The virtual hosts were configured similar to the listing in Section 3.4.3.1, with support for the “`TLS.ECDHE.ECDSA.WITH.AES.128.CCM.8`” cipher suite. This cipher suite provides perfect forward secrecy by means of an ephemeral Diffie–Hellman key exchange between the virtual hosts and the DTLS clients. Additionally, DTLS clients authenticate virtual hosts by means of the x.509 certificates of the hosts, which are signed by a Certificate Authority (CA) trusted by the clients. Similarly, the DTLS clients also present an x.509 certificate during the DTLS handshake, which is signed by a CA that is trusted by the proxy. As a result, the clients may be authenticated at the proxy-side (by mapping attributes from the certificate to a user in the proxy; see Section 3.4.3.4), which is mandatory for the use of the access control adapter in order to provide fine-grained authorization as presented in Section 3.4.3.2. Each virtual host was allocated a global IPv6 address from the LAN network of the Raspberry Pi2 and has one default adapter chain with access control, caching and proxy adapters. The CoAPs clients ran as part of the CoAP++ framework on a PC that was located three IPv6 hops away from the Raspberry Pi2. All IPv6 addresses in use (i.e., CoAPs clients, RPI, virtual hosts and WSN nodes) were working, global IPv6 addresses. An overview of the evaluation setup is shown in Figure 3.8.

In all experiments, a number of CoAPs clients observes a resource on either the virtual host or the sensor node. As such, the experiments considered two cases: end-to-end (E2E) CoAPs observations and CoAPs observations via the SSP. In both cases, experiments were run for two CoAPs resources: a resource with a one-second notification period and another resource with a five-second notification period. In the E2E case, experiments were performed with one, two and three simultaneous CoAPs clients. In the SSP case, experiments were performed with one, two, three, four, five and ten simultaneous CoAPs clients. In total, eighteen experiments were performed. Each experiment was run for at least twenty minutes,

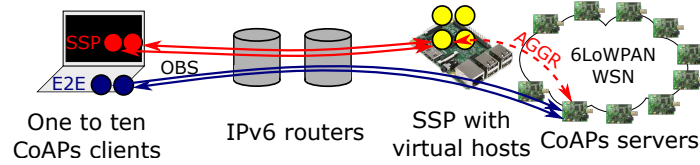


Figure 3.8: Evaluation setup: a variable number of CoAPs clients observe one of two resources on either the virtual host (SSP) or the sensor node (E2E)

during which the energy outputs for all sensor nodes were captured every five seconds, and the outputs from the CoAPs clients were stored, as well. This enabled us to quantify the energy usage, as well as the application-layer performance, the results of which are presented in the following section.

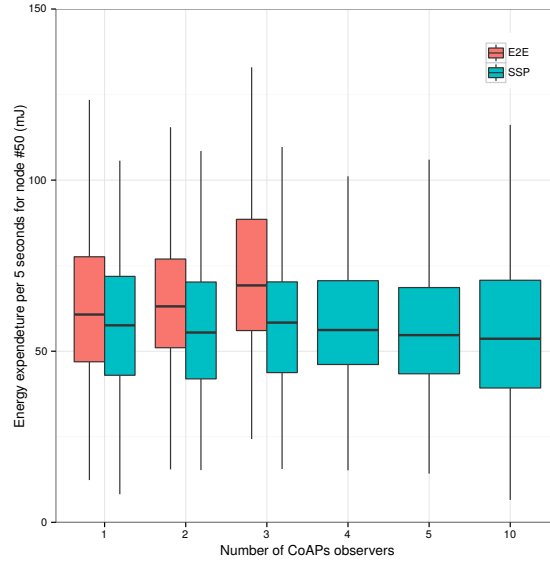
3.6.2.2 Results

When comparing the energy expenditure graphs for Node #50 in Figure 3.9, it becomes clear that aggregating CoAPs observation relationships leads to energy savings. The savings are proportional to the rate of notifications: they increase as the number of clients goes up and decrease as the notification interval becomes longer. Note that the sensor node between Node #50 and the border router experiences similar energy savings as every notification is received and retransmitted by this intermediary node. For the case of three CoAPs observers, the median energy expenditures differ by 10.8 mJ for the one-second interval and 2.5 mJ for the five-second interval.

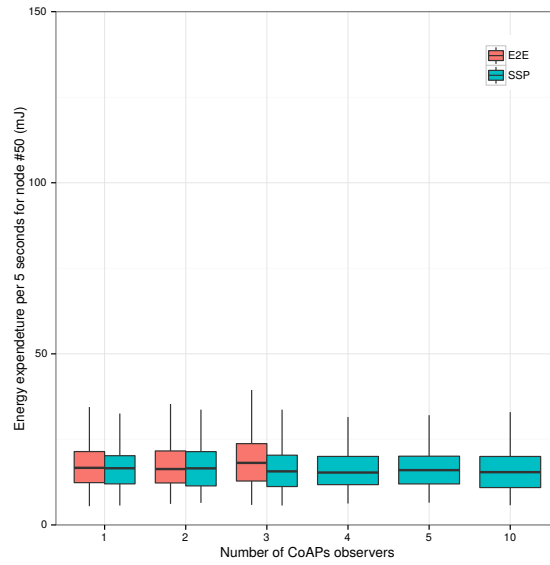
For one CoAPs observer and the one second interval, there exists a small difference in energy expenditure between the end-to-end and the SSP case even though the notification rate is the same for both cases (i.e., one notification per second). This is primarily due to a difference in notification packet size, as the 6LoWPAN compression for SSP notifications is more effective than for E2E notifications. The compression is more effective because the IPv6 address of the SSP is part of the 6LoWPAN network, whereas the CoAPs client's IPv6 address is part of a different network. As such, the prefix of the SSP's IPv6 address can be elided (due to stateful 6LoWPAN compression), which leads to an eight-byte savings in packet size per notification.

The graphs in Figure 3.10 clearly illustrate the difference in notification rate between the end-to-end and SSP experiments. Due to the aggregation of CoAPs observations at the SSP, there exists only one CoAPs observation between the SSP and the sensor node. This is illustrated in the constant notification rate for SSP as the number of CoAPs observers increases. For the end-to-end experiments, the notification rate rises linearly with the number of observers, as the sensor node sends notifications to each client separately. The slope of this linear relation is proportional to the notification frequency.

Figure 3.11 plots the Notification Loss Ratios (NLR) for each of the eighteen

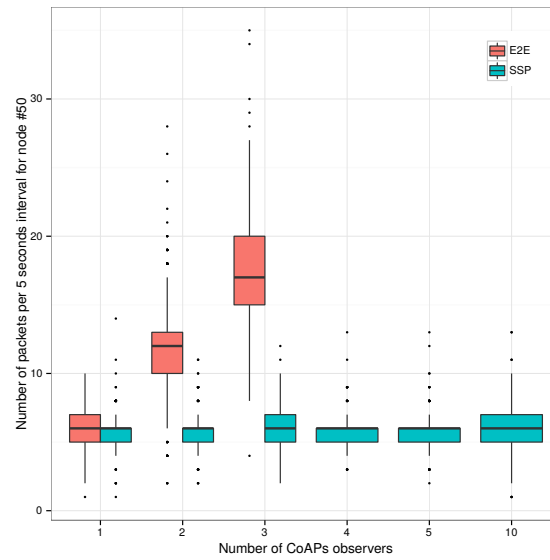


(a) One second notification interval.

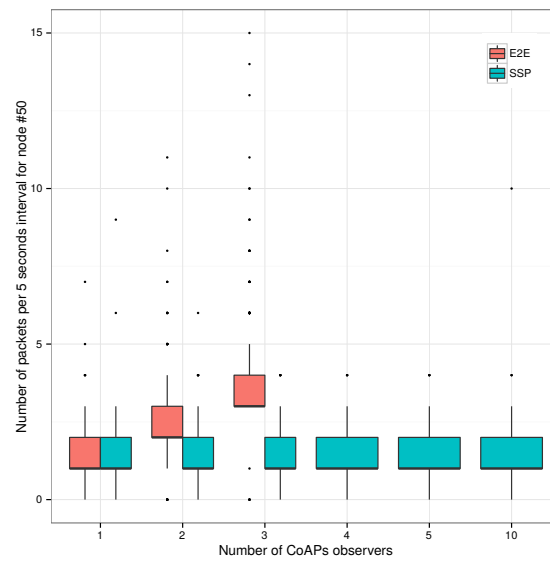


(b) Five seconds notification interval.

Figure 3.9: Total energy expenditure for node #50 per five seconds interval for end-to-end (E2E) CoAPs observation versus CoAPs observation through the Smart Service Proxy (SSP)



(a) One second notification interval.



(b) Five seconds notification interval.

Figure 3.10: Number of exchanged packets for node #50 per five seconds interval for end-to-end (E2E) CoAPs observation versus CoAPS observation through the Smart Service Proxy (SSP)

experiments. For example for the E2E, one-second interval and one observer case, 1845 notifications were sent, three of which never arrived at the client. This leads to an NLR of 0.163%. Note that every vertical series of data contains as many points as there are observers; however, very similar and identical NLR's overlap too much to distinguish them as separate points in the plot. The graphs for the one second interval show that the end-to-end case suffers from network congestion due to its higher notification rate. Furthermore, the observed loss is heavily dependent on the CoAPs client in the E2E experiments: i.e., the client that is last on the list of observers experiences the highest NLR (mostly apparent when there are three observers). Finally, the SSP sends every notification as a confirmable message. While in this setup, packet loss is mainly a problem in the constrained WSN, sending all notifications as CON messages can help to improve the NLR in situations where the client is a part of a lossy network.

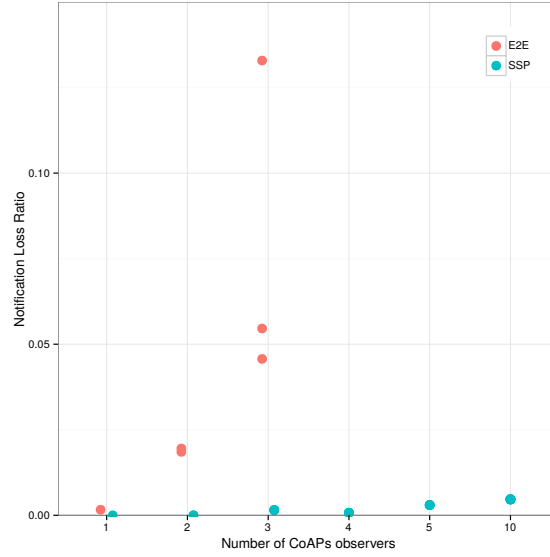
To conclude, there are a number of limitations that are overcome by aggregating observations at the SSP:

1. Memory and processing constraints on the sensor node, which limit the number of simultaneously active DTLS sessions and active CoAP observation relationships.
2. Limited throughput in constrained (multi-hop) networks, which impacts the successful delivery of notifications and limits the rate of notifications.
3. Limited lifetime for battery-operated sensors: by reducing the load on constrained devices, the lifetime is lengthened.

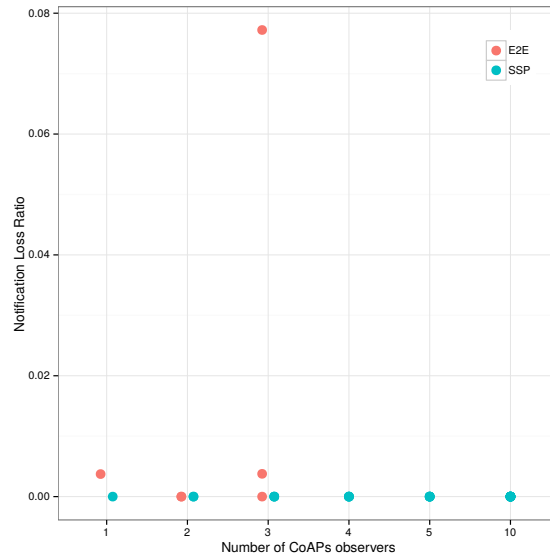
Note that while only the results for Node #50 are shown, similar savings apply for other nodes. Also note that applying observation aggregation at the SSP delays the point at which the WSN reaches congestion, as the message rate in the WSN is reduced by the aggregation. Finally, note that this experiment is only possible because the SSP terminates the end-to-end security; indeed, should this not be the case, then the SSP would be unable to aggregate observation relationships, as all communications would be encrypted end-to-end.

3.7 Conclusions

In this work, we have presented the Secure Service Proxy: a CoAP(s) intermediary for use in resource-constrained RESTful environments. It has been designed to provide scalable end-to-end security for constrained devices and to extend constrained devices with additional functionality. The presented work follows a reverse proxy approach, where the SSP hosts virtual devices on behalf of resource-constrained devices. This approach enables the SSP to extend the virtual devices with security features that are hard to attain in constrained environments, such as authentication based on public key infrastructure (which, inherently, scales better than using PSKs), perfect forward secrecy and fine-grained authorization based on



(a) One second notification interval.



(b) Five seconds notification interval.

Figure 3.11: Notification loss ratios as measured at the CoAPs clients for end-to-end (E2E) CoAPs observation versus CoAPs observation through the Smart Service Proxy (SSP)

host identify and the nature of the request and resource. Additionally, the SSP extends virtual devices with a variety of different functions by means of an adapter chain system. Adapters are modular blocks of functionality that are hosted on the virtual device. Examples include caching, static resource and congestion control adapters. The SSP hosts a RESTful Web interface for managing virtual devices and adapter chains.

The SSP has been evaluated in two different setups. First, tests were performed in an LLN simulator to measure the effect of terminating end-to-end security on the SSP. The results of the simulator study demonstrate that session termination combined with long-term sessions in the constrained network leads to significant savings in network traffic, communication delay and processing and, consequently, leads to a longer battery life. The second study was run on a WSN testbed and quantified the impact of aggregating multiple observation relations with a constrained device over DTLS. The results confirm that the load on the constrained device and constrained network is independent of the number of observers. As a result, the packet rate and energy expenditure remain equal to those of the one observer case as the number of observers increases. Note that the session termination is a necessary condition for observation aggregation in case of DTLS-based security.

In conclusion, the presented Secure Service Proxy breaks end-to-end security in order to offer security primitives that are hard to attain on constrained systems while reducing the load on resource-constrained devices and networks. Additionally, the proxy provides extra application-layer features on behalf of constrained devices to services, which are built on top of these devices. Combined, the proxy facilitates the integration of constrained RESTful environments in services, thereby furthering the vision of an open, secure and scalable Web of Things.

References

- [1] Joseph Bradley, J. Barbier, and D. Handler. *Embracing the Internet of Everything To Capture Your Share of 14.4 Trillion USD*. Cisco Ibsg Group, page 2013, 2013. Available from: <http://www.cisco.com/web/about/ac79/docs/innov/IoE{ }Economy.pdf>.
- [2] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. *Internet of things: Vision, applications and research challenges*. Ad Hoc Networks, 10(7):1497–1516, 2012.
- [3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. *Internet of Things (IoT): A vision, architectural elements, and future directions*. Future Generation Computer Systems, 29(7):1645–1660, sep 2013. Available from: <http://www.sciencedirect.com/science/article/pii/S0167739X13000241>, doi:10.1016/j.future.2013.01.010.
- [4] C. Bormann, M. Ersue, and A. Keranen. *RFC 7228: Terminology for Constrained-Node Networks*. Technical report, IETF, 2014. Available from: <http://tools.ietf.org/html/rfc7228>.
- [5] P. Baronti, P. Pillai, V. W. C. Chook, S. Chessa, A. Gotta, and Y. F. Hu. *Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards*. Computer Communications, 30(7):1655–1695, 2007. Available from: <http://www.sciencedirect.com/science/article/pii/S0140366406004749>, doi:<http://dx.doi.org/10.1016/j.comcom.2006.12.020>.
- [6] T. Winter and P. Thubert. *RFC 6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, 2012. Available from: <https://tools.ietf.org/html/rfc6550>, doi:10.17487/RFC6550.
- [7] H. Tschofenig and T. Fossati. *RFC 7925: Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things*. RFC 7925, 2016. Available from: <https://rfc-editor.org/rfc/rfc7925.txt>, doi:10.17487/RFC7925.
- [8] M. Vucinic, B. Tourancheau, T. Watteyne, F. Rousseau, A. Duda, R. Guizzetti, and L. Damon. *DTLS Performance in Duty-Cycled Networks*. In International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC - 2015), 2015. Available from: <http://arxiv.org/abs/1507.05810>, arXiv:1507.05810.
- [9] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. *RFC 7252: Constrained Application Protocol (CoAP)*, 2014. Available from: <https://tools.ietf.org/html/rfc7252>.
- [10] K. Kuladinithi, O. Bergmann, and M. Becker. *Implementation of CoAP and its Application in Transport Logistics*. In Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks, 2011.

- [11] K. Hartke. *RFC 7641: Observing Resources in the Constrained Application Protocol (CoAP)*. Technical report, IETF, 2015. Available from: <https://tools.ietf.org/html/rfc7641>.
- [12] E. Rescorla and N. Modadugu. *RFC 6347: Datagram transport layer security version 1.2*, 2012. Available from: <https://tools.ietf.org/html/rfc6347>.
- [13] T. Dierks and E. Rescorla. *RFC 5246: The Transport Layer Security (TLS) protocol version 1.2*. Technical report, IETF, 2008. Available from: <https://tools.ietf.org/html/rfc5246>.
- [14] D. McGrew. *RFC 5116: An Interface and Algorithms for Authenticated Encryption*. Technical report, IETF, 2008. Available from: <https://tools.ietf.org/html/rfc5116>.
- [15] D. McGrew and D. Bailey. *RFC 6655: AES-CCM Cipher Suites for Transport Layer Security (TLS)*. Technical report, IETF, 2012. Available from: <https://tools.ietf.org/html/rfc6655>.
- [16] P. Eronen and H. Tschofenig. *RFC 4279: Pre-shared key ciphersuites for transport layer security (TLS)*. Technical report, IETF, 2005. Available from: <https://tools.ietf.org/html/rfc4279>.
- [17] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen. *RFC 7250: Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. Technical report, IETF, 2014. Available from: <https://tools.ietf.org/html/rfc7250>.
- [18] D. Bailey, M. Campagna, R. Dugal, and D. McGrew. *RFC 7251: AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS*. Technical report, IETF, 2014. Available from: <https://tools.ietf.org/html/rfc7251>.
- [19] E. Rescorla. *RFC 4492: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode*. Technical report, IETF, 2008. Available from: <https://tools.ietf.org/html/rfc4492>.
- [20] E. Barker. *NIST Special Publication 800-57 Part 1 Revision 4, Recommendation for Key Management Part 1: General*. Technical report, NIST, 2016. Available from: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf>, doi:10.6028/NIST.SP.800-57pt1r4.
- [21] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. *RFC 6960: X. 509 Internet public key infrastructure online certificate status protocol-OCSP*. Technical report, IETF, 2013. Available from: <https://tools.ietf.org/html/rfc6960>.
- [22] F. Van den Abeele, T. Vandewinckele, J. Hoebeke, I. Moerman, and P. Demeester. *Secure communication in IP-based wireless sensor networks via a trusted gateway*. In IEEE Tenth International Conference on Intelligent

- Sensors, Sensor Networks and Information Processing (IEEE ISSNIP 2015), 2015.
- [23] Z. Shelby. *RFC 6690: Constrained RESTful Environments (CoRE) Link Format*, 2012. Available from: <https://tools.ietf.org/html/rfc6690>.
- [24] G. Selander, J. Mattsson, F. Palombini, and L. Seitz. *Object Security of CoAP (OSCOAP) (Work in progress)*, 2017. Available from: <https://tools.ietf.org/html/draft-ietf-core-object-security-03>.
- [25] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. *Securing communication in 6LoWPAN with compressed IPsec*. In 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS), pages 1–8, 2011. doi:10.1109/DCOSS.2011.5982177.
- [26] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. *Fog computing and its role in the internet of things*. Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12, page 13, 2012. Available from: <http://dl.acm.org/citation.cfm?doid=2342509.2342513>, doi:10.1145/2342509.2342513.
- [27] M. Nitti, V. Pilloni, G. Colistra, and L. Atzori. *The Virtual Object as a Major Element of the Internet of Things: A Survey*. IEEE Communications Surveys & Tutorials, 18(2):1228–1240, 2016. Available from: <http://ieeexplore.ieee.org/document/7320954/>, doi:10.1109/COMST.2015.2498304.
- [28] M. Kovatsch, S. Mayer, and B. Ostermaier. *Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things*. In 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pages 751–756, 2012. Available from: http://ieeexplore.ieee.org/xpls/abs/_all.jsp?arnumber=6296948, doi:10.1109/IMIS.2012.104.
- [29] A. J. Jara, P. Moreno-Sanchez, A. F. Skarmeta, S. Varakliotis, and P. Kirstein. *IPv6 addressing proxy: mapping native addressing from legacy technologies and devices to the Internet of Things (IPv6)*. Sensors (Basel, Switzerland), 13(5):6687–712, jan 2013. Available from: <http://www.mdpi.com/1424-8220/13/5/6687/htm>, doi:10.3390/s130506687.
- [30] A. Ludovici and A. Calveras. *A Proxy Design to Leverage the Interconnection of CoAP Wireless Sensor Networks with Web Applications*. Sensors, 15(1):1217–1244, jan 2015. Available from: <http://www.mdpi.com/1424-8220/15/1/1217>, doi:10.3390/s150101217.
- [31] A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk. *RFC 8075: Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)*. Technical report, IETF, 2017. Available from: <https://tools.ietf.org/html/rfc8075>.

- [32] E. Mingozzi, G. Tanganelli, and C. Vallati. *CoAP Proxy Virtualization for the Web of Things*. In 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, pages 577–582. IEEE, dec 2014. Available from: <http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=7037719>, doi:10.1109/CloudCom.2014.163.
- [33] G. Tanganelli, C. Vallati, E. Mingozzi, and M. Kovatsch. *Efficient proxying of CoAP observe with quality of service support*. In 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), pages 401–406. IEEE, dec 2016. Available from: <http://ieeexplore.ieee.org/document/7845444/>, doi:10.1109/WF-IoT.2016.7845444.
- [34] I. Farris, A. Lera, A. Molinaro, and S. Pizzi. *A CoAP-compliant solution for efficient inclusion of RFID in the Internet of Things*. In 2014 IEEE Global Communications Conference, pages 2795–2800. IEEE, dec 2014. Available from: <http://ieeexplore.ieee.org/document/7037231/>, doi:10.1109/GLOCOM.2014.7037231.
- [35] R. Hummen, H. Shafagh, and S. Raza. *Delegation-based Authentication and Authorization for the IP-based Internet of Things*. In 11th IEEE International Conference on Sensing, Communication, and Networking (SECON'14), 2014. Available from: <http://www.comsys.rwth-aachen.de/fileadmin/papers/2014/2014-hummen-secon-delegation.pdf>.
- [36] J. Park, H. Kwon, and N. Kang. *IoTCloud collaboration to establish a secure connection for lightweight devices*. *Wireless Networks*, 23(3):681–692, apr 2017. Available from: <http://link.springer.com/10.1007/s11276-015-1182-y>, doi:10.1007/s11276-015-1182-y.
- [37] O. Garcia-Morchon, S. L. Keoh, S. Kumar, P. Moreno-Sanchez, F. Vidal-Meca, and J. H. Ziegeldorf. *Securing the IP-based internet of things with HIP and DTLS*. *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks - WiSec '13*, page 119, 2013. Available from: <http://dl.acm.org/citation.cfm?doid=2462096.2462117>, doi:10.1145/2462096.2462117.

4

Improving User Interactions with Constrained Devices in the Web of Things

While the previous chapter focused on the underlying technical principles of Distributed Intelligence (DI), this chapter employs Sensor Function Virtualization (SFV) to improve user interactions with constrained RESTful devices. As constrained devices are unable to offer rich Graphical User Interfaces (GUIs), this chapter presents a web template rendering system that transforms CoAP responses into HTML pages that may be served to a user's mobile device over HTTP. Using bootstrap, a common web template, responsive web GUIs are demonstrated for device and resource discovery, as well as temperature monitoring and actuator control.

Floris Van den Abeele, Enri Dalipi, Ingrid Moerman, Piet De-meester and Jeroen Hoebeke

Published in the proceedings of the IEEE World Forum on the Internet of Things (WF-IoT 2016), 12–14 Dec. 2016, Reston, Virginia, USA.

Abstract As the Internet of Things continues to grow rapidly in the coming years, the number of devices with limited resources will continue to grow as well. These so-called constrained devices typically implement specialized protocols and data

formats for increased efficiency. While this reduces the load on constrained devices, it also limits the usability of such devices for their users. This paper presents a HTTP-CoAP proxy for improving the usability of constrained devices that implement embedded web services. This is accomplished by rendering user interfaces and solving naming and routing issues. As a result, the user experience of highly-optimized embedded web services is similar to that of conventional web services. By means of small-scale experimentation, the presented approach is evaluated functionally and the usability is evaluated in terms of user interface responsiveness.

4.1 Introduction

According to numerous market research reports [1] [2] [3], the number of Internet-connected devices will have risen drastically by the end of this decade. A considerable amount of these newly connected devices, are so-called “constrained devices”, i.e. devices with tight limits on power, memory, connectivity, processing power, cost and physical size. Due to their anticipated widespread usage, IoT devices are expected to impact a large number of aspects of our society and our daily lives by enabling a whole new range of services [4]. In order to realize these services, IoT devices have to interact with each other, with their environments and with their users.

On the other hand, users of such services expect an experience similar to the conventional Internet services with which they are familiar: e.g. web browsing, mobile applications, web search, etc. For services that build on constrained devices, this is a challenge as the constraints intrinsic to these devices impose limits on their functionality and as such on their usability. Consider, for example, a battery-powered air monitoring device with a hundred kilo bytes of memory. For such a device it is impossible to offer a web-based user interface that offers a user experience similar to today’s popular web platforms.

There exist a number of solutions for overcoming this problem. A subset of these rely on systems external to the constrained device for overcoming the limited usability of such devices. More specifically, this work presents a web proxy based approach for improving interactions between users and constrained devices in a web of things context. We demonstrate how our approach solves a number of usability problems common to low-power and embedded web services, thereby showing the feasibility and effectiveness of our work.

4.2 Problem statement and research goals

Constrained devices are subject to a number of limitations, most of which are the result of the required low device cost. As these limitations directly impact the usability of constrained devices, they are briefly discussed in this paragraph. Firstly, popular low-power micro controller families such as the ARM Cortex M3,

TI MSP430 and AVR ATmega offer many different models where the available volatile memory varies between 8KB and 100KB and the read-only memory between 32K and 1024KB. In every use case a trade-off has to be made between cost vs available memory space: 16KB RAM and 128 or 256KB ROM is a common choice for systems in sensor and mesh networks. For battery-powered devices, power consumption is a second important consideration. Low-cost devices typically have lifetimes equal to their battery lifetimes, as replacing the battery is deemed too expensive. As a result, energy consumption should be kept to a minimum by e.g. limiting computation and communication. Finally, a third important constraint of networked systems is the employed communication technology. For our discussion, it should enable low-power communication while keeping the component cost (e.g. transceiver, amplifiers, antenna) low. A comprehensive overview of the constraints is available in RFC 7228 [5]. As per RFC 7228 terminology, this work focuses on Class 1 constrained devices, with $\sim 10\text{KiB}$ RAM and $\sim 100\text{KiB}$ ROM.

Each of these discussed limitations impacts the usability of constrained devices in different ways. In the class 1 systems under consideration, the limited memory commonly has to fit the entire communication stack (i.e. everything above the PHY layer) as well as the necessary logic to realize the intended service. As a result, the remaining amount of memory left to also implement a high quality user interface is typically very low. For example, a simple index page based on the popular bootstrap template for responsive web interfaces¹ requires 89.6KB of memory: Javascript (minimal jQuery and bootstrap: 44.6KB), CSS styling (38.4KB) and HTML (6.5KB). One can increase the available memory to include all necessary files or host the static media files (JS and CSS) externally. Even so, a considerable amount of additional data for the UI (in the order of (tens of) kilo bytes) would have to be transferred between the constrained device and the client. For battery-powered devices, this would drastically hasten the depletion of the energy source and therefore limit the lifetime of the device. In the case of low-throughput networks, transferring the additional UI data could lead to long latency penalties as the networks are not dimensioned to transmit large chunks of data. Consequently, user experience would suffer under these long delays. As a result, class 1 devices are considered to offer ‘bare-bone’ RESTful resources - via the specialized CoAP protocol - that are cumbersome to use due to the lack of a UI.

Low-power network protocols such as 6LoWPAN and the RPL routing protocol also impact the usability of constrained nodes. In such networks, separate IPv6 networks are typically assigned to the low-power and lossy networks (LLNs). In cases where global IPv6 routing for the LLNs is unfeasible (e.g. private LLNs), the user is expected to reconfigure its network configuration to add a routing rule to the LLN. For most users this is unrealistic. Additionally, the use of IPv6 means that constrained nodes are reachable via 128 bit IPv6 addresses. As these long addresses are impractical for human users, an alternative has to be provided. Also, discovery of devices by a user might be difficult.

¹<https://getbootstrap.com/>

The goal of this work is to answer the following research question: given the problems outlined above, how can direct user interactions with constrained devices in low-power and lossy networks be improved? In answering the question, this work looks at the problem from an embedded web services point of view as realized with the IPv6 and CoAP protocols [6]. Although the focus is on these technologies, the core concepts of this work are more broadly applicable.

4.3 User friendly interactions

4.3.1 Requirements

In analyzing the posed research question, the following requirements for suitable solutions that improve user interactions are identified:

1. **Impact on constrained devices should be kept to a minimum.** Consequently the device constraints outlined in the previous section remain unaltered.
2. **Easy to use interfaces for the user.** The user experience should be similar to popular web-based services.
3. **Handle a wide variety of constrained and user devices.** When looking only at embedded web services, a constrained device can serve many purposes. Similarly, there exists a large range of user devices.
4. **Minimal configuration and easy discovery.** Any usable solution should require minimal configuration from the user. It should also facilitate easy discovery of constrained devices and their services.
5. **Easy to build user interfaces.** While building interfaces will require some technical knowledge, it should be based on open and readily available technology to facilitate designers.

There are a number of approaches that fit the requirements outline above, some of which are presented in the related work section. The approach in this work is discussed in the following section and relies on web-based application proxies combined with naming and discovery services.

4.3.2 Approach

Figure 4.1 outlines the approach of this work and how it differs from what is available today. Today, users interface with devices directly via CoAP or indirectly via HTTP through a gateway. In both cases the user is served the unaltered CoAP response, which is typically encoded in a compact but obscure binary format. Our approach introduces web-based application proxies, whose main task is serving web interfaces to users. The proxies process web requests from the user's browser and translate the requests into RESTful CoAP requests for constrained devices.

Responses from constrained devices are processed by proxies and used as input for rendering web interfaces to users.

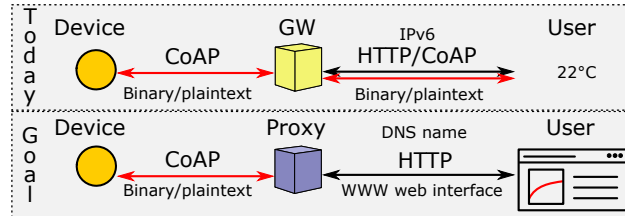


Figure 4.1: Our approach serves users web interfaces of embedded web services on constrained devices.

One of the benefits of this approach is that all user interactions with the constrained devices make use of standard web technology (i.e. TCP/HTTP/WWW). As a result, the web interfaces are available to a wide range of user devices (only a web browser is needed). The proxies implement a template lookup interface that returns the web interface template to be used for rendering a response to the user. This lookup interface takes into account the resource type of the RESTful resource and the device type of the user device. Combined with leveraging the CoAP standard, different web interfaces can be rendered for different types of resources thereby supporting a wide range of constrained devices. Additionally, the device detection of the proxies combined with well-known web technology for designing templates enable user friendly interfaces that are adapted to the device of the user.

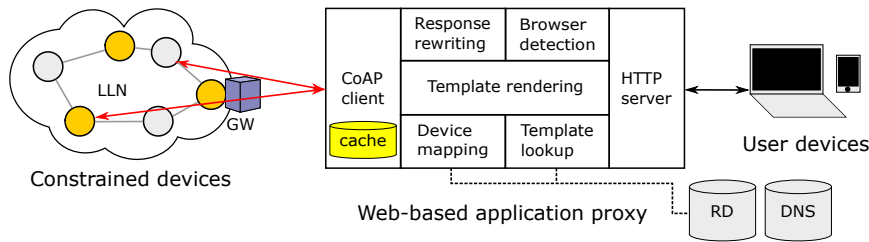


Figure 4.2: System overview

As CoAP is a highly optimized, binary application protocol specifically designed for constrained environments, the impact of our approach on constrained devices in terms of memory usage, communication overhead and battery life is kept to a minimum. Also, this approach does not require any changes to the software running on the constrained devices or the user's web browser. Finally, by interfacing with directory and naming services the proxies enable easy discovery and naming of devices respectively.

4.3.3 Design

An overview of the resulting system is shown in figure 4.2. The devices through which a user interacts with the constrained devices are shown on the right, the constrained devices are shown on the left. The middle shows the seven submodules which make up the design of the web-based application proxy. In order to offer all necessary functionality the proxies also interface with a Resource Directory (RD) [7] and a DNS name server, these are shown in the bottom right.

When a user directs its browser to a constrained device via the proxy, the proxy starts processing the HTTP request by extracting the target CoAP URI of the constrained device from the hosting HTTP URI. Depending on whether the proxy is processing the request as a forward or reverse HTTP-CoAP cross protocol proxy (as per “Guidelines for HTTP-to-CoAP Mapping Implementations” terminology [8]), the extraction method will differ. In the forward proxy case, the target CoAP URI can readily be extracted from the hosting HTTP URI; e.g.: `http://proxy.example.com/hc/coap://s.example.com/light`. In the reverse proxy case, where the user surfs directly to the device (e.g. `http://s.example.com/light`), the device mapping submodule maps the HTTP URI to the target CoAP URI.

Once the target CoAP URI is known, the proxy looks up the template for rendering the CoAP resource response to the user. The template lookup module maintains a database of templates for different CoAP resource types and different web browsers (mobile, desktop and miscellaneous). In case the resource type of the CoAP resource is unknown, the template lookup module contacts the Resource Directory to retrieve all meta information related to the target CoAP URI. The browser of the user is identified by the browser detection module. This module processes the HTTP header fields (mainly the User-Agent header) and determines whether the user is surfing from a mobile or desktop device. Once the user’s browser type and CoAP resource type are known, the template lookup module searches for a matching web template and returns the result to the template renderer. In case no matching template is found, a default template may be used depending on the user browser.

Apart from the web page contents, a template also specifies whether the proxy should wait for the CoAP response before rendering the template and returning the HTTP response to the user. As CoAP response times might be long and unpredictable (order of seconds), the user could experience long delays if the proxy were to wait on the CoAP response for rendering the web interface. Therefore, templates that anticipate long response times can indicate to the proxy that they should be rendered immediately. These templates then retrieve the CoAP response (via the proxy) once they have been rendered by the browser of the user.

For retrieving CoAP responses, a template may employ Asynchronous JavaScript And XML (AJAX) techniques to send an AJAX request to the hosting HTTP URI. The proxy detects that the request is an AJAX request (via the HTTP XMLHttpRequest header) and skips the web template lookup procedure (the web browser is detected as a miscellaneous device in this case). Instead the proxy sends a CoAP request to the target CoAP URI and returns the CoAP response in

the AJAX response (which might take a long time). The template is then free to process the AJAX response: e.g. update a text area, a graph, an HTML form, etc. Additionally, these AJAX techniques can be used to drive an actuator (via PUT or POST requests), to poll a resource periodically (e.g. while updating a graph), ... Note that services building on top of the proxy for data access, will not be served a web interface as their user agent is not recognized as a web browser: e.g. the user-agent of the urllib HTTP client in Python 3.4 is "Python-urllib/3.4". In this case the proxy operates as a standard HTTP-CoAP proxy.

The CoAP client module in the proxy sends requests to the constrained devices for retrieving CoAP responses. It incorporates a cache in order to speed up response retrieval.

The final module in the design is the response rewriting block. For certain Content-Types, this block rewrites the CoAP response in order to display it in the web interface. At the moment, the block only rewrites CoRE link format responses [9] by replacing web links with links that are handled by the proxy. This is necessary when discovering CoAP devices and resources via the proxy, as explained next.

4.3.4 Device mapping, discovery and naming

In the reverse HTTP-CoAP configuration, one might wonder how the device mapping module builds the mapping from HTTP to CoAP URIs. As minimal configuration is an important requirement, the user cannot be expected to maintain this mapping. Instead, the proxy retrieves a list of known constrained devices from the Resource Directory and assigns reverse IPv6 LAN endpoints for each of these devices. In order to make these new endpoints discoverable, the proxy registers the reverse endpoints in the RD (with the same resources as the constrained endpoint). Thus the RD contains both the known constrained devices (in a non-default domain, which is used by the proxy) and the corresponding reverse endpoints (in the default domain, which is used by users for discovery). As a result, when users surf to a reverse endpoint (as discovered in the RD), the proxy is readily able to determine the target CoAP URI. An example of this mapping and discovery procedure is presented in the evaluation section.

In the forward HTTP-CoAP configuration, the device mapping is not needed as the URI mapping is explicit. In this configuration, the user discovers the proxy by means of a HTTP resource with resource type "core.hc" (as per [8]) in the proxy's .well-known/core.

As mentioned earlier, the use of IPv6 and 6LoWPAN can lead to long IPv6 literals in hosting HTTP URIs. To remedy this, the proxy offers a /dns resource for each constrained device that renders a form where users can set a DNS hostname for the constrained device. Afterwards, users can use the host name instead of the IPv6 literal for surfing to the device. Alternatively, the host name could also be retrieved from a CoAP resource on the constrained device itself (e.g. in case the device was preprogrammed with a host name).

4.4 Evaluation

4.4.1 Evaluation setup

For evaluating the web-based application proxy, extensive tests are performed using the setup depicted in figure 4.3. The setup consists of two types of constrained devices: Zolertia Z1s sensor nodes and nodeMCU ESP8266 nodes. There are eight Z1s that form a 6LoWPAN LLN where one Z1 is connected via SLIP to the Raspberry Pi as the border router. Note that the 6LoWPAN network is a private network as the IPv6 prefix (fd00::/64) is a unique local prefix. The Z1s are equipped with a msp430f2617 micro controller (8KB RAM and 92KB flash memory), an IEEE 802.15.4 CC2420 transceiver and run the Contiki OS. The nodeMCUs are IPv4 only devices and are connected to the Raspberry Pi via the Wi-Fi access point. NodeMCUs are based on the low-power ESP8266 ESP-12E Wi-Fi SoC and have 32KB RAM and 4MB flash memory. Both the Z1s and the nodeMCUs are running CoAP servers. All constrained devices are configured to register themselves with the Resource Directory at start-up.

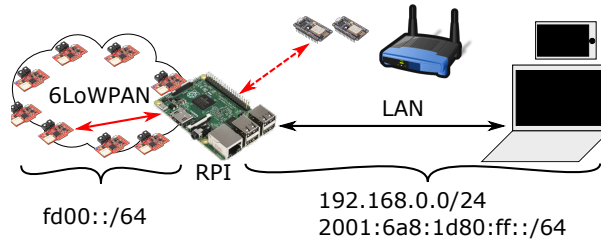
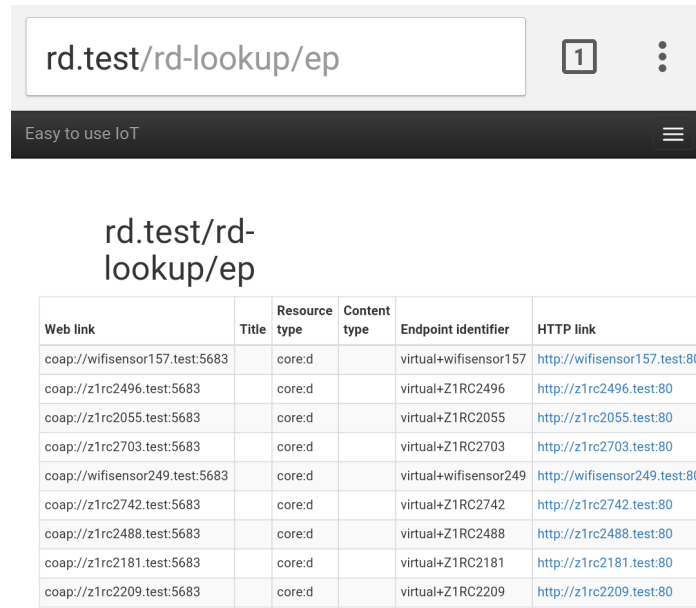


Figure 4.3: Evaluation setup: 6LoWPAN Zolertia Z1's (red) and 802.11 WLAN nodeMCUs (gray) as constrained devices. Raspberry Pi as application proxy. Red arrows indicate CoAP exchanges, black arrows HTTP exchanges. Solid lines indicate IPv6 datagrams, dashed lines IPv4.

The Raspberry Pi is a dual-stack device and is part of both the 6LoWPAN network and the LAN network. The RPI runs the application proxy, as well as the resource directory and the DNS server for the LAN network. The application proxy is implemented as part of our CoAP++ framework, which is built on top of Click Router. The proxy is configured as a reverse proxy for each of the constrained devices. As such, the proxy listens for new device registrations in the RD, allocates reverse endpoints in the IPv6 LAN network, stores the resulting device mapping and registers the allocated endpoint in the resource directory. As a result, the user can access constrained devices (6LoWPAN or Wi-Fi) through the reverse endpoints via the proxy. Finally, the notebook is running Ubuntu 14.04 and the smart phone is a Google Nexus 5. The round trip time between the Z1's and the notebook was measured via ping6 and is on average (μ) 163.1 ms with standard deviation (σ) 69.4 ms.

4.4.2 Functional evaluation

In order to discover the constrained devices, the user surfs to the resource directory in its browser: `http://rd.test/` which redirects the user to the `rd-lookup/ep` web interface. This HTTP request is handled by the proxy and is translated into a CoAP request for the local resource directory. The CoRE link format discovery response is rewritten to a HTML table and this table is rendered in the template for the `core.rd-lookup` resource type. Figure 4.4 shows the discovery response in the mobile Google Chrome browser.

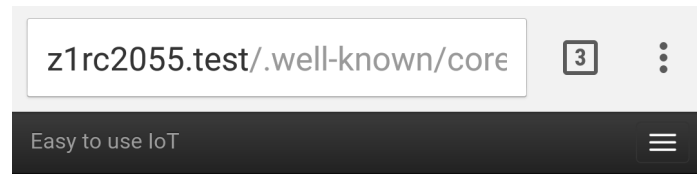


Web link	Title	Resource type	Content type	Endpoint identifier	HTTP link
coap://wifisensor157.test:5683		core:d		virtual+wifisensor157	http://wifisensor157.test:80
coap://z1rc2496.test:5683		core:d		virtual+Z1RC2496	http://z1rc2496.test:80
coap://z1rc2055.test:5683		core:d		virtual+Z1RC2055	http://z1rc2055.test:80
coap://z1rc2703.test:5683		core:d		virtual+Z1RC2703	http://z1rc2703.test:80
coap://wifisensor249.test:5683		core:d		virtual+wifisensor249	http://wifisensor249.test:80
coap://z1rc2742.test:5683		core:d		virtual+Z1RC2742	http://z1rc2742.test:80
coap://z1rc2488.test:5683		core:d		virtual+Z1RC2488	http://z1rc2488.test:80
coap://z1rc2181.test:5683		core:d		virtual+Z1RC2181	http://z1rc2181.test:80
coap://z1rc2209.test:5683		core:d		virtual+Z1RC2209	http://z1rc2209.test:80

Figure 4.4: Device discovery via the RD endpoint lookup interface

Next, the user taps on the HTTP link of the device of interest which takes the user to the `.well-known/core` resource. Here all the resources offered by the device as well as HTTP links are listed, as shown in figure 4.5.

Finally, the user taps on a resource of interest to interact with. Depending on the resource type, the interface will be different. Figure 4.6(a) shows a template that periodically updates a graph (using `chartjs.org`) of a temperature resource. The underlying `/sensors/temp` CoAP resource returns plain-text temperature readings. Figure 4.6(b) shows a template that renders a button for controlling an actuator (in this case a LED is turned ON and OFF). Tapping the button sends an HTTP POST request to the resource on the proxy, which is sent to the underlying CoAP resource by the proxy. Note that the temperature resource is hosted on a 6LoWPAN device, whereas the actuator resource is hosted on a Wi-Fi device.



z1rc2055.test/.well-known/core

Web link	Title	Resource type	Content type	HTTP link
/.well-known/core			40	http://z1rc2055.test/.well-known/core
/deviceName		ibcn.dev.name		http://z1rc2055.test/deviceName
/owner		ibcn.owner		http://z1rc2055.test/owner
/d		ibcn.dev		http://z1rc2055.test/d
/sensors/temp		ibcn.temp		http://z1rc2055.test/sensors/temp
/actuators/fan		ibcn.fan		http://z1rc2055.test/actuators/fan
/location		ibcn.location	0	http://z1rc2055.test/location
/image		ibcn.image	1001	http://z1rc2055.test/image
/dns		ibcn.dns	0	http://z1rc2055.test/dns

Figure 4.5: Rendering .well-known/core of a constrained device

4.4.3 Interface responsiveness: load times

The previous section illustrated the user interfaces that can be expected from our approach. An important performance metric of such user interfaces is the responsiveness: e.g. a sluggish interface can ruin the user experience. To qualify the responsiveness, the load times for two different types of templates are compared: a simple template, which blocks on the CoAP response before it is rendered, and an AJAX template, which is rendered immediately and fetches the response afterwards. In order to measure worst case responsiveness, caching in the CoAP client has been disabled.

The cumulative distribution functions of the load times are plotted in figure 4.7 (for 100 measurements per function). Notice that the load time² of the AJAX template (blue line) is always smaller and more consistent when compared to the simple template (green line). While the AJAX template has to perform a second

²As browsers typically render a considerable fraction of the UI before the load time has expired, the load time is considered an upper limit for the UI responsiveness. However, larger average load times do indicate a decrease in UI responsiveness.

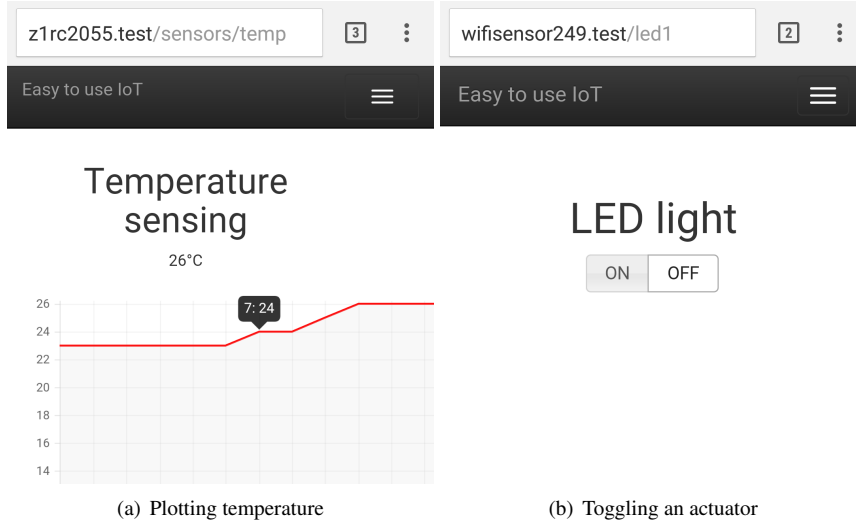


Figure 4.6: Different templates are rendered depending on the resource type of the target CoAP resource: e.g. `ibcn.temp` and `ibcn.light` are shown here.

request to fetch the CoAP response (yellow line), it is already rendered in the browser before this second request is issued. Also note that when the round trip time to the constrained devices would increase, the load time for the simple template would increase (green line would shift to the right) whereas the load time for the AJAX template remains constant as it does not depend on constrained device communication. As such, AJAX templates are clearly superior to simple templates in terms of responsiveness. Finally, the difference between the CoAP (black) and no template (red) lines shows that the delay introduced by our web interface rendering approach is around 14 ms (their minimums differ 13.9 ms).

4.5 Related work

There are many options for improving user interactions with constrained devices. In the HTTP-CoAP protocol category, the work of Ludovici et al. [10] presents a design of a forward cross protocol proxy that supports event-like notifications through WebSockets as an alternative to HTTP long polling. In contrast to our work, the proxy of Ludovici et al. does not provide a user interface for web browser-based users. Additionally, the proxy only operates in a forward configuration which requires the user to support the URI format implemented by the proxy. In [11] Colitti et al. describe both a HTTP-CoAP proxy and a HTTP web application for visualizing sensor measurements from a wireless sensor network. While the proposed proxy does implement a reverse proxy model, the web application is written as a stand-alone application on top of the HTTP-CoAP proxy. As

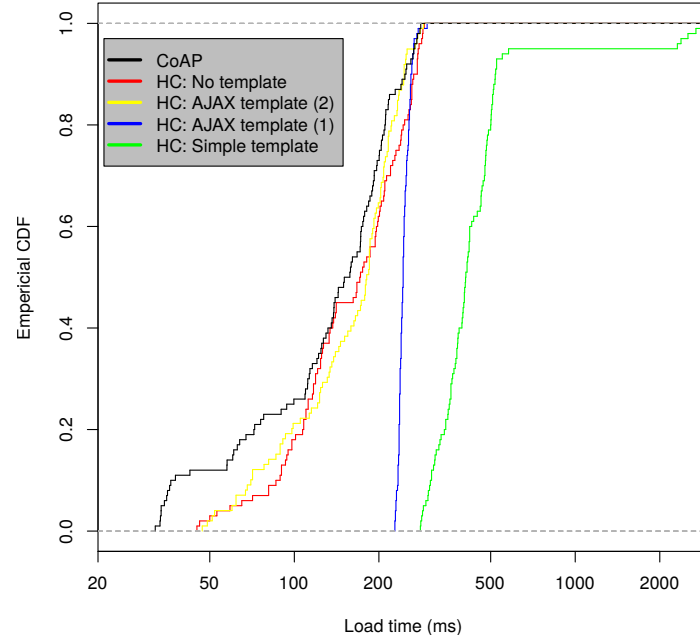


Figure 4.7: CDFs of load times for different proxy configurations

such, the approach differs from ours where the template rendering is an integral part of the cross protocol proxy. In [12] Jin et al. present a CoAP service gateway for automatically creating service mash-ups based on semantic similarity between related CoAP servers. While CoAP SG includes a HTTP-CoAP proxy that can return plain-text or JSON HTTP responses, the focus is on aggregating data from multiple CoAP servers rather than on generating user interfaces. Nevertheless, the work does prove that proxies on gateways are valuable for implementing extra functionality.

4.6 Conclusion

This paper presented a number of methods for improving user interactions with constrained devices. Essential in implementing these methods is the presented HTTP-CoAP proxy, which renders user interfaces for RESTful resources of constrained devices based on web templates. The paper demonstrated this concept by means of two templates for constrained device resources and one template for discovery of devices and resources. Additionally, the interface responsiveness and the delay of our approach was also quantified.

In the future, the authors plan to extend the concept to include transport layer security (i.e. HTTPS-CoAPs proxy) and to support other functionality than user interface generation (e.g. data aggregation, data format rewriting).

Acknowledgment

The research from the DEWI project (www.dewi-project.eu) leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n°621353 and from the agency for Flanders Innovation & Entrepreneurship (VLAIO). The research from the ITEA2 FUSE-IT project (13023) leading to these results has received funding from the agency for Flanders Innovation & Entrepreneurship (VLAIO).

References

- [1] D. Evans. *The internet of things: how the next evolution of the internet is changing everything*, 2011. Available from: <https://www.cisco.com/c/dam/en{ }us/about/ac79/docs/innov/IoT{ }IBSG{ }0411FINAL.pdf>.
- [2] Ericsson Inc. *More than 50 Billion Connected Devices - Taking connected devices to mass market and profitability*, 2011. Available from: <http://vdna.be/publications/Wp-50-Billions.Pdf>, doi:284 23-3149 Uen.
- [3] P. Middleton, T. Tully, J. F. Hines, T. Koslowski, B. Tratz-Ryan, K. F. Brant, E. Goodness, A. McIntyre, and A. Gupta. *Forecast: Internet of Things - Endpoints and Associated Services, Worldwide, 2015*, 2015. Available from: <https://www.gartner.com/doc/3159717/forecast-internet-things--endpoints>.
- [4] A. Asín and D. Gascón. *50 Sensor Applications for a Smarter World: Libelium white paper*. Libelium, 2012. Available from: <http://www.libelium.com/top{ }50{ }iot{ }sensor{ }applications{ }ranking/>.
- [5] C. Bormann, M. Ersue, and A. Keranen. *RFC 7228: Terminology for Constrained-Node Networks*. Technical report, IETF, 2014. Available from: <http://tools.ietf.org/html/rfc7228>.
- [6] I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. Van den Abeele, E. De Poorter, I. Moerman, and P. Demeester. *IETF standardization in the field of the Internet of Things (IoT): a survey*. *Journal of Sensor and Actuator Networks*, 2(2):235–287, 2013.
- [7] Z. Shelby, M. Koster, C. Bormann, and P. van der Stok. *CoRE Resource Directory*, 2016. Available from: <https://tools.ietf.org/html/draft-ietf-core-resource-directory-07>.
- [8] A. P. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk. *Guidelines for HTTP-to-CoAP Mapping Implementations*, 2016. Available from: <https://tools.ietf.org/html/draft-ietf-core-http-mapping-10>.
- [9] Z. Shelby. *RFC 6690: Constrained RESTful Environments (CoRE) Link Format*, 2012. Available from: <https://tools.ietf.org/html/rfc6690>.
- [10] A. Ludovici and A. Calveras. *A Proxy Design to Leverage the Interconnection of CoAP Wireless Sensor Networks with Web Applications*. *Sensors*, 15(1):1217–1244, jan 2015. Available from: <http://www.mdpi.com/1424-8220/15/1/1217>, doi:10.3390/s150101217.
- [11] W. Colitti, K. Steenhaut, N. D. Caro, B. Buta, and V. Dobrota. *REST Enabled Wireless Sensor Networks for Seamless Integration with Web Applications*. In 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, pages 867–872, 2011. doi:10.1109/MASS.2011.102.

-
- [12] X. Jin, K. Hur, S. Chun, M. Kim, and K. H. Lee. *Automated mashup of CoAP services on the Internet of Things*. IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings, pages 262–267, 2016. doi:10.1109/WF-IoT.2015.7389063.

5

Integration of heterogeneous devices and communication models via the Cloud in the constrained Internet of Things

While the previous chapter focused on improving the usability from an end user's perspective, this chapter looks at increasing the usability of heterogeneous Internet of Things (IoT) technologies from the point of view of a service developer wanting to combine diverse IoT technologies. This chapter presents a cloud-based platform to facilitate the integration of heterogeneous constrained IoT devices and communication models into services by means of a uniform, open standards-based device abstraction. A three-fold evaluation demonstrates that the platform improves latency, is effective at hiding heterogeneous communication modules and is straightforward to integrate into services developed by third parties.

**Floris Van den Abeele, Jeroen Hoebeke, Ingrid Moerman and
Piet Demeester**

**Published in International Journal of Distributed Sensor Networks Volume 11
Issue 10, October 2015, special issue on “Leveraging the Internet of Things:
Integration of Sensors and Cloud Computing Systems”.**

Abstract As the Internet of Things continues to expand in the coming years, the need for services that span multiple IoT application domains will continue to increase in order to realize the efficiency gains promised by the IoT. Today however, service developers looking to add value on top of existing IoT systems are faced with very heterogeneous devices and systems. These systems implement a wide variety of network connectivity options, protocols (proprietary or standards-based) and communication methods all of which are unknown to a service developer that is new to the IoT. Even within one IoT standard, a device typically has multiple options for communicating with others. In order to alleviate service developers from these concerns, this paper presents a cloud-based platform for integrating heterogeneous constrained IoT devices and communication models into services. Our evaluation shows that the impact of our approach on the operation of constrained devices is minimal while providing a tangible benefit in service integration of low-resource IoT devices. A proof of concept demonstrates the latter by means of a control and management dashboard for constrained devices that was implemented on top of the presented platform. The results of our work enable service developers to more easily implement and deploy services that span a wide variety of IoT application domains.

5.1 Introduction

In the coming years more and more everyday objects are expected to be interconnected to the Internet, which will lead to a vast expansion of the Internet as we know it today. White papers released by Ericsson [1] and Cisco [2] estimate that the Internet will grow tenfold in the near future, with up to 50 billion connected devices by 2020. A considerable amount of these new Internet citizens will be so-called constrained devices. These are small, embedded and low-cost devices that are purposefully designed for executing specific tasks such as monitoring the physical environment. For performing their tasks, they are often fitted with a microcontroller, sensors, actuators, a wireless transceiver and an energy source. Due to their low cost, these devices are constrained in terms of processing power, communication capabilities and energy budget. The widespread deployment and use of such constrained devices across a range of application domains (e.g. smart city, building control, logistics, transportation, etc.) is expected to generate significant efficiency gains as well as drive new business. The combined effect of which is estimated to create 14.4 trillion USD in net value in the next decade [3].

Due to the IoT spanning such a wide range of application domains, there is a diversity of devices, protocols, network connectivity methods and resulting application models on the market today. Within these IoT solutions some are legacy systems that rely on proprietary technology (sometimes suitably referred to as the Intranet of Things [4]), while others adopt more open IoT standards such as MQTT [5] and the IETF protocol stack for (constrained) IoT devices [6]. This diversity often results in the vertical silos seen today and hinders development of value-added services that use these low-resource IoT devices [7]. For example,

in [8] the authors state that the logistics sector should move away from proprietary, stand-alone solutions that are not connected to the rest of the IoT to new platforms that combine various existing hardware and software solutions for end-to-end integrity control of supply chains. But even within one (standardized) protocol suite there are a number of different communication and application design strategies available which are often tightly tied to the underlying use case. For example in logistics, separate communication strategies are necessary for battery-powered tracker devices (which are typically only intermittently connected to the Internet) and trackers with an abundant energy source (which can afford network connectivity for longer periods of time). Forcing IoT users (e.g. service developers, constrained devices) to support these different types of diversity, is unfeasible as they typically lack the proper resources (e.g. know-how, time, processing resources) to handle the specifics of the underlying constrained devices and networks. The goal of this work is to hide this wide range of diverse technologies, protocols and applications models from IoT users.

As the demand for low-resource IoT devices is expected to rise, this problem is only expected to worsen in the future. Thus, it will become necessary to improve the integration of a wide variety of constrained devices in the IoT. Cloud computing is a suitable method, due to its availability, elasticity (improving scalability) and low-cost of computing resources [9]. Such a cloud-based software system is interesting because it can support different types of low-resource IoT devices by means of an adaptation layer and offer a uniform device abstraction that hides the diversity in devices, protocols, network connectivity methods and application models from IoT users. By offering well-known interfaces via open standards protocols (RESTful and CoAP in this work) for this device abstraction, the later becomes significantly easier to integrate than the underlying constrained devices. The resulting design greatly improves integration of and service development for constrained IoT devices, while neither burdening the constrained IoT devices nor the service developers.

Our contributions in this paper are as follows. First we design, implement and evaluate a cloud-based software architecture that enables the integration of heterogeneous low-resource devices in the Internet and more importantly into services. By offering a uniform CoAP interface on top of a virtual device abstraction, our approach is able to support highly heterogeneous devices as well as provide interoperability with legacy IoT devices that were built using proprietary technology. Secondly, we illustrate the feasibility of our developed architecture with a real-life deployment consisting out of constrained devices that embrace open standards and one representative example of a proprietary low-resource IoT device. The proof-of-concept demonstrates that the developed architecture supports a number of different communication models.

The remainder of this paper is structured as follows. First, a case study is presented in the following section to highlight some of the issues faced when developing applications on top of two industrial IoT systems. Section 5.3 details the research questions addressed by this paper. The next section introduces supporting technologies and other background information used in the remainder of this pa-

per. Section 5.5 presents our approach for integrating heterogeneity devices in the constrained Internet of Things. Our proposed solution is evaluated in section 5.6 in a wireless sensor network setup and via a real-life proof of concept. Section 5.7 presents the literature related to our work. Finally, the paper is concluded with possible topics for future work and our conclusions in section 5.8.

5.2 Case study: logistics and transport

The case study presented here considers harbor cranes and freight containers owned by different parties. The harbor cranes are tasked with (un)loading containers from cargo ships. Both the cranes and the containers are equipped with a GPS-enabled tracking device, each of which belongs to a different entity (i.e. vendors A and B). The trackers from vendors A and B each report to their separate back-end systems, where the vendor's customers can follow up on the status of their cranes or container's (contents) via a vendor-specific web portal (see figure 5.1). The trackers are connected to the Internet via a GPRS connection.

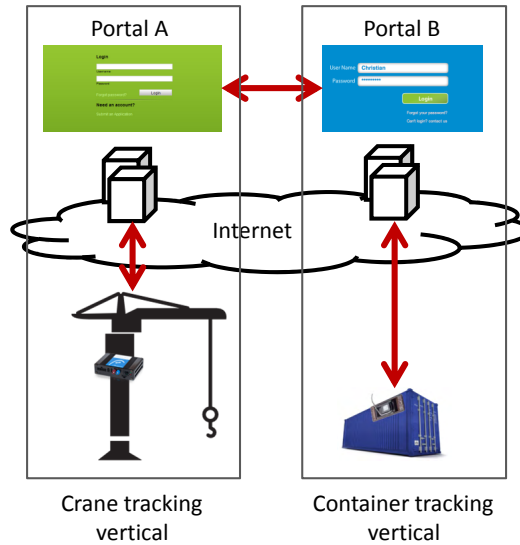


Figure 5.1: Isolated vertical platforms hinder cross-vendor service delivery

Due to the container's mobility its tracker is battery-powered, whereas the crane's tracker is powered via the crane's alternator. Subsequently, the container's tracker is in sleep mode most of the time to conserve energy. In order to conserve the limited energy of the container's tracker even further, vendor B wants to update the location of its container by using the location information supplied by the crane when the container is picked up. This way the container's tracker avoids acquiring a GPS and a GPRS signal and transmitting its position, thus reducing its energy

expenditure. However, both vendors only offer a web portal to their customers and there is no other interface for retrieving data from the trackers. Thus, vendors are forced to either rely on expensive and error-prone human intervention to interface between both systems or to try and build on top of interfaces designed for user interaction.

Situations such as these are common in today's Internet of Things, with its abundance of isolated vertical platforms (e.g. in building management systems as per [10]). The costs of deploying and maintaining all the individual systems in such verticals should not be underestimated as the re-usability of components between verticals is typically low. As vendors start to expand their products into other IoT application domains (each with its own heterogeneous set of properties) and as IoT applications start to span across multiple domains (where everything will interact more and more with each other), this approach of building purpose-specific vertical silos will rapidly become inefficient and expensive.

5.3 Problem statement and research goals

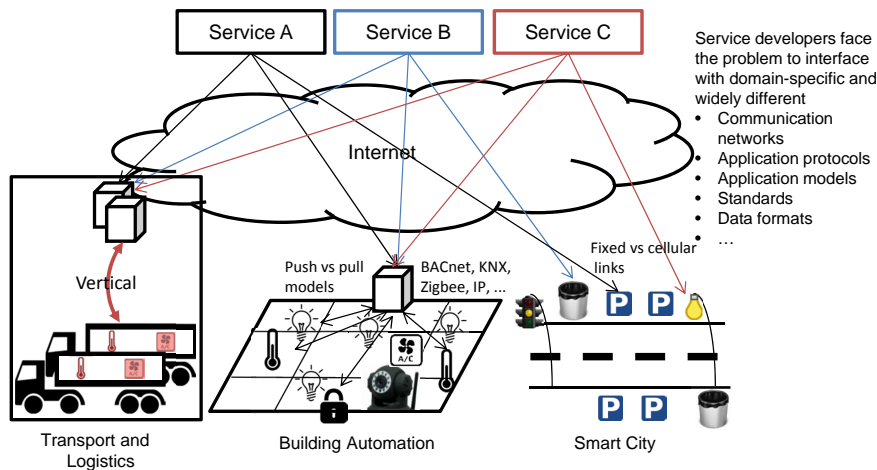


Figure 5.2: Problem statement: As each domain of the IoT comes with its own set of IoT devices, protocols, standards, data formats and connectivity options, service providers are forced to integrate a multitude of different technologies when developing cross-domain services.

Before the problem statement is formulated, a number of examples of heterogeneity are presented here to illustrate the problems addressed by this work. Due to the wide range of environments where the IoT is considered to be employed [11], a number of different network connectivity technologies will be used depending on the specific use case. For some applications, devices will be on a tight energy budget (e.g. battery-powered or energy harvesting devices) which

might mean conserving energy by sleeping and remaining unreachable for long periods of times. In other applications mobility, remoteness or financial cost might lead to devices with an intermittently-connected link to the Internet. And in other cases still, devices might be mains-powered and have a near always-on connection to the Internet. Furthermore, some low-resource devices might not be able to support certain transport protocols (e.g. TCP) and might have to follow a different approach (e.g. UDP or SMS). While in all these cases devices are accessible via a communications link, the properties and behaviour of this link are not always fully comprehended by service developers. Consider as an example a smart freight shipping container. As these containers are transported (often over long journeys), their Internet connectivity will vary: e.g. there will be times when they are unreachable and when they are reachable, their Internet end point will change frequently due to their mobility.

Another cause of heterogeneity is the application communication model chosen by low-resource IoT devices. Note that this choice is often influenced by the available network connectivity as discussed in the previous paragraph. Some devices might rely on a "pull model", where the service is expected to initiate all interactions to and from the devices. Other devices, due to network constraints, might employ a "push" approach where devices periodically send data to a data store and sleep for the majority of the time. In this case, push data is often aggregated in order to increase the communication efficiency. In other cases, devices might employ a mix of push and pull: for example critical events and monitoring data is sent immediately (e.g. for generating alerts), non-critical measurements are aggregated and pushed together and configuration and management of the devices takes places via a pull model (i.e. a management service might send configuration data to the IoT device directly).

Another fundamental cause of diversity is whether the IoT device employs proprietary or one of the many standards-compliant application protocols and data formats available today. While proprietary purpose-built technology will in some cases outperform standards-compliant technology, it also has a number of downsides: of which the most important for the discussion here is that they are difficult to integrate for third parties. Furthermore, proprietary protocols often lead to higher development and maintenance costs and are less future-proof than open standards. For our smart freight container example, this is also what we see on today's market [8]. There are a number of players that offer tracking services for freight containers, but their systems are using private communication networks, in-house data standards and typically only offer high-level access to data (e.g. via a web-based dashboard for tracking). This greatly impedes other players to integrate smart containers into their products. However, even when relying on standards-compliant technology, there are many different standards available today. Most of these are popular in specific domains, such as BACnet in building automation, Zig-Bee in home automation and SEP2.0 in smart metering. Thus, services that want to encompass multiple application domains are forced to interface with a mix of proprietary and domain-specific standards, technologies and data formats.

A final source of heterogeneity that is addressed by this work, which is re-

lated to the diversity of application protocols and data formats, is the wide variety of supporting services for interacting with constrained devices. Every technology comes with its own mechanisms for device and data source discovery, its own security concepts and methods (if available at all), its own notification service, etc. As a result there is little reuse between these technologies, which further impedes cross-technology integrations. Moving to open standards will allow using standard-compliant approaches for such supporting services and will provide uniform mechanisms for these services to third party service developers. While the list of heterogeneity presented here is not meant to be exhaustive it does present a clear overview of the types of heterogeneity that this paper considers. A comprehensive overview of the different types of heterogeneity commonly found in wireless sensor networks is presented in [12].

Parties looking to offer new services in this mix of connectivity, application models, standards and protocols will quickly find themselves forced to integrate a multitude of different technologies. Given the previous paragraphs and their conceptual representation in figure 5.2, it becomes clear that supporting all these different types of heterogeneity cannot be expected from service developers. Furthermore, constrained devices are unable to adapt to the different service providers due to their low resources. These devices will typically implement one of many instances of heterogeneous technologies presented here, depending on availability of resources, communication, energy and application requirements. While cloud-based platforms are a good match for complementing constrained low-resource devices [9], a gap analysis of existing IoT platforms [13] shows that support for heterogeneous and constrained devices in such platforms is still lacking. One of the problems identified by Mineraud et al. [13] is that most IoT platforms assume constrained devices to support HTTP, which - given the heterogeneous nature of these devices - is definitely not always the case.

In this paper our goal is to tackle the heterogeneity presented here by focusing on a standards-compatible cloud-based software system that can integrate with a multitude of different technologies and protocols with a focus on low-resource IoT devices. Our research aims to answer the following questions:

- How can standards-based IoT technology be combined with cloud computing to handle the diversity of underlying low-resource IoT devices?
- How can such a cloud computing approach present a uniform interface to third party developers given the different sources of heterogeneity?
- What types of diversity in protocols and communication models can such a cloud-based approach handle?
- What is the impact of this fusion of cloud and IoT on the communication with low-resource IoT devices? Can this communication be made more efficient?
- How can we further exploit the power of cloud computing to support interactions from third parties with IoT devices?

5.4 Background: Embedded web services via CoAP

The diverse environments in which IoT devices have to operate, has led to a mix of proprietary and standard-based protocols and different application models that are deployed in today's Internet of Things. While there are a number of standards relevant for the Internet of Things [14], this paper will build on the standardization as per the IETF protocol stack for constrained devices [6] [15] and more specifically on the embedded RESTful approach followed by the Constrained Application Protocol (CoAP) [16] [17]. CoAP was chosen because it is lightweight yet powerful protocol that is an ideal candidate for integrating constrained devices into the cloud.

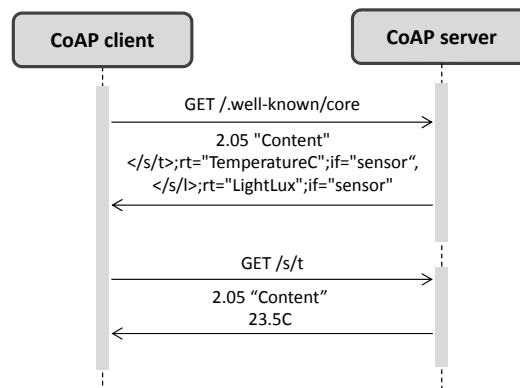


Figure 5.3: A typical request/response exchange between a CoAP client and server

CoAP is a specialized web transfer protocol for use with constrained devices and networks. In CoAP, every physical object (i.e. thing) hosts multiple resources that represent data gathered from sensors or actions available to actuators. Every resource is accessible via a unique uniform resource identifier (URI) and can be interacted with via the GET, PUT, POST and DELETE REST methods. CoAP can be considered as a highly optimized version of HTTP/1.1 for use in the low-resource embedded domain. Main differences with HTTP include the use of connectionless UDP, support for multicast-based group communication, built-in discovery support, simplified header parsing and a publish/subscribe extension [18]. Daniel et al. present a detailed comparison between CoAP, HTTP and SPDY in [19].

A typical CoAP exchange is shown in figure 5.3. The CoAP RFC specifies the `.well-known/core` resource as the entry point for resource discovery. In this example the CoAP server responds that it hosts a temperature and light intensity resource. The server then responds to the client's temperature resource request. CoAP requests and responses can be sent in Confirmable (CON) and Non-confirmable (NON) CoAP messages. As the name suggests CON messages expect the receiver to acknowledge the reception of the message via an acknowledgment (ACK). Most of the time, the response to the client's request is piggy-backed

on top of this ACK.

As mentioned, CoAP provides a publish/subscribe extension in the form of the CoAP Observe mechanism [18]. When a client is observing a resource, the server promises to send new representations of the resource to the client following a best-effort strategy. This frees the client from having to explicitly poll the resource for changes. As observe notifications are regular CoAP responses with the Observe option set, they can be sent as CON and NON messages.

In the CoAP ecosystem there are a number of other works relevant to our discussion here. A CoRE Resource Directory [20] facilitates the discovery of CoAP devices and resources in cases where direct discovery is not practical due to sleeping nodes, disperse networks or networks where multicast is inefficient. To this end, a resource directory hosts descriptions of resources that are available on other CoAP servers. Clients can perform lookups within a resource directory via the web interface specified in the IETF Internet draft.

A second relevant CoAP mechanism is that of a CoRE Mirror Server [21]. Such a server mirrors the resources of a constrained devices, thereby enabling these devices to go into sleep mode and to disconnect their network link in order to save resources. [21] defines the web interfaces for registering resources, sending resource updates, querying for mirrored resources and retrieving updates of mirrored resources. A mirror server can also be extended to mirror resources on behalf of mobile devices, which frequently change their Internet endpoint due to their mobility. An example of a mirror server is shown in figure 5.4.

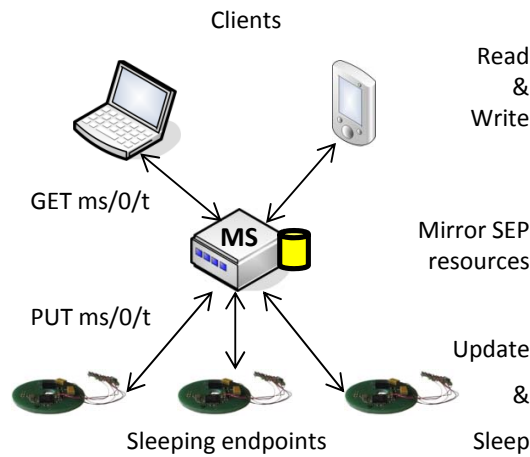


Figure 5.4: CoAP mirror server: clients and sleeping endpoints can communicate in an asynchronous fashion

Sleeping endpoint (bottom of the figure) start by registering their to-be mirrored resources with the mirror server via a POST request (not shown in the figure). From then on, sleeping endpoints (bottom of the figure) operate as CoAP clients. They update their resources on the mirror server via CoAP PUT requests and can

query the mirror server for updates to their resources via a CoAP POST request. After a sleeping endpoint has retrieved a list of changed resources (via the POST request), it can choose to process the changes by retrieving every updated resource via a CoAP GET request. The “Client Operation” interface of the mirror server enables clients (top of the figure) to retrieve and change the mirrored resources. Note that clients must know and implement this Client operation interface.

5.5 Cloud platform for supporting heterogeneous devices and communication models

A high-level overview of our approach is presented in figure 5.5. Low-resource IoT devices with various forms of heterogeneity are shown at the bottom. These employ diverse hardware, protocols and communication models. For each of these devices, our cloud-based platform hosts a virtual counterpart that is made available as a dedicated IPv6 endpoint (i.e. the virtual device). Clients interact only with the virtual device and the cloud takes care of mapping these interactions to the particular constrained device. In our approach the dedicated IPv6 endpoint hosts both a CoAP and a DTLS server and therefore all interactions between a client and a virtual device run over CoAP.

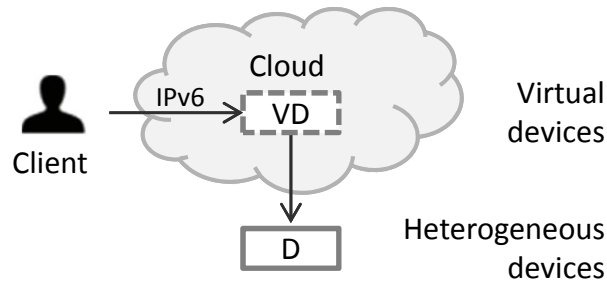


Figure 5.5: Virtual devices in the cloud represent their real heterogeneous counterparts

In effect, our approach follows the Sensor as a Service (SenaaS) paradigm where clients interact with a virtual cloud-based counterpart of a real-life sensor or device. The main benefit of SenaaS is that a virtual device has significantly more resources at its disposal than its constrained counterpart and is therefore not hindered by the constraints common to low-resource devices. For example, a virtual device is always available whereas a constrained device might be asleep or temporarily unreachable (e.g. due to mobility). Furthermore, by deploying our platform in the cloud it can support on-demand dimensioning of computing resources when the load on the platform fluctuates. This allows to scale efficiently as the size of the deployment grows and avoids under- and over-dimensioning computing capacity. Finally, maintaining the platform’s computing infrastructure is outsourced to a specialized external party in this case.

The use of CoAP leads to a lightweight solution where a virtual device can be used by both conventional services as well as by the low-resource devices themselves. The straightforward mapping between CoAP and HTTP has the benefit that virtual devices can easily be integrated into existing RESTful web services. CoAP also provides built-in support for response caching. The result is a resource-oriented architecture that facilitates the integration of constrained devices into third-party services and applications.

To realize this architecture, we have split our cloud-based platform into two layers. The first layer offers a uniform interface to the underlying constrained devices by means of virtual devices. The second layer handles the heterogeneity in constrained devices and applications models. Both of these layers are presented in the following two subsections.

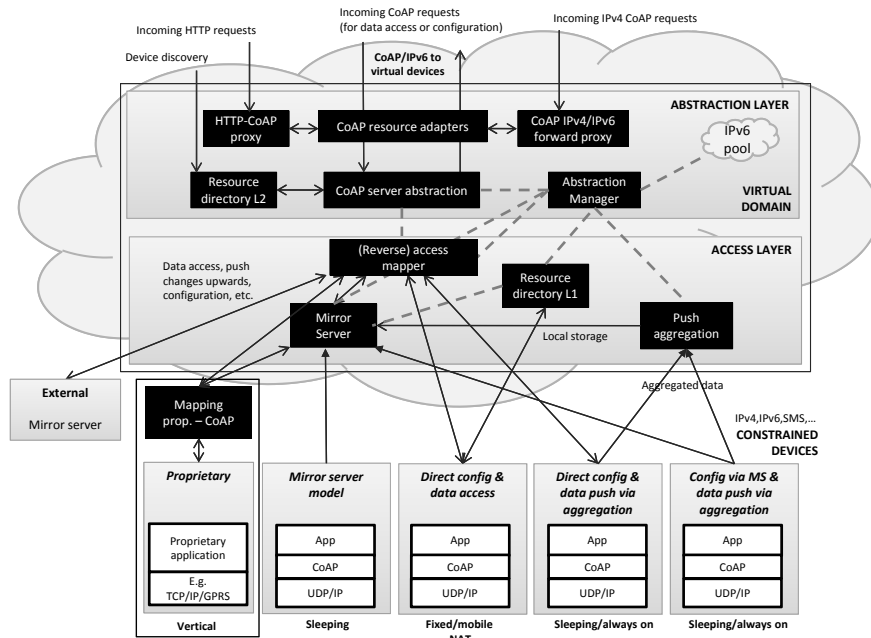


Figure 5.6: The access and abstraction layers of the design enable uniform access to heterogeneous constrained IoT devices.

5.5.1 The access layer: providing access to heterogeneous devices and communication models

The access layer is closest to the constrained devices and is responsible for communication to and from these heterogeneous devices. In some cases the access layer also stores the data provided by the constrained devices. As can be seen from figure 5.6, access is provided to constrained devices that employ a number of

heterogeneous communication models. At the heart of this layer is the access mapper component, this module maps the uniform representation from the abstraction layer to the particular implementation of the constrained device and vice versa. To this end, the access mapper translates requests for CoAP resources hosted by the device abstraction to device-specific actions. In the next few paragraphs we will discuss specific instances of this mapping for the different communication models that are shown at the bottom of figure 5.6.

The first communication model is that of the mirror server model that was already introduced in section 5.4. For constrained devices hosted on mirror servers, the access layer translates requests for virtual devices to requests destined to the corresponding mirror server. It does this by taking the URI path of the incoming request and adding it to the mirror server URI handle of the device. E.g. a request for `coap://[2001:6a8:1d80:600::21]/s/t` on the virtual device is translated to `coap://ms.example.com/ms/4/st` on the mirror server. Requesting the `.well-known/core` discovery resource will trigger a request to the device's handler resource on the mirror server (i.e. `ms/X`). For `.well-known/core` responses, the access mapper will remove all occurrences of the device's mirror server URI-path handle in the response from the mirror server (otherwise clients would see the `s/t` resource as `ms/4/s/t`). Note that the mirror server itself can be running alongside the cloud platform, but that it can also be hosted externally (e.g. on-site for reduced latency).

In the direct configuration and data access model, the constrained device hosts a plain CoAP server. In this case, the access mapper operates as a standard CoAP reverse proxy [16] where requests for the virtual device are mapped to the CoAP server on the constrained device. The constrained device can be a mobile device, where its IP endpoint changes as the device's location changes. Also, a constrained device might be behind NAT and/or a stateful firewall. In both cases, access to the device is typically restricted to those parties with whom the device maintains active communication. In case of NAT, the transport layer mapping at the NAT box is expected to be volatile as well. To this end the access layer allows updating the mapping of a device via the `"/registerendpoint"` CoAP resource. This way the IP endpoint of a device is kept up to date and any state in intermediaries is kept alive. One optimization supported by our reverse proxy is combining multiple CoAP observe relationships into one relationship. If N clients are observing the same resource on the virtual device, then the mapper will maintain only one observe relationship with the constrained device. Consequently the constrained device has to send only one notification instead of N notifications per resource change, thereby reducing its load and energy consumption.

Communication models three and four are hybrid models where data (i.e. originating from the device) and configuration changes (i.e. destined to the device) are handled via different methods. Both models differ from the mirror server model in that they push data for multiple resources in a single request as opposed to pushing data for a single resource per request (as is the case in the mirror server model). The push aggregation module in the access layer is responsible for splitting the incoming aggregated data into multiple requests (i.e. one per resource) to the mirror server. This can lead to considerable energy savings for the constrained device as

the total number of requests is reduced. The difference between these two models, is the method they use for configuration changes. One model allows hosting a CoAP server that enables direct configuration changes, whereas the other model relies on a mirror server for configuration.

For the proprietary communication model there are a number of different mapping strategies available, the exact choice of which will be highly dependent the proprietary technology that is being mapped. There are however two straightforward ways to provide a mapping to constrained devices that are operating inside a vertical. In the first method the constrained devices in a vertical are represented as CoAP clients, while in the second they are represented as CoAP servers. In both cases the vertical defines its own set of resources that provide a suitable RESTful interface for interacting with the proprietary constrained device. In case of the CoAP client option, the vertical can register its devices on a mirror server and all data and configuration changes are handled via the mirror server. The evaluation section presents an example of this approach for a container tracking vertical. In the CoAP server case, the vertical can sometimes act as a cross-protocol proxy.

The final component is the access layer resource directory where mirror servers and direct access CoAP servers register themselves so that they can be discovered by the abstraction manager. The abstraction layer uses this information to instantiate the CoAP server abstractions (i.e. one per constrained device) and the corresponding access mapper instances. The next subsection looks at how this is realized in the abstraction layer.

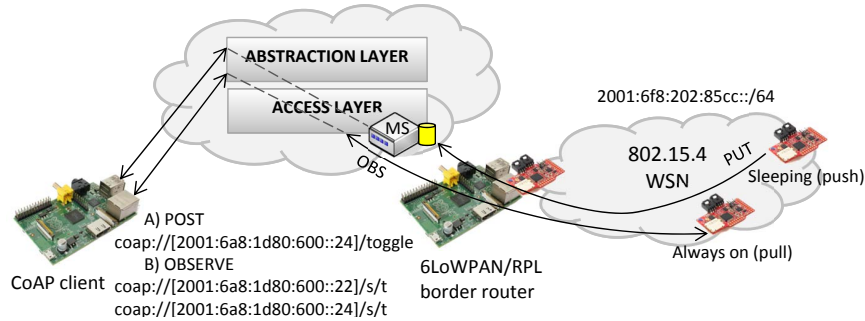


Figure 5.7: Two Raspberry Pi's and an 802.15.4 wireless sensor network operating 6LoWPAN form the evaluation setup for our cloud platform

5.5.2 The abstraction layer: a homogeneous RESTful interface for constrained devices

The abstraction layer is responsible for providing a homogeneous interface to devices that implement the different communication models mentioned in the previous paragraphs. Furthermore, it also allows to extend this device abstraction with new functionality. Finally, the layer provides proxy services to IPv4 and HTTP for

broadening the interfacing possibilities with the device abstraction. The layer is part of the virtual domain, where also the device abstractions reside.

In term of the device abstraction, we chose to represent every constrained device as a virtual device that implements a CoAP server with one or more resources. This virtual device is hosted as a dedicated IPv6 endpoint by allocating an IPv6 address from an IPv6 subnet that is routed to the cloud. While this abstraction is hosted at the network layer, our abstraction layer will only process CoAP (and DTLS) traffic for virtual devices, other types of traffic are not processed. One benefit of using a network-layer abstraction, is that multiple IPv6 subnets and IPv6 routing can be used for distributing device abstractions over a number of different cloud systems in order to improve scalability. The result is that every constrained device is made available as an open standards-compliant (virtual) CoAP server regardless of the underlying communication model and protocols of the device. To this end, the L2 resource directory contains web links to all virtual devices and their resources.

On top of the server abstraction is a block that enables to extend the functionality provided by the server abstraction on both the transport layer and the application layer (i.e. on the level of CoAP resources). The “CoAP resource adapters” module allows a managing party to instantiate chains of plugin-like functional blocks that extend the server abstraction in a desired way. This concept has been presented in previous work [22] and has been shown to significantly reduce the communication overhead of DTLS in IP-based wireless sensor networks [23]. In this work we reuse the concept of adapter chains for realizing the security model of our platform as explained in the next paragraph. It is also used to implement a cache for CoAP responses, which is running on top of memcached.

In terms of the security model, the cloud platform is considered as a trusted entity by constrained devices. Policies are defined in the cloud platform to delegate (parts of) this trust to clients. Clients of a virtual device are authenticated by the credentials they provide during the DTLS handshake. Currently the platform supports pre-shared keys (useful for constrained clients), raw public keys and standard X.509 certificates issued by a party trusted by the policy (see later). The server abstraction is authenticated to the client via the same types of credentials. The resource adapters block provides a DTLS adapter type for authenticating clients and handling the DTLS protocol, as per [23]. It also provides an adapter type that handles authorization. This is accomplished by allowing administrators to define policies based on the credentials of the user, the destination of the request and the desired operation (i.e. CoAP method and targeted CoAP resource). The authorization adapter processes the output of the DTLS adapter (i.e. plain text CoAP requests) and drops all requests that do not adhere to the defined policy.

The final two components enable access to the IPv6 CoAP server abstraction from HTTP clients and from IPv4-only CoAP clients. As mentioned CoAP is designed to interface easily with HTTP, so mapping HTTP and CoAP messages is a straightforward process for the HTTP/CoAP proxy. Furthermore, our device abstraction enables us to host a HTTP server using the IPv6 address of the virtual device. Therefore, the URI-mapping mechanism is very simple (i.e. change the

scheme from http to coap) and the URI-mapping issues raised by [24] are not applicable in this case. Finally, our proof of concept deployment learned that not all clients have IPv6 Internet access. More specifically, GPRS-based Internet access is often restricted to the IPv4 Internet. To address this issue, the platform provides an IPv4/IPv6 forward CoAP proxy where the target (IPv6) CoAP URI is encoded in a “Proxy-URI” CoAP option [16].

5.5.3 Machine to Machine communications

An important application for IoT platforms is machine to machine communications. One common issue in M2M is device management: i.e. tracking and configuring devices that are deployed in the field. In 2014, the Open Mobile Alliance (OMA) has defined a set of standards for device management in M2M: Lightweight M2M¹. LWM2M has adopted CoAP and its RESTful mechanisms as the protocol of choice for interfacing with devices. They also mandate the use of DTLS for security and a resource directory for discovery of devices. While the platform presented here doesn’t implement the standardized OMA interfaces (i.e. CoAP resources) for bootstrapping, management and services, adding them would be straightforward as both our platform and LWM2M share the same building blocks.

One typical characteristic of M2M systems is the use of SMS services for communications, as opposed to IP-based communications. The CoAP protocol was designed with a range of low-power and lossy network types in mind [25] and an adaptation of CoAP to SMS transport is presented in [26]. Thus, the same application protocol can be used for interfacing with both SMS and IP-based constrained devices. Furthermore, the design of our platform can easily be extended with SMS functionality in both the abstraction and the access layers. Virtual devices could be allocated a unique mobile number, which would make them accessible to machines that are restricted to SMS communications. Finally, adding an SMS gateway in the access layer would allow virtual devices to map to SMS-only constrained devices.

5.6 Evaluation

We have evaluated our platform via three methods, two of which provide a quantitative evaluation about specific aspects of the platform while the third is a qualitative evaluation in the form of a proof of concept.

The setup for the quantitative evaluations is shown in figure 5.7. The client is a Raspberry Pi model B connected natively to the IPv6 Internet via a commercial Belgian ISP. The cloud platform is running on a virtual machine hosted at our University’s data center, where Internet peering is provided by Belnet². The wireless sensor network consists out of three Zolertia Z1 nodes, which are equipped

¹<http://openmobilealliance.org/about-oma/work-program/m2m-enablers/>

²Belnet is a Belgian Internet provider for educational institutions, research centers, scientific institutes and government services

with TI's 16 bit msp430 microcontrollers (8 KB RAM and 128 KB ROM) and a CC2420 802.15.4 radio. According to IETF terminology, these devices fall under the "Class 1" category [27]. These are devices that are constrained in code space and processing capabilities, in that they cannot easily communicate with other Internet nodes employing a full protocol stack such as using HTTP, TLS and related security protocols and XML-based data representations. However, they have enough power to use a protocol stack specifically designed for constrained nodes (such as CoAP over UDP) and participate in meaningful interactions without the help of a gateway node.

All Zolertia nodes are running the IETF stack for constrained devices available in contiki. The two sensor nodes on the right of the figure are battery-powered and employ the ContikiMAC MAC and Radio Duty Cycling protocol. The channel check frequency of ContikiMAC was lowered to 4 Hz to conserve energy. Routing inside the 6LoWPAN network is enabled by the RPL routing protocol. One Zolertia node is configured with the direct access model (pull) and one with the mirror server model (push). The third sensor node acts as a 6LoWPAN and RPL border router and is connected to a Raspberry Pi model 2. This Raspberry Pi is connected to the IPv6 Internet via a SixXS.net tunnel (on the Belgian Easynet PoP). The 2001:6f8:202:85cc::/64 subnet is routed over this tunnel and distributed inside the 6LoWPAN network.

5.6.1 Virtual device abstraction: scalability and latency

While routing all traffic to the cloud-based virtual device has many benefits (the most important being availability of computing resources), it may also introduce some undesirable downsides. One obvious issue is that of scalability. As the number of IoT devices continues to increase in the coming years, the volume and velocity of IoT traffic is expected to grow exponentially [28]. However, this issue can be addressed by relying on the flexibility provided by our network layer abstraction in terms of allocating IPv6 addresses to virtual devices and in terms of configuring routing tables to distribute traffic to the machines running our cloud platform. Via dynamic routing tables we can move virtual devices from machines that experience a high load, to newly allocated computing resources. The reverse mechanism allows to scale down when the load decreases, thus realizing the resource elasticity common to cloud computing. However, this is still future work.

Another potential problem is latency: i.e. how does routing traffic via the virtual device abstraction impact latency between a client and a constrained device? In a lot of cases our approach can actually provide latency improvements by e.g. serving content from a cache. However, not all requests can be satisfied from the cache, activating an actuator is a prime example of this. What is the impact on latency in this case? In order to quantify this impact, we have conducted response time measurements using the setup from figure 5.7. Confirmable CoAP POST requests, that toggle a LED on the Zolertia sensor node, were sent for measuring the round trip time (RTT) from client to constrained device. Only the always-on node (with RDC) was used in this experiment. Over 8000 CoAP requests were

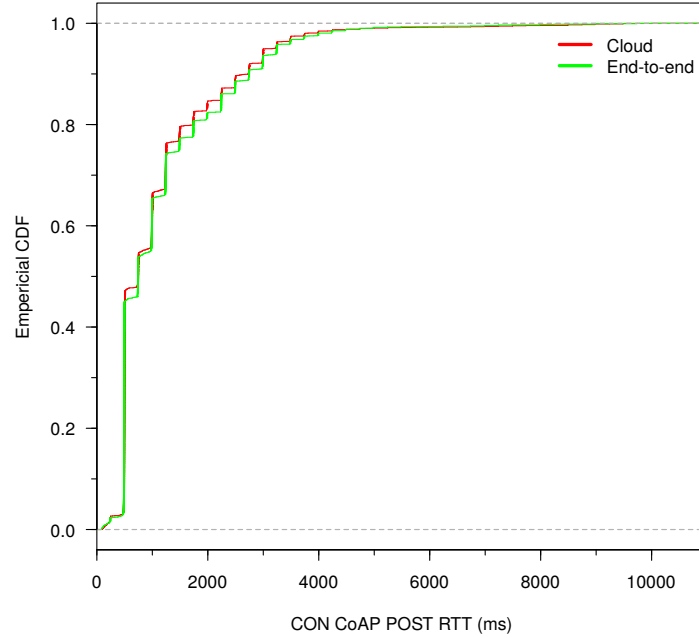


Figure 5.8: Cumulative distribution function of response times for confirmable CoAP POST requests

sent sequentially for the following two configurations:

1. Cloud: the client sends the POST request to the virtual device in the cloud, where it is translated to a POST request to the corresponding constrained device. The response follows the reverse path.
2. End-to-end: the client sends the POST request to the constrained device directly (not shown in figure 5.7).

A CDF of the response times is shown in figure 5.8. We can see that the differences between the two configurations are small and negligible for most use cases. This is because most of the response time is spent on sending the request over the last wireless hop from the border router to the sensor node (due to the RDC). For the end-to-end configuration, the average RTT between the client and the 6LoWPAN router was only 29.8 ms (with a standard deviation σ of 5.6 ms). For the cloud configuration, the average RTT between client and cloud was 22.5 ms ($\sigma = 5.1$ ms) and between cloud and 6LoWPAN router was 17.3 ms ($\sigma = 1.5$ ms). For the end-to-end configuration the minimum CoAP POST RTT measured was 74.7 ms, for the cloud configuration it was 89.7 ms; i.e. a difference of 15.0 ms. However, considering the fact that for the cloud configuration more than 90% of the measured response times were longer than 482.0 ms (and 99.9% were longer

than 100 ms), we can see that the impact of the lossy 802.15.4 network in combination with radio duty cycling on latency is much higher than hosting our virtual device in the cloud.

5.6.2 Communication models: push vs pull

In this subsection the mirror server and direct access communication models, introduced in the previous section, are compared in terms of energy consumption. For these experiments the setup from figure 5.7 is also used. There are two configurations for each of the models: one where the data communication period is 10 seconds and one where this period is 30 seconds. In all experiments the Raspberry Pi CoAP client observes a temperature resource on the virtual CoAP server. The abstraction of the sleeping device is hosted at 2001:6a8:1d80:600::22, whereas 2001:6a8:1d80:600::24 corresponds to the direct access device. When the sleeping device is not engaged in an active CoAP exchange it switches its radio into sleep mode until its next transmission, otherwise it employs RDC. The always-on device continuously employs RDC.

When the CoAP client observes the temperature resource on the virtual device, the cloud platform will observe the corresponding resource on either the mirror server or the constrained device itself. The sleeping device is configured to update the mirror server every 10 or 30 seconds via CoAP PUT requests (push model). Every twentieth PUT request is a confirmable CoAP message, whereas the other requests are non-confirmable messages with the CoAP No-Response option³. The always-on device is configured to update its resource every 10 or 30 seconds and to send one confirmable notification for every twenty notifications (pull model). The other nineteen notifications are sent as non-confirmable messages⁴. The resulting four configurations are named PUSH10, PUSH30, PULL10 and PULL30 respectively. Every time the client receives a notification it estimates the energy consumption of the constrained device for transmitting the data that triggered the notification from Energest data that is retrieved from the constrained device [29]. Energest values are read from a serial connection with the constrained node that is running over a TCP connection to a serial forwarder that is attached to the UART0 line of the sensor node. More than 400 measurements were collected per configuration.

The box plots of total energy usage in figure 5.9 show that in general the push model consumes less energy than the pull model for this experimental setup. This is due to the fact that for the push model the radio can be kept in sleep mode for longer periods of time per transmission. The stacked bar plot confirms this for the “median energy usage” case, where it is clear that the energy spent in the reception category for the pull models is significantly higher than in the push models. Note that the PUSH30 data points are significantly higher than the PUSH10 data points. If we compare the median cases, then we can see that while the median radio

³Responses are suppressed because the client would not be awake to receive them, cfr. <https://tools.ietf.org/html/draft-tcs-coap-no-response-option-10>

⁴This is the default behaviour for CoAP observe in Contiki’s Erbibum

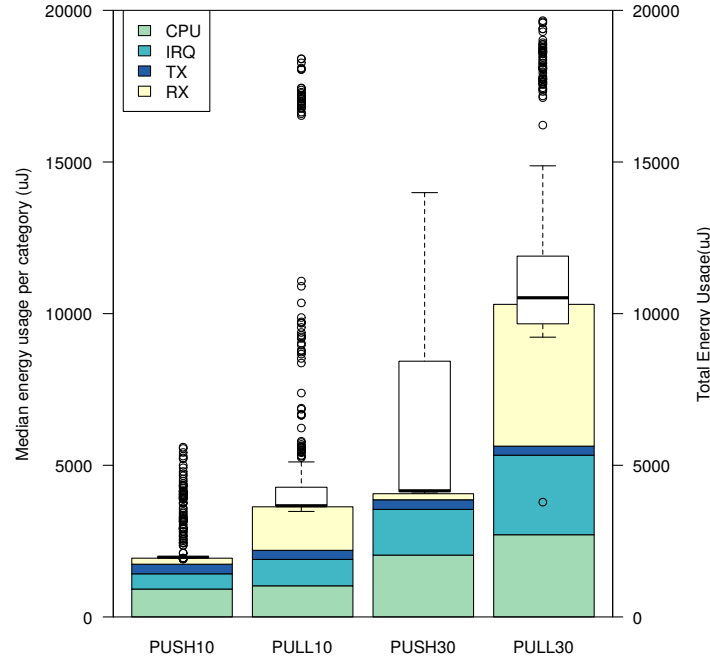


Figure 5.9: Left: stacked bar plot of median energy usage per category. Right: box plot of total energy usage.

energy expenditures are similar (the same amount of data is being sent in both cases) the CPU and IRQ categories are not. This is because the micro controller has to process more timer interrupts per data transmission as the time between transmissions is three times longer. Finally, also note that the two distributions of measurements for the 30 seconds configurations are clearly skewed towards higher energy expenditures (i.e. larger errors bars with larger variation at higher values). This difference is caused by the fact that the total number of sent RPL messages for refreshing upwards and downwards routes increases as the data transmission interval increases. Thus for larger intervals the fraction of energy expenditure data points that include RPL message transmissions will be higher and therefore the box plots will be skewed towards higher values.

Figure 5.10 shows the sum of packets received and transmitted by the constrained node per data transmission period. The median for all configurations is one, which is explained by the fact that nineteen out of twenty data messages are NON messages with no response. Here we can also see that the RPL messages skew the box plots for the PUSH30 and PULL30 scenarios towards the top. Note that the effect of retransmissions of CON messages is hard to see in this plot as they only make up five percent of the data points.

5.6.3 Proof of concept: real world deployment

In the context of a national project⁵, a proof of concept was developed that demonstrates the feasibility of the platform presented in this paper. The project involved three industrial parties that develop monitoring solutions for waste bins, construction cranes and freight containers. Each of these partners has developed their own vertical system, where customers access monitoring data via a web portal that is hosted inside the vertical. The goal of the proof of concept was to show that our platform enables third parties to collect data and configure constrained devices spanning multiple application domains while maintaining the same levels of performance (i.e. code size, energy expenditure and life time) and service of the constrained devices as offered by the existing (proprietary) solutions. Figure 5.11 shows the different components in the proof of concept. At the bottom there are four different types of constrained devices, each fulfilling a monitoring task for a specific use case.

The waste bin trackers consist of a 32 bit ARM Cortex-M3 microchip (Silicon Labs EFM32) and an Analog Devices ADF7021-N narrow-band transceiver that operates at 169 MHz. These trackers run on contiki, implement the IETF stack for constrained devices and form a wireless sensor network via RPL in non-storing mode. The trackers are sleeping devices and employ the mirror server model to

⁵<https://www.iminds.be/en/projects/2014/03/03/comacod>

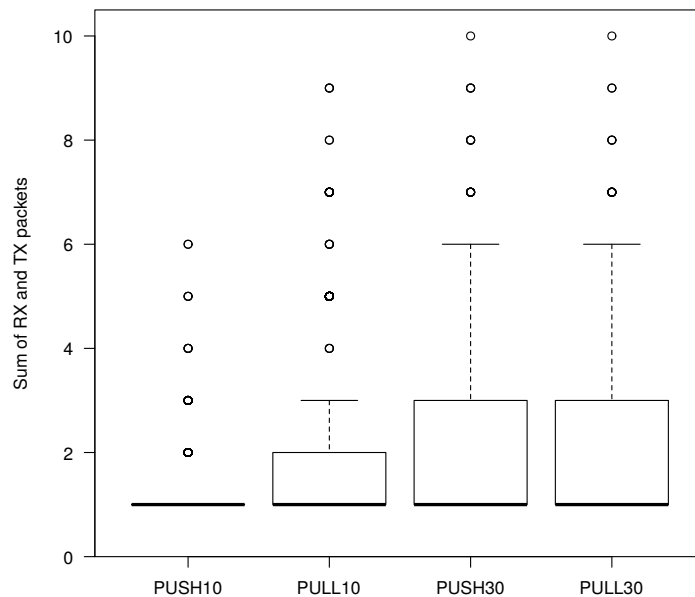


Figure 5.10: Sum of packets received and transmitted for different communication models

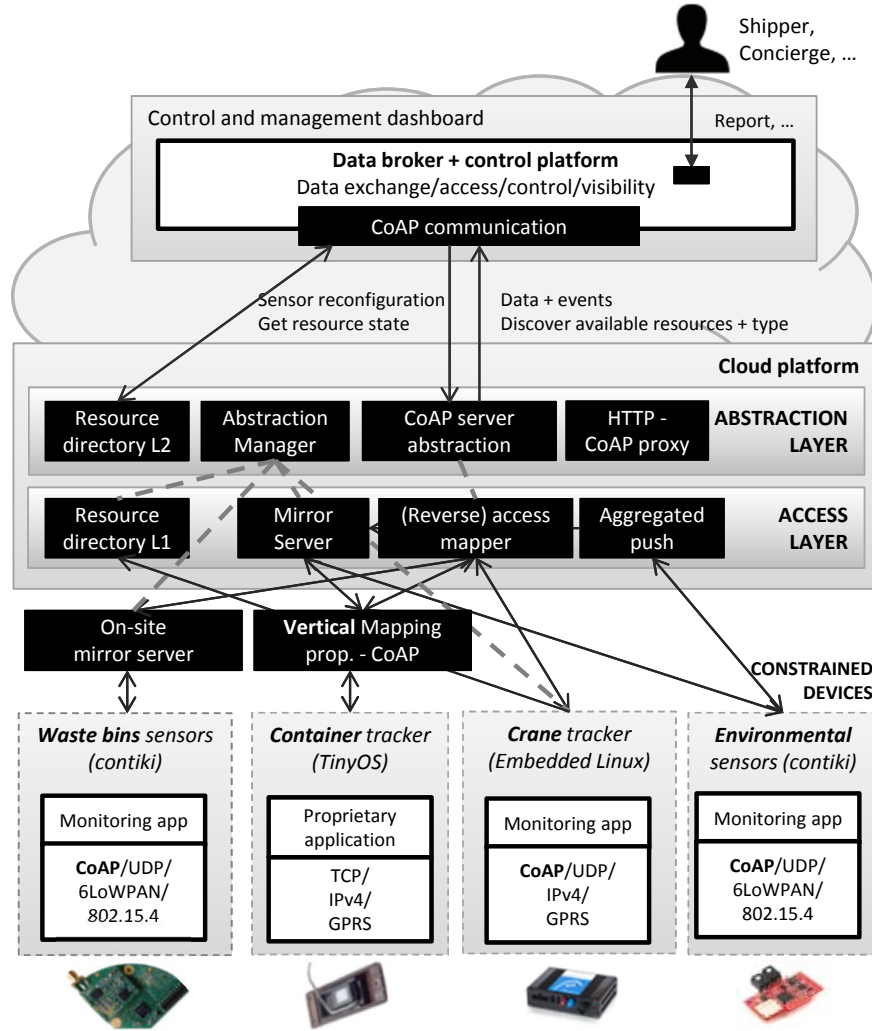


Figure 5.11: Proof of concept: components and setup

send periodical status information (waste bin tray events and battery status) to a mirror server that is running on the 6LoWPAN router (i.e. the on-site mirror server in figure 5.11). The 6LoWPAN router is connected to the private network of the company via a VPN tunnel over the public Internet. By also connecting to this private network from the access layer and offering a virtual device for every waste bin tracker registered on the on-site mirror server, the cloud platform has enabled applications to integrate the waste bins as normal CoAP servers. This is a great improvement over the database with web API integration option which was the only choice offered by the company in the past.

The container tracker is composed of a msp430f5437 microcontroller, a CC2520 802.15.4 transceiver and a Telit HE910 HSPA+ modem with built-in GPS. Containers form an 802.15.4 network that is used for inter-container communication towards a data sink that transmits the data to a back-end system using its HSPA modem. The containers run TinyOS and use a proprietary data format over TCP for communicating with the back-end. In this case, it was not considered viable to change the proprietary stack running on the devices as the development effort was deemed too high. Instead, the container tracker back-end system implements a mapping from the proprietary technology to embedded web services by means of a CoAP mirror server. For every tracker, the back-end system registered an endpoint on the mirror server with the corresponding resources for location, container mode, temperature and humidity. Whenever the back-end receives data from the container trackers, the resources of the corresponding tracker on the mirror server are updated. Whenever the container trackers wake up, the back-end checks the mirror server for changes (e.g. the container's mode resource has changed), retrieves all updates for the mirrored resources and translates these updates to corresponding actions in the proprietary technology. Because of the virtual device abstraction, all trackers are hosted as CoAP servers.

The crane tracker is a powerful embedded Linux device that is running an ARM9 CPU with 128MB of ROM and 64MiB RAM. The trackers incorporate a GPS and GPRS chip and used to establish a VPN tunnel with the back-end. In this private network, a tracker remained reachable at a fixed IP address even when its public IP address changed on the cellular network and when it was behind NAT. Data was exchanged using a proprietary format over TCP. There was no real integration possible as the company only offered a web portal for its customers for following up on their cranes. For this use case, the proprietary technology was dropped and the tracker implements a mobile CoAP server instead. Using our cloud platform, the VPN connection was no longer necessary as the virtual device in the cloud can offer a fixed IPv6 endpoint for the tracker. Whenever the tracker's GPRS IPv4 address changes, it updates the mapping in the access layer via a CoAP POST request to the cloud system. In order to traverse any firewall and NAT systems between the tracker and the cloud, the cloud's access layer uses a UDP source address that is equal to the destination UDP address that was used by the tracker for updating its mapping. As in the previous case, there was no viable integration strategy for third parties (only a web portal was available). By using our cloud platform, trackers are offered as virtual CoAP servers and made discoverable via the resource directory. Figure 5.12 shows the tracker updating its access layer mapping as well as the message exchange triggered by a CoAP request to the /gps/full resource of the corresponding virtual device. The two packets marked in black show that our system uses the destination UDP address of the registration to contact the tracker afterwards.

The final class of devices consists of Zolertia Z1 nodes that were used as the main testing platform and were deployed as per figure 5.7. These devices periodically collect temperature and humidity data using the aggregated push and mirror server communication models.

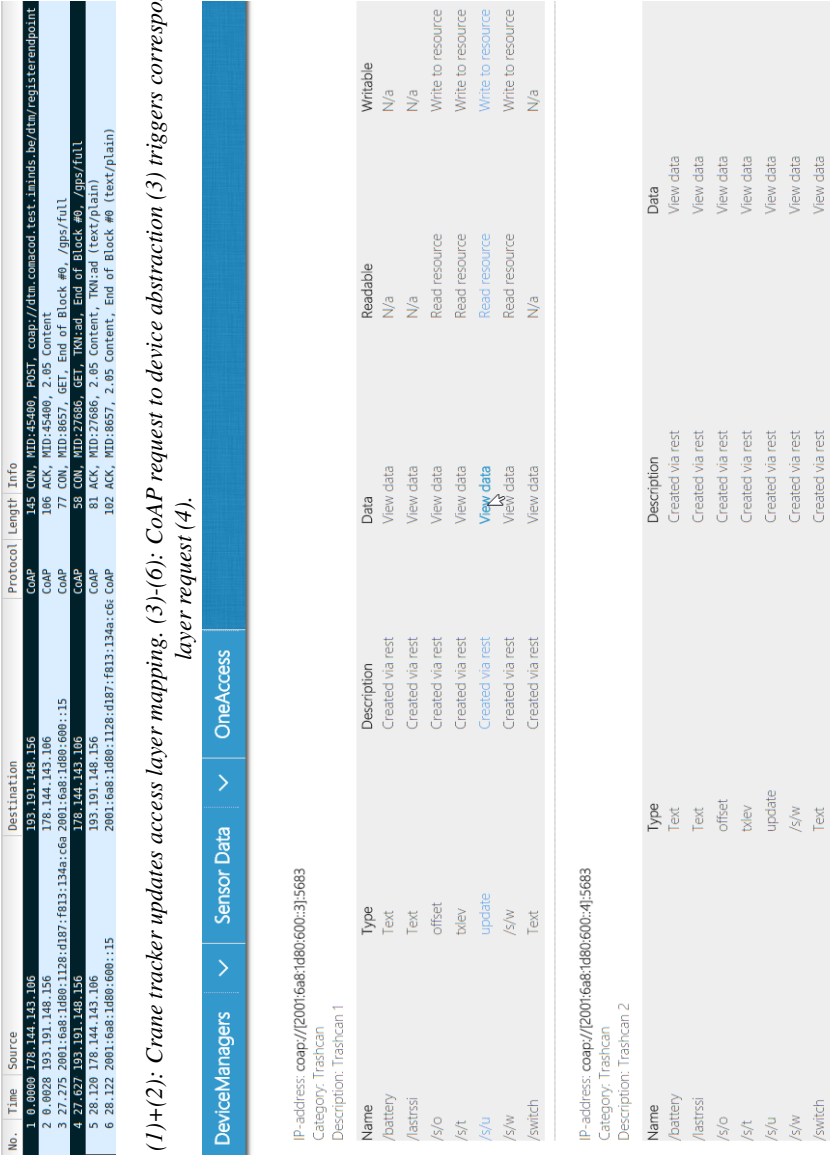


Figure 5.12: (1)+(2): Crane tracker updates access layer mapping. (3)-(6): CoAP request to device abstraction (3) triggers corresponding access layer request (4).

Figure 5.13: Control and management dashboard: the user is presented with a list of devices and resources. The user can access data collected from resources, retrieve a new resource representation or update a resource.

To illustrate that our cloud platform facilitates integration of constrained devices in the Internet, the project also implemented a control and management dashboard for the different devices from the previous paragraphs. This dashboard is hosted on server infrastructure owned by one of the project partners. HTTP is used for communication from the dashboard to the different types of constrained devices via the corresponding virtual devices and the HTTP/CoAP proxy. The dashboard discovers the available devices and their resources via the resource directory hosted in the abstraction layer of the platform. Figure 5.13 shows the interface offered to a user to inspect the list of available devices and resources. Resource attributes (such as type, readable and writable) are determined from the CoAP resource type that is available at discovery. For every resource, the user can choose to view collected data, retrieve a fresh representation or update the resource. Figure 5.14 shows the collected data for the /s/w resource of the waste bin tracker hosted at the `coap://[2001:6a8:1d80:600::4]` virtual device. Data includes switch events (i.e. waste bin lid events) and battery status.

Finally, the case study from section 5.2 where the crane tracker updates the location of the container tracker when it picks up a container was realized. One problem here was the IPv4-only Internet access provided by the crane's GPRS modem. Here the CoAP IPv4/IPv6 forward proxy was used for updating the location resource on the virtual device of the container tracker. Another problem was that of local discovery: i.e. how does the crane tracker discover the virtual device corresponding to the container it picked up. Here a number of solutions were formulated: using RFID communication for communicating the virtual device's

DeviceManagers

Sensor Data

OneAccess

Device IP-address: coap://[2001:6a8:1d80:600::4]:5683

Device Category: Trashcan

Device Description: Trashcan 2

Resource Name: /s/w

Resource Type: /s/w

Timestamp	Data
01/12/2014	Switch Current=0 Battery Voltage=3.61724853515625
02/12/2014	Switch Current=0 Switch Logitem=1 Switch Logitem=0 Battery Voltage=3.62091064453125
03/12/2014	Switch Current=0 Battery Voltage=3.61724853515625
03/12/2014	Switch Current=0 Switch Logitem=1 Switch Logitem=0 Battery Voltage=3.66943359375
03/12/2014	Switch Current=0 Battery Voltage=3.61724853515625
04/12/2014	Switch Current=0 Battery Voltage=3.6676025390625
04/12/2014	Switch Current=0 Switch Logitem=1 Switch Logitem=0 Battery Voltage=3.62274169921875
13/01/2015	Switch Current=0 Switch Logitem=1 Switch Logitem=0 Battery Voltage=3.6016845703125
14/01/2015	Switch Current=0 Switch Logitem=1 Switch Logitem=0 Battery Voltage=3.59893798828125
21/01/2015	Switch Current=0 Switch Logitem=1 Switch Logitem=0 Battery Voltage=3.59161376953125
22/01/2015	Switch Current=0 Battery Voltage=3.662109375
22/01/2015	Switch Current=0 Battery Voltage=3.5943603515625
23/01/2015	Switch Current=1 Switch Logitem=1 Battery Voltage=3.58062744140625
26/01/2015	Switch Current=0 Battery Voltage=3.59527587890625
27/01/2015	Switch Current=0 Switch Logitem=1 Switch Logitem=0 Battery Voltage=3.6602783203125
27/01/2015	Switch Current=0 Switch Logitem=1 Switch Logitem=0 Battery Voltage=3.5943603515625

Figure 5.14: The dashboard presents a table with collected data of a waste bin tracker to the user

address or searching the platform's L2 resource directory using the container's endpoint identifier (communicated via RFID or optically via a barcode). However, in the end a solution for this local discovery problem was not implemented in this proof of concept.

5.7 Related work

While cloud computing and the Internet of Things come from different backgrounds, recent developments show that they are increasingly being adopted together. The cloud and IoT can complement one another in a number of interesting ways, as will become clear from the works presented in the following paragraphs.

The work of Guinard D. was one of the first to propose a resource-oriented architecture for the Internet of Things [30] [31]. By implementing RESTful resources on things, the interfaces of things have become similar to those found on the World Wide Web and therefore this is referred to as the "Web of Things". When these resources are integrated into larger services, the result becomes a service-oriented architecture which is a well-known concept that is often realized on top of cloud computing infrastructure.

Historically, one of the first research domains where cloud computing and IoT have been combined was wireless sensors networks. In [32] a Sensor-Cloud infrastructure is introduced, where physical sensors are virtualized as virtual sensors on cloud computing. Motivation for doing so include, the limited resources and capabilities of physical sensors and the ease of use and management of virtual sensors. A comprehensive work on WSN and cloud is the survey [32] by Alamri et. al. The authors argue that the combination of Cloud and WSN enables remote access and allows to annotate sensors with XML in the cloud. The use of XML encourages the interchangeability of different types of sensors and allows to describe services offered by sensors (e.g. via the Web Service Description Language). Here also the use of web-services is presented to enable different applications to talk to one another. The resulting WSN is coined as a service-oriented sensor network.

In recent years, a sizeable amount of research has been performed on the combination of Cloud computing and the IoT. This has resulted in numerous papers as well cloud-based IoT platforms. In [9] Botta et al. introduce the CloudIoT paradigm which is the integration of Cloud and IoT. The paper is a good introduction to the topic as it lists the motivation, applications, related work, research challenges, open issues and platforms for the CloudIoT paradigm. The listed motivations for CloudIoT include the abundance of resources (storage, computational and communication) of the Cloud as well as significant improvements in scalability, interoperability and security. One identified research challenge is the integration of huge amounts of highly heterogeneous things into the Cloud, which overlaps with our research objectives from section 5.3.

Another work focuses on moving application logic from the firmware to the cloud. In [33] Kovatsch et al. introduce the Thin Server Model, where devices in the role of a server does not host any application logic. Instead, devices expose

their elementary functionality via RESTful services and any application logic of devices runs on application servers (possibly in the Cloud). The resulting IoT infrastructure is said to be agnostic of any applications. Our work adopts some of these ideas (e.g. thin devices and extra-device applications), however we also address the issue that not all devices can be modeled using the Thin Server Model. Some (legacy) devices implement proprietary technology or implement a client-based communication model (e.g. sleeping devices). The work presented here also looks at how these devices can be integrated into the Internet and be made available to such “Application servers”.

Works by Alam et. al [34] and Zaslavsky et. al [28] propose the Sensing and Sensor as a Service ideas. SenaaS exposes functional aspects of sensors as services by hiding technical details of the sensor from the user. By specifying and delivering sensor functionalities and capabilities as services, one can exploit all the existing service standards for interacting with sensors. Similar to our work, [34] defines an “IoT Virtualization Framework” where IoT devices are virtualized and offer virtual services. However, in our work the virtualization takes place at the device level (i.e. with every device having a virtual counterpart that is hosted at a specific IPv6 address) whereas the virtualization in [34] takes place on the service layer. Furthermore, the technology used in both approaches is different: our work focuses on the IETF stack for constrained devices whereas the work of Alam et. al focuses on the standards developed by the Open Geospatial Consortium (i.e. Sensor Web Enablement or SWE). One important difference is that our approach enables virtual devices to be used by constrained devices. This is typically not the case for the more verbose SWE standards.

In [10], Li et. al present a cloud-based Platform as a service solution (PaaS) for delivering IoT services. Like our work, the authors make the observation that today IoT services are typically delivered as physically-isolated vertical solutions where all system components are customized and tightly coupled for each use case. The authors propose instead to move to “virtual verticals” that are built on top of a common cloud infrastructure according to the presented IoT PaaS architecture. By building on top of the functionality already provided by the platform, IoT solution providers can deliver their services more efficiently and can continuously extend their product. Our work is similar to the IoT PaaS concept, but it is focused on solving the specific problem of integrating heterogeneous constrained devices into the IoT. Furthermore we argue that the IoT should move away from all verticals (both virtual and physical) in order to realize applications that span multiple domains and IoT product manufacturers.

In Cloud4Sens [35] two strategies for managing sensing resources in the cloud and providing them as a services are discussed and a cloud framework that handles these two strategies is presented. In the data-centric model, the cloud offers data to its clients as a service without knowing how the data is measured and processed. In the second model, the device is at center and clients can access data via devices and customize one or more virtual devices. To this end the architecture includes an SWE abstraction layer that enables a number of different thing deployments to interface with the cloud. The framework also includes a Software as a Service

component (SaaS) for offering data to interested clients and an Infrastructure as a Service (IaaS) component that enables clients to interact with virtual devices. Our solution can be seen as an alternative to the IaaS component that is geared towards supporting heterogeneous devices and that is designed to allow constrained devices to interact with other virtual devices. Through the resource adapters, our approach also allows to enhance virtual devices with additional functionality.

Next to the works presented above, there are also many IoT platforms - both commercial and academic - available today. The gap-analysis presented in [13] considers over 30 of such platforms. One identified shortcoming is the integration of sensor technologies without the use of gateways. Problems include the lack of standards and heterogeneous interaction models. Both are addressed by our work by relying on CoAP and showing how heterogeneous devices (with different communication models) can be integrated. Other shortcomings include data ownership and data fusion & sharing, while these topics are not addressed by our platform itself they were part of the control and management dashboard in the proof of concept in subsection 5.6.3. In the future we will look at how these can be integrated as part of the platform.

In [36] a system architecture for IoT cloud services based on CoAP named Californium is presented. The architecture consists of three stages (network, protocol and logic) that form a processing pipeline where each stage has its own separate thread pool. The result is a highly scalable architecture as proven by a comparison with state of art alternatives from both the CoAP and HTTP server domain. Even though the focus of the work is different than ours (scalability vs integration and heterogeneity), it is mentioned here to demonstrate that CoAP is a suitable protocol for (scalable) IoT cloud services.

CloudThings [37] is a service platform that allows users to run IoT application on cloud hardware. It includes tools for application development and for operational management and deployment. The platform uses 6LoWPAN and CoAP for communication with things and RESTful web services for integration with the cloud. The platform offers web services for data subscription and discovery. However, these are only accessible over HTTP which limits their use for constrained devices that wish to discover things via the platform. Also, the evaluation is very limited and considers only HTTP for Cloud-Thing communications.

5.8 Conclusion

We have shown that our cloud-based platform facilitates the integration of constrained IoT devices into other services without significantly impacting service and device operation. Experiments show that our platform supports a number of different communication models that are abstracted away from services by interfacing with a virtual CoAP server abstraction. Furthermore, the proof of concept demonstrates that the platform supports heterogeneous hardware platforms, communication models and proprietary protocols. The developed control and management dashboard shows that our platform can easily integrate constrained devices

into third party services.

Future work will focus on what is currently missing in the platform. By offering Data as a Service (DaaS) services, others can access (historical) data without being forced to capture this data when it is made available via the virtual device abstraction. This will also require a data ownership mechanism, as data will be offered separate from the device abstraction. Another weak point of the platform is device to device interaction between local devices. In case of real-time applications, mechanisms to allow direct access between devices (e.g. by redirecting devices to the device itself) might be necessary. However, due to the heterogeneity of the devices involved, we expect this to be a tall order. Finally, improving and quantifying the scalability of our platform (e.g. via dynamic routing tables as mentioned in section 5.6.1) will also be considered in the future.

Acknowledgements

The authors would like to acknowledge that part of this research was supported by the COMACOD project. The iMinds COMACOD project is co-funded by iMinds (Interdisciplinary institute for Technology) a research institute founded by the Flemish Government. Partners involved in the project are Multicap, OneAccess, Track4C, Invenso and Trimble, with project support of IWT.

References

- [1] Ericsson Inc. *More than 50 Billion Connected Devices - Taking connected devices to mass market and profitability*. Technical Report February, Ericsson, 2011. Available from: <http://vdna.be/publications/Wp-50-Billions.Pdf>, doi:284 23-3149 Uen.
- [2] D. Evans. *The internet of things: how the next evolution of the internet is changing everything*. Technical report, Cisco, 2011. Available from: <https://www.cisco.com/c/dam/en{ }us/about/ac79/docs/innov/IoT{ }IBSG{ }0411FINAL.pdf>.
- [3] Joseph Bradley, J. Barbier, and D. Handler. *Embracing the Internet of Everything To Capture Your Share of 14.4 Trillion USD*. Cisco Ibsg Group, page 2013, 2013. Available from: <http://www.cisco.com/web/about/ac79/docs/innov/IoE{ }Economy.pdf>.
- [4] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi. *From today's intranet of things to a future internet of things: a wireless-and mobility-related view*. Wireless Communications, IEEE, 17(6):44–51, 2010.
- [5] U. Hunkeler, H. L. Truong, and A. Stanford-clark. *MQTT-S – A Publish/Subscribe Protocol For Wireless Sensor Networks*. In Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on, pages 791–798, 2008. doi:10.1109/COMSWA.2008.4554519.
- [6] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler. *Standardized Protocol Stack for the Internet of (Important) Things*. Communications Surveys Tutorials, IEEE, 15(3):1389–1406, 2013. doi:10.1109/SURV.2012.111412.00158.
- [7] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. *Internet of things: Vision, applications and research challenges*. Ad Hoc Networks, 10(7):1497–1516, 2012.
- [8] J. Macaulay, L. Buckalew, and G. Chung. *Internet of Things in Logistics*. Technical report, DHL Trend Research, 2015. Available from: www.dhl.com/en/about{ }us/logistics{ }insights/dhl{ }trend{ }research/internet{ }of{ }things.html.
- [9] A. Botta, W. D. Donato, V. Persico, and A. Pescapé. *On the Integration of Cloud Computing and Internet of Things*. 2nd International Conference on Future Internet of Things and Cloud (FiCloud), 2014. Available from: <http://wpage.unina.it/walter.dedonato/pubs/iot{ }ficloud14.pdf>.

- [10] F. Li, M. Voegler, M. Claessens, and S. Dustdar. *Efficient and Scalable IoT Service Delivery on Cloud*. In 2013 IEEE Sixth International Conference on Cloud Computing, pages 740–747. IEEE, jun 2013. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6676764>, doi:10.1109/CLOUD.2013.64.
- [11] L. Atzori, A. Iera, and G. Morabito. *The Internet of Things: A survey*. Computer Networks, 54(15):2787–2805, oct 2010. Available from: <http://linkinghub.elsevier.com/retrieve/pii/S1389128610001568>, doi:10.1016/j.comnet.2010.05.010.
- [12] K. Romer and F. Mattern. *The design space of wireless sensor networks*. IEEE Wireless Communications, 11(6):54–61, dec 2004. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1368897>, doi:10.1109/MWC.2004.1368897.
- [13] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma. *A gap analysis of Internet-of-Things platforms*. Computer Communications, 89-90:5–16, feb 2016. Available from: <http://arxiv.org/abs/1502.01181>, arXiv:1502.01181.
- [14] S. Husain, A. Prasad, A. Kunz, A. Papageorgiou, and J. Song. *Recent Trends in Standards Related to the Internet of Things and Machine-to-Machine Communications*. Journal of Information and communication convergence engineering, 12(4):pp.228–236, 2014. Available from: <http://dx.doi.org/10.6109/jicce.2014.12228>, doi:10.6109/jicce.2014.12.4.228.
- [15] I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. Van den Abeele, E. De Poorter, I. Moerman, and P. Demeester. *IETF standardization in the field of the Internet of Things (IoT): a survey*. Journal of Sensor and Actuator Networks, 2(2):235–287, 2013.
- [16] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. *RFC 7252: Constrained Application Protocol (CoAP)*, 2014. Available from: <https://tools.ietf.org/html/rfc7252>.
- [17] C. Bormann, A. P. Castellani, and Z. Shelby. *CoAP: An Application Protocol for Billions of Tiny Internet Nodes*. IEEE Internet Computing, 16(2):62–67, mar 2012. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6159216>, doi:10.1109/MIC.2012.29.
- [18] K. Hartke. *Observing Resources in CoAP*. IETF CoRE Working Group, 2014. Available from: <https://tools.ietf.org/html/draft-hartke-coap-observe>.
- [19] L. Daniel, M. Kojo, and M. Latvala. *Experimental Evaluation of the CoAP, HTTP and SPDY Transport Services for Internet of Things*. In 7th International Conference on Internet and Distributed Computing Systems, pages 111–123, 2014.

- [20] Z. Shelby and C. Bormann. *Core resource directory*. IETF CoRE Working Group, 2014. Available from: <https://tools.ietf.org/html/draft-ietf-core-resource-directory-02>.
- [21] M. Vial. *CoRE Mirror Server*, 2013. Available from: <https://tools.ietf.org/html/draft-vial-core-mirror-server-01>.
- [22] F. Van den Abeele, J. Hoebeke, I. Moerman, and P. Demeester. *Fine-grained management of CoAP interactions with constrained IoT devices*. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–5, 2014. doi:10.1109/NOMS.2014.6838368.
- [23] F. Van den Abeele, T. Vandewinckele, J. Hoebeke, I. Moerman, and P. Demeester. *Secure communication in IP-based wireless sensor networks via a trusted gateway*. In 2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (IEEE ISSNIP 2015), Porto, Portugal, 2015.
- [24] E. Dijk, A. Rahman, T. Fossati, S. Loreto, and A. Castellani. *Guidelines for HTTP-CoAP Mapping Implementations*. IETF CoRE Working Group, 2014. Available from: <https://tools.ietf.org/html/draft-ietf-core-http-mapping-06>.
- [25] B. Savolainen, Teemu Silverajan. *CoAP Communication with Alternative Transports*, 2015. Available from: <https://tools.ietf.org/html/draft-silverajan-core-coap-alternative-transports-08>.
- [26] M. Becker, K. Li, K. Kuladinithi, and T. Poetsch. *Transport of CoAP over SMS*, 2014. Available from: <https://tools.ietf.org/html/draft-becker-core-coap-sms-gprs-05>.
- [27] C. Bormann, M. Ersue, and A. Keranen. *RFC 7228: Terminology for Constrained-Node Networks*. Technical report, IETF, 2014. Available from: <http://tools.ietf.org/html/rfc7228>.
- [28] A. Zaslavsky, C. Perera, and D. Georgakopoulos. *Sensing as a service and big data*. In International Conference on Advances in Cloud Computing, 2013.
- [29] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. *Software-based sensor node energy estimation*. In Proceedings of the 5th international conference on Embedded networked sensor systems, pages 409–410. ACM, 2007.
- [30] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. *Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services*. Services Computing, IEEE Transactions on, 3(3):223–235, 2010. doi:10.1109/TSC.2010.3.

- [31] D. Guinard, V. Trifa, F. Mattern, and E. Wilde. *From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices*. In *Architecting the Internet of Thing*, pages 97–129. Springer Berlin Heidelberg, 2011. Available from: https://link.springer.com/chapter/10.1007/978-3-642-19157-2_5, doi:10.1007/978-3-642-19157-2-5.
- [32] M. Yuriyama and T. Kushida. *Sensor-Cloud Infrastructure - Physical Sensor Management with Virtualized Sensors on Cloud Computing*. 2013 16th International Conference on Network-Based Information Systems, 0:1–8, 2010. doi:<http://doi.ieeecomputersociety.org/10.1109/NBiS.2010.32>.
- [33] M. Kovatsch, S. Mayer, and B. Ostermaier. *Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things*. ... Mobile and Internet ..., 2012. Available from: http://ieeexplore.ieee.org/xpls/abs/_all.jsp?arnumber=6296948.
- [34] S. Alam, M. M. R. Chowdhury, and J. Noll. *Senaas: An event-driven sensor virtualization approach for Internet of Things cloud*. In 2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications, pages 1–6. IEEE, nov 2010. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5678060>, doi:10.1109/NESEA.2010.5678060.
- [35] M. Fazio and A. Puliafito. *Cloud4sens: a cloud-based architecture for sensor controlling and monitoring*. IEEE Communications Magazine, 53(3):41–47, mar 2015. Available from: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7060517>, doi:10.1109/MCOM.2015.7060517.
- [36] M. Kovatsch, M. Lanter, and Z. Shelby. *Californium: Scalable cloud services for the internet of things with coap*. In *Proceedings of the 4th International Conference on the Internet of Things (IoT 2014)*, 2014.
- [37] J. Zhou, T. Leppanen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, and H. Jin. *CloudThings: A common architecture for integrating the Internet of Things with Cloud Computing*. In *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2013*, pages 651–657, 2013. doi:10.1109/CSCWD.2013.6581037.

6

Scalability analysis of large-scale LoRaWAN networks in ns-3

Background

The advent of Low Power Wide Area Networks (LPWANs) promises to bring low power, long range communication to the Internet of Things (IoT) at a low cost. The low data rate communication offered by LPWANs is a good choice for use cases where devices are battery powered, communication is sporadic and message sizes are small. As such, LPWANs are a cost effective technology for monitoring many of the objects around us. For example, connected waste bins, connected street lights, connected fire hydrants and connected water and electricity meters are either already available or under development. In industry, monitoring of rail way infrastructure, liquid and gas tanks, pipelines and electricity and water networks are some of the promising use cases.

Today, LPWAN end devices are integrated into services by programming against a data integration API offered by the backend of a LPWAN network operator. E.g. SigFox offers a REST API to their backend for device integration and The Things Networks offers an MQTT-based API for backend integration. Appendix A presents an alternative for LoRaWAN device integration which employs device virtualization. This alternative hosts LoRaWAN end devices as virtual CoAP servers, which handle the integration with the LPWAN backend API. This frees developers from integrating with the LPWAN-specific data integration API, instead they are able to integrate via CoAP. Additionally, (resource-constrained) WoT devices are able to interface with these virtual CoAP servers. A third integration option stems from the lpwan IETF working group, which is looking to bring IP to LP-

WAN end devices. Due to the severely limited frame sizes in LPWANs (e.g., 12 octets for SigFox and 19 octets for LoRaWAN (worst case)), the working group is standardizing compression schemes for IPv6 that reduce the header size. Next to IP header compression, the lpwan working group is also looking into CoAP header compression.

Once end-to-end IP and CoAP are available in LPWANs, many of the techniques presented in this dissertation can be applied to LPWANs. For example, session-based security and comprehensive authentication and authorization could be supported as discussed in Chapter 3. Extensive traffic filtering is expected to be necessary, in order to avoid saturating the limited capacity of LPWANs. As LPWAN devices are asleep for long periods of time, the mirror server model as discussed in Chapter 5 could be applied. Alternatively, the reverse proxy approach of Chapter 3 could prove valuable for handling sleepy devices.

The request-response message exchange pattern of CoAP differs from the traffic pattern dominant in LPWANs, where traffic mostly originates from end devices (i.e. upstream traffic) and traffic towards end devices (i.e. downstream traffic) is uncommon. In LoRaWAN, confirmed messages have a traffic pattern comparable to the request-response pattern, as every upstream message solicits a downstream acknowledgment. One of the questions studied in this chapter is how sending data as confirmed messages impacts the traffic delivery ratio in LoRaWAN networks. The results show that the limited downstream capacity of LoRaWAN networks is detrimental to the delivery ratio of confirmed upstream messages, as the network is unable to send downstream acknowledgments due to radio duty cycle restrictions. This problem is further exacerbated as missing acknowledgments lead to end devices retransmitting messages that were in fact successfully received by the network server. Similar problems would occur for CoAP traffic, where acknowledgments for confirmable messages or responses to (non)confirmable requests would suffer a similar fate. Solutions such as CoAP No Server Response (cfr. RFC7967) could help to alleviate this problem by letting LPWAN end devices express their disinterest to a CoAP server in all responses against a particular request. In order to comprehend these effects, this chapter studies the scalability of large-scale LoRaWAN by means of extensive modeling and evaluations in a network simulator. The impact of sending upstream data as unconfirmed or confirmed messages and the impact of downstream data traffic are studied.

Floris Van den Abeele, Jetmir Haxhibeqiri, Ingrid Moerman and Jeroen Hoebeke

Submitted to IEEE Internet of Things Journal on the Fifth of May, 2017.

Abstract As LoRaWAN networks are actively being deployed in the field, it is important to comprehend the limitations of this Low Power Wide Area Network

technology. Previous work has raised questions in terms of the scalability and capacity of LoRaWAN networks as the number of end devices grows to hundreds or thousands per gateway. Some works have modeled LoRaWAN networks as pure ALOHA networks, which fails to capture important characteristics such as the capture effect and the effects of interference. Other works provide a more comprehensive model by relying on empirical and stochastic techniques. This work uses a different approach where a LoRa error model is constructed from extensive complex baseband bit error rate simulations and used as an interference model. The error model is combined with the LoRaWAN MAC protocol in an ns-3 module that enables to study multi channel, multi spreading factor, multi gateway, bi-directional LoRaWAN networks with thousands of end devices. Using the lorawan ns-3 module, a scalability analysis of LoRaWAN shows the detrimental impact downstream traffic has on the delivery ratio of confirmed upstream traffic. The analysis shows that increasing gateway density can ameliorate but not eliminate this effect, as stringent duty cycle requirements for gateways continue to limit downstream opportunities.

6.1 Introduction

With the ongoing continuous growth of the Internet of Things, the number of IoT application domains and deployments continues to increase. Market forecasts illustrating this growth estimate that the number of connected IoT devices will continue to grow at an annual rate of 32% and will reach 20.8 billion IoT end points by the end of this decade [1]. Some of these novel IoT applications require low-rate, long-range and delay-tolerant wireless communication at very low energy usage and cost. These types of requirements are hard to fulfill using traditional Machine to Machine technologies such as cellular or WPAN [2]. Low Power Wide Area Networks (LPWANs) are a new set of technologies that are designed to fill this gap in traditional technologies. By combining low energy usage with long range communication, they promise to bring connectivity that suits large scale, low power and low cost IoT deployments with battery lives up to ten years [2]. The 2016 Cisco VNI 2015-2020 data traffic forecast estimates that the share of LPWANs in global M2M connections will grow from 4% in 2015 to 28% in 2020 [3]. Similar market share numbers for LPWANs are reported in [4].

LoRaWAN [5] is an LPWAN technology that builds on top of the LoRa modulation scheme, which is developed by Semtech. The LoRa alliance has standardized LoRa radio usage in sub-GHz unlicensed spectrum for most areas in the world. By combining sub-GHz propagation and the LoRa modulation, LoRaWAN networks can cover large areas with only limited amounts of infrastructure. LoRaWAN networks are being deployed today. For instance in Belgium, Proximus, a large telecommunications company, provides LoRaWAN coverage in the whole of Flanders and in the major cities in Wallonia [6]. Another interesting initiative is The Things Network, where a community of mostly volunteers is collaborating to build a world-wide LoRaWAN network.

While LoRaWAN networks are already being actively deployed, a number of questions remain unanswered about their performance as said networks grow larger and larger. Most LPWAN technologies promise to connect a massive number of devices (e.g. tens of thousands of devices per LoRaWAN gateway), but how valid is this claim in the case of LoRaWAN networks? What is the impact of network parameters on large-scale LoRaWAN networks? Apart from a large number of devices, what is the impact of multiple gateways in large-scale networks? Does increasing gateway density yield measurable network performance benefits?

The work presented here aims to provide answers to the research questions posed above by studying LoRaWAN networks in the ns-3 network simulator. A network simulator provides the flexibility to relatively quickly study a large number of different LoRaWAN scenarios at the expense of accuracy due to limitations in the modeling complexity. As such, the presented ns-3 module allows to study LoRaWAN networks with a varying numbers of end devices and gateways, different traffic types and patterns, different data rates, different (re)transmission and receive configurations and many other parameters. Finally, the presented work also allows to study the impact of more fundamental changes to LoRaWAN network mechanics as everything is implemented in software.

While a number of existing works have studied the scalability of LoRaWAN networks, they do not take into account the impact of downstream traffic, interference between transmitters and the presence of multiple LoRaWAN gateways in dense deployments. Additionally, the spectrum modeling technique newly introduced in ns-3.26, which is applied in the LoRaWAN ns-3 module, should enable the module to be used in future coexistence studies with heterogeneous technologies such as 802.11ah. A more detailed comparison with works from literature is available in section 6.6.

The contributions of this work are as follows. Firstly, we built an error model for the LoRa modulation for different code rates and spreading factors. Secondly, we developed a comprehensive implementation of the LoRaWAN standard in the ns-3 simulator with support for class A end devices, multi gateway networks and an elementary network server. Thirdly, we conducted a scalability study focusing on the impact of confirmed versus unconfirmed messages and the impact of downstream traffic in large-scale LoRaWAN networks.

Before detailing how LoRaWAN networks are modeled in ns-3 in section 6.4, the next section provides the necessary background on LoRa and LoRaWAN in order to comprehend the modeling efforts in the ns-3 module. Section 6.5 presents the scalability analysis itself. The results of the analysis are discussed and are compared to literature in section 6.6.

6.2 Background: LoRa, LoRaWAN and ns-3

As a Low Power Wide Area Network technology, LoRaWAN networks offer benefits such as large coverage areas and long battery life operation for end devices. Unlike conventional network technologies (e.g. cellular and LAN), the trade-off in

data rate versus range leans heavily towards range in LoRaWAN. LoRaWAN networks further employ a proprietary physical modulation technique (named LoRa) which has been developed with long range and low power operation at its core. This section describes some of the aspects of LoRaWAN networks that are important for the remainder of this paper.

The LoRa physical modulation is a chirp spread spectrum (CSS) technique where the base symbol is an up-chirp. An up-chirp is a signal in which the frequency increases with time. More specifically, in LoRa the frequency increases linearly with time so that the frequency of the up-chirp sweeps the entire bandwidth of the signal. The constellation diagram of LoRa consists of time-shifted up-chirps. Therefore, at the receiver the demodulation process attempts to determine the time shift in the received up-chirp.

Next to bandwidth, the spreading factor (SF) is a second important parameter of the LoRa modulation as it provides the flexibility of trading range for data rate. The spreading factor can range from seven to twelve and determines the LoRa symbol rate as: $R_s = \frac{BW}{2^{SF}}$. As the spreading factor increases the symbol rate is lowered, thereby trading reduced data rate for increased range. For a bandwidth of 125kHz, the PHY data rates range from 6835bps to 365bps for SF seven to twelve. The different spreading factors are orthogonal. This means that a LoRa gateway can receive multiple transmissions on different spreading factors simultaneously. Note that SF bits are mapped per LoRa symbol.

In order to further increase the robustness of the LoRa modulation, additional techniques such as forward error correction and interleaving are employed. These techniques are discussed in section 6.4.1.1. Additionally, LoRa radios operate in the sub-GHz unlicensed bands as they provide a good trade-off between available unlicensed spectrum and reduced path loss. The combination of these design considerations results in a high link budget at the expense of data rate, which means that LoRa transmissions can still be received successfully even though they are below the noise floor.

LoRaWAN networks employ the robust LoRa modulation in order to achieve long range operation. They are standardized by the LoRa alliance, which has defined medium access, frame formats, provisioning and management messages, security mechanisms, device management and other aspects. Figure 6.1 illustrates that LoRaWAN networks form one hop star topologies around gateways, which act as packet forwarders between end devices and a central network server (NS). The network server is responsible for MAC layer processing and acts as a portal between applications running on end devices and application servers. The LoRaWAN standard defines three classes for end devices (A, B and C) in order to cater to a number of different scenarios. This work focuses on class A end devices as this class provides the longest battery life.

Class A end devices have their transceivers in deep sleep for the majority of the time and wake up infrequently to transmit data toward the network server. Wireless medium access in LoRaWAN follows a ALOHA scheme, which does not employ listen before talk, and is therefore subject to restrictions in most areas of the world. In Europe for example, the 868MHz band consists of a number of sub-

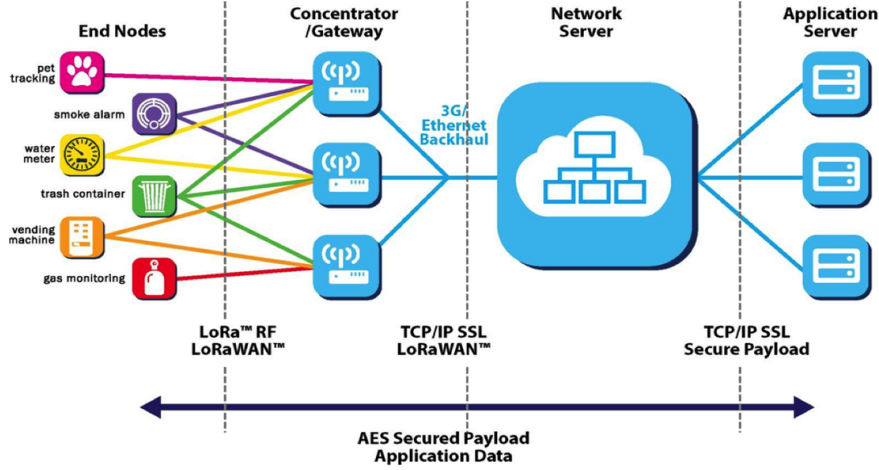


Figure 6.1: Architecture of LoRaWAN networks (image courtesy of Semtech)

bands where radio duty cycle restrictions range from 0.1% to 10% with 1% being most common. Note that each of these sub-bands is composed of one or more channels.

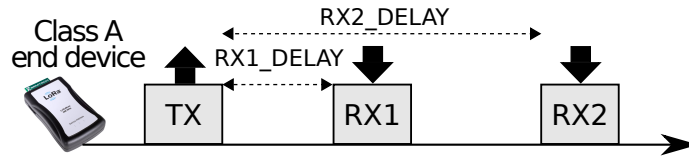


Figure 6.2: Downlink receive window timing for LoRaWAN class A end devices

As class A end devices are unreachable most of the time, the opportunities for sending to the device are scarce. As per the standard, class A end devices are obliged to open one or two receive windows after each upstream transmission in order to allow the NS to deliver a potential message to the end device. Figure 6.2 illustrates the timing for opening these windows, which is equal to one and two seconds after the end of the upstream transmission for the first and second receive window respectively. When an end device receives a downlink transmission in the first window, it is freed from opening the second window. Otherwise, it must open the second window. Note that an end device listens on the same channel and SF as the last upstream transmission in the first receive window (unless RX1DROffset differs from zero), while it listens on a separate channel and SF12 in the second window. Additionally, a class A end device must defer all pending upstream transmissions until after the receive window(s).

Finally, both upstream and downstream messages may be sent as either confirmed and unconfirmed messages. Confirmed messages are sent using a straight-

forward retransmission scheme at the discretion of the end device, without violating the duty cycle restrictions. Downstream (re)transmissions have to wait for an open receive window and their timing is therefore controlled by the upstream traffic timing of an end device. The NS can however set a frame pending bit in a downstream message in order to signal to an end device that it might want to open a receive window sooner than normal.

To conclude this section ns-3 is briefly introduced. Ns-3 is an open source discrete-event network simulator, targeted primarily for research and educational use [7]. It provides support for Wi-Fi, LTE, IEEE 802.15.4 and other networks and also implements an IP networking stack. Ns-3 is often used for evaluating wireless networks. Examples of recent work include [8], which investigates the performance of IEEE 802.11n frame aggregation, and [9], which has added support for the new sub-GHz IEEE 802.11ah standard to ns-3. A novel feature in ns-3 is the SpectrumPhy class, which enables modeling of inter-technology interference and which is used in this work to implement the LoRa PHY.

6.3 Problem statement and approach

As mentioned, the anticipated growth of IoT in general and LPWANs in particular raises the question how these technologies will scale as the size of these networks grows. Vendors are keen to highlight the positive aspects of their respective products, but are less inclined to point out certain flaws. Therefore, an objective study of the limits of these LPWANs is warranted and needed.

Specifically for LoRaWAN networks, this work attempts to explore the limits of these networks in terms of size. This work attempts to answer the following research questions:

- What is the impact of end device density on network performance?
- What is the impact of the different message types on network performance?
- What is the impact of assigning data rates to end devices?
- What is gained in terms of network performance by increasing the gateway density?

In order to formulate an answer to these questions, a simulation based approach was followed as it allows modeling large scale networks (i.e. up to 10 000 end devices). The ns-3 simulator was a natural choice due to its widespread adoption in the network research community. Also, the new SpectrumPhy feature promises to enable modeling multiple LPWAN technologies in parallel in ns-3 and to study co-existence and other problems in the future. Finally, this work aspires to provide a useful tool for research into LoRaWAN networks.

6.4 LoRaWAN ns-3 module

Our modeling of LoRaWAN networks in ns-3 comprises a number of different elements. Firstly, an error model for the LoRa modulation was implemented in ns-3 based on baseband simulations of a LoRa transceiver over an Additive White Gaussian Noise (AWGN) channel. Secondly, the LoRaWAN PHY and MAC layers were added in ns-3 for gateways and class A end devices. Thirdly, ns-3 applications were developed to represent LoRaWAN class A end devices and LoRaWAN gateways. Finally, a simple LoRaWAN network server was added to ns-3.

An overview of the LoRaWAN ns-3 module is presented in figure 6.3. While end device nodes contain a single MAC/PHY pair, gateways consists of one MAC/PHY pair per supported spreading factor. For example a gateway that supports multi-SF (i.e. able to receive all LoRa spreading factors simultaneously) on six channels contains 36 MAC/PHY pairs. Apart from the components listed in figure, the ns-3 module also contains a number of unit tests and examples of varying complexity. The lorawan ns-3 module is publicly available at <https://github.com/imec-idlab/ns-3-dev-git/tree/lorawan>. Future works that use this ns-3 module, are requested to cite this manuscript.

6.4.1 LoRa PHY error model

6.4.1.1 LoRa PHY baseband implementation

In order to model the effects of path loss and intra-LoRa interference, an error model for the LoRa PHY has been developed in ns-3. The basis for this error model is a series of complex baseband Matlab simulations that measure the bit error rate (BER) for different LoRa PHY configurations over an AWGN channel. A block diagram of the BER simulations is shown in figure 6.4. The EP 2763321 A1 patent [10] and the works of Matt Knight [11] and Pieter Robyns [12] were invaluable resources for building the LoRa baseband implementation.

The information bits generated at the information source are mapped to code bits by the error correction encoder. This encoder implements the 5/4, 7/4 and 8/4 code rates available in LoRa¹. While the 5/4 CR is a simple parity check code, the 7/4 and 8/4 CRs are (7,4) and (8,4) linear error-correcting Hamming codes. These codes can correct one bit error and detect up to two bit errors.

Next, the diagonal interleaver shuffles the code bits so that at its output, groups of PPM bits consist of bits from the same bit position of PPM consecutive code words. For example, the first output word groups the bits at position 0 from PPM consecutive code words. PPM represents the bit length of the output words of the interleaver. In LoRa the PPM of the interleaver is equal to the LoRa spreading factor. Consequently, the number of bits mapped per LoRa symbol is equal to the spreading factor. Due to the interleaver, a lost symbol at the receiver is converted into PPM 1-bit errors over PPM consecutive code words (rather than one PPM-bit error in one code word without the interleaver).

¹The 6/4 code rate was not implemented as it is seldom used.

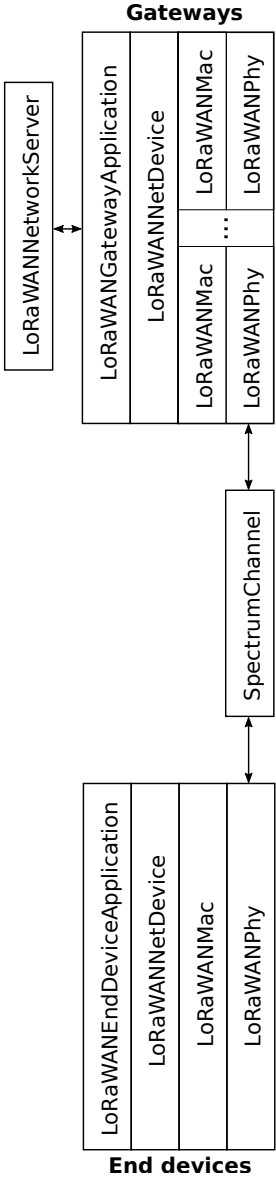


Figure 6.3: LoRaWAN ns-3 module overview: class A end devices, gateways and the network server

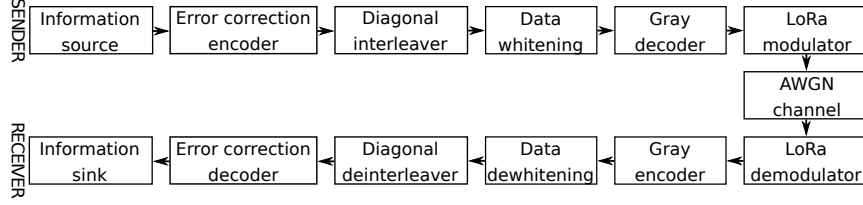


Figure 6.4: Block diagram of LoRa PHY baseband implementation: sender, AWGN channel and receiver

After the interleaver, the output words are whitened in order to boost the entropy of the information source. Note that in the BER simulations the information bits are drawn from a uniform distribution, therefore the entropy of the information source is already at its maximum. Before passing the whitened bit stream to the modulator, it is reverse Gray mapped first. This produces a sequence of integers, which are fed to the LoRa modulator. At the LoRa modulator, a sequence of N time-shifted complex baseband up-chirp samples is generated via a phase accumulator as given by equation 6.1 where N , the number of samples per baseband symbol, is equal to $2^{SF} \frac{f_s}{BW}$. The input integer determines the time-shift of the up-chirp.

$$m(i) = \begin{cases} \exp(-j\pi) & \text{if } i = 0 \\ m(i-1) \exp(jf(i)) & \text{if } i = 1, \dots, N-1 \end{cases} \quad (6.1)$$

Where the instantaneous frequency $f(i)$ is given by equation 6.2:

$$f(i) = -\pi + \frac{i}{N} 2\pi, \text{ for } i = 1, \dots, N-1 \quad (6.2)$$

Next, the samples of the LoRa symbol are sent over the AWGN channel for a given signal to noise ratio (SNR) as per equation 6.3:

$$c(i) = m(i) + \sqrt{\frac{E_s}{2SNR}} [\mathcal{N}(0;1) + j\mathcal{N}(0;1)] \quad (6.3)$$

for $i = 0, \dots, N-1$

where $\mathcal{N}(0;1)$ is the standard normal distribution and $SNR = 10^{SNR_{dB}/10}$. Note that the energy per symbol is equal to one for the LoRa modulator.

At the receiver, the LoRa demodulator employs correlation-based demodulation where the received symbol is correlated to all known LoRa symbols. The decision on which symbol was sent, is made by selecting the LoRa symbol with the maximum correlation value. After demodulation, the receiver chain is the reverse of the sender chain. The error rate is measured in the information bits, after error correction.

6.4.1.2 LoRa PHY BER simulations

In order to determine the BER of the LoRa physical layer, simulations were ran for the LoRa PHY parameters listed in table 6.1. There was no oversampling, so therefor $N = 2^{SF}$ holds. The simulations were ran for SNR values in steps of 1dB in the ranges as published in the table.

Table 6.1: LoRa PHY parameters for BER simulations

BW	SF	CR	SNR(dB)	BW	SF	CR	SNR(dB)
125kHz	7	1,3	[-20..0]	125kHz	11	1	[-23..-13]
125kHz	8	1,3	[-20..0]	125kHz	11	3	[-25..-13]
125kHz	9	1,3	[-20,-8]	125kHz	12	1,3	[-26..-17]
125kHz	10	1,3	[-22,-8]				

Afterwards an exponential curve as per equation 6.4 was fitted to a subset of the logarithmic values of the measured BER values. The subset of BER values used for curve fitting was determined as followed. Firstly, measured BER values of zero were discarded. Secondly, BER values were added to the subset until the BER reached a value where the corresponding packet delivery rate (PDR) dropped below one in a million for a 13B packet. The packet length of 13B stems from the minimum LoRa PHY payload length for a LoRaWAN transmission: 1B MAC header, 8B frame header and 4B MIC. Additionally, for every curve fit a SNR cut-off point was chosen so that the packet delivery rate for a 13B (=108b) packet was equal to one in a million at the cut-off point. Table 6.2 lists the details of the curve fit plus the SNR cut-off point for every LoRa PHY configuration that was simulated. Figure 6.5 plots the curve fits for SF7 to SF12, only code rate 3 is plotted.

$$\log_{10}(BER(SNR_{dB})) = \alpha \exp(\beta SNR_{dB}) \quad (6.4)$$

6.4.2 LoRaWAN PHY layer

After the error model was completed, work started on the implementation of the LoRaWAN PHY layer in ns-3. By building the LoRaWANPhy class on the SpectrumPhy concept [13], inter-technology simulations (e.g. interference testing) are anticipated to be feasible in the future. The majority of the PHY models available in ns-3 employ a chunk-based signal to interference noise ratio (SINR) approach for modeling the influence of propagation loss and intra-technology interference during packet reception [14] [15] [16]. Every time the SINR changes during packet reception (e.g. an interfering transmission starts or ends), a new chunk is started and the error rate of the previously received chunk is evaluated based on the constant SINR and bit length of this chunk and the BER as provided by the error model. Note that the LoRaWANPhy only initiates packet reception

Table 6.2: Exponential curve fit parameters for the LoRa PHY error model in ns-3

SF	CR	α	β	rsquare	SNR cut-off (dB)
7	1	-30.2580	0.2857	0.9997	-12.2833
7	3	-105.1966	0.3746	0.9999	-12.6962
8	1	-77.1002	0.2993	0.9999	-14.8485
8	3	-289.8133	0.3756	0.9995	-15.3588
9	1	-244.6424	0.3223	0.9993	-17.3749
9	3	-1114.3312	0.3969	0.9994	-17.9260
10	1	-725.9556	0.3340	0.9996	-20.0254
10	3	-4285.4440	0.4116	0.9991	-20.5581
11	1	-2109.8064	0.3407	1.0000	-22.7568
11	3	-20771.6945	0.4332	0.9996	-23.1791
12	1	-4452.3653	0.3317	0.9986	-25.6243
12	3	-98658.1166	0.4485	0.9993	-25.8602

for transmissions with an SINR value above the SNR cut-off value. Incoming transmissions which fall below the cut-off value are dropped immediately by the PHY.

Apart from the reception modeling, the LoRaWANPhy ns-3 class also implements a finite state machine to structure its execution flow. The FSM has six states as shown in fig 6.6. The transitions between states are mostly triggered from the MAC layer (not shown in the figure). In the RX_ON and TX_ON states the PHY is ready to respectively start a packet reception or transmission. In the BUSY_RX state the PHY is busy receiving a transmission (as per the aforementioned chunk-based reception). In the BUSY_TX state the PHY is sending a transmission. Ongoing receptions and transmissions may be canceled at any time, which is indicated by the state transitions to TRX_OFF. The PHYs of class A end devices are expected to be in the Idle state most of the time, whereas the PHYs of gateways are expected to be in the RX_ON state for the majority of the time. There are no differences in the PHY ns-3 classes between class A end devices and gateways. Hence, differences in transceiver design between end devices and gateways are not taken into account in this model.

6.4.3 LoRaWAN MAC layer

The driver of the PHY layer is the LoRaWANMac ns-3 class. Its functionality includes queuing packets for delivery, opening receive windows and handling retransmissions on end devices and keeping track of a node's radio duty cycle (RDC). While there is one LoRaWANMac class for both class A end devices and gateways, the functionality of this class differs as e.g. retransmissions for gateways are handled by the network server (see section 6.4.6). Likewise, the gateway MAC has no concept of receive windows as it always listening for upstream traffic (when not

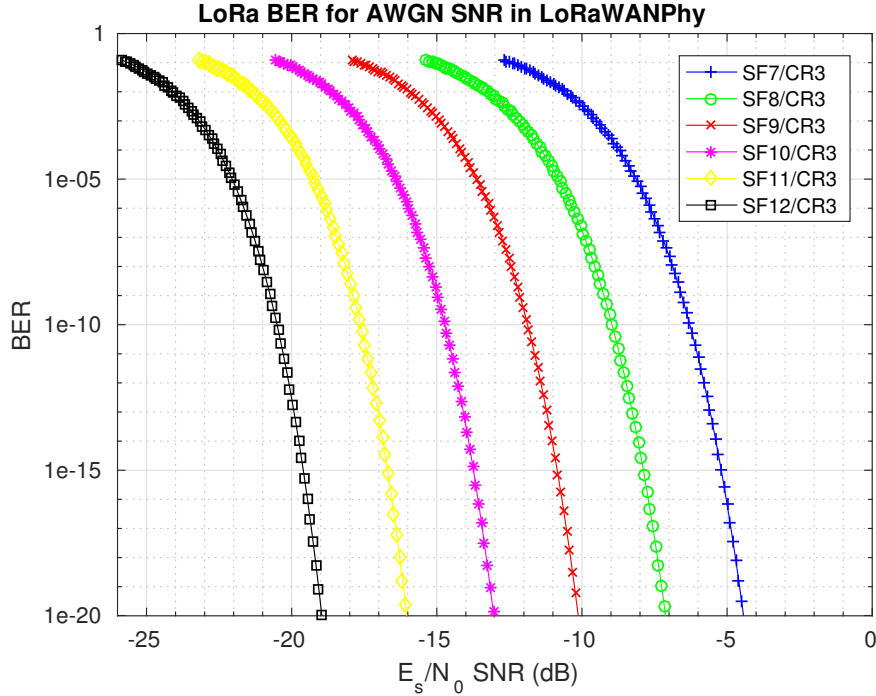


Figure 6.5: Curve fits used for the LoRa PHY error model in ns-3.

transmitting).

Similarly to LoRaWANPhy, the LoRaWANMac class also implements an FSM as depicted in figure 6.7. While an end device MAC object passes through all states in figure 6.7, gateway MAC objects are limited to three states. The UNAVAIL state is a case unique to gateways, where the MAC is blocked from sending a packet. This state is activated when one of the other MACs on the gateway is in the TX state, thereby prohibiting simultaneous transmissions on different MAC objects on the same gateway.

The chain at the top of the figure is related to the mandatory receive windows for class A end devices in LoRaWAN networks. After the TX state, an end device always transitions to the WRW1 state. The end device spends one second (starting from the end of the transmission) in this ‘wait for RW1’ state, after which it opens RW1. The end device checks whether a PHY preamble has been received 12.25 LoRa symbols after the beginning of the receive window. If a preamble has been detected, the device continues receiving the downstream transmission. If a preamble has not been detected, it closes the receive window and transitions to the WRW2 state. If the end device successfully receives a downstream transmission in RW1, it transitions to the IDLE state. Otherwise, it closes the receive window and transitions to the WRW2 state.

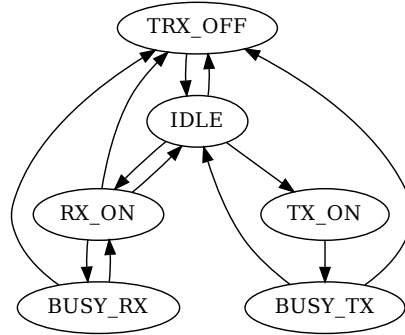


Figure 6.6: Finite state machine of the LoRaWANPhy class in ns-3

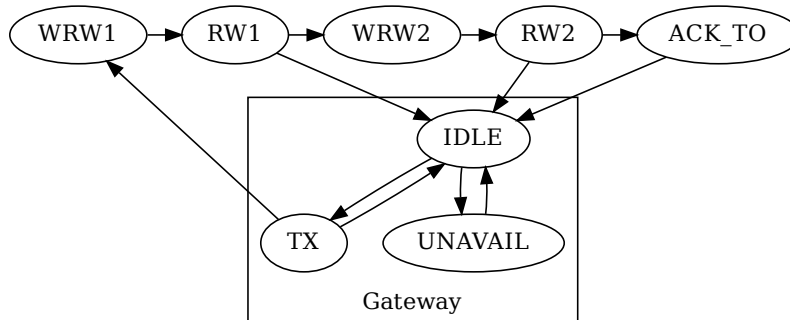


Figure 6.7: The LoRaWANMac FSM consists of three states for gateways and seven states for class A end devices

In the WRW2 state, the end device is waiting to open the second receive window (in the RW2 state) after two seconds after the end of the transmission. The same PHY preamble check from RW1 is performed after opening RW2. If a downstream transmission is received in RW2, the end device will transition to the IDLE state. Otherwise, it might transition to the ACK_TO (acknowledgment timeout) state where it spends a random length of time before transiting to the IDLE state. The ACK_TO state is only visited when the end device expected an acknowledgment in one of its receive windows (i.e. after the transmission of a confirmed upstream message). In case an acknowledgment is not expected, the device transitions directly to the IDLE state from RW2.

Retransmissions in the case of end devices are handled entirely by the LoRaWANMac class. As long as the number of remaining transmissions has not expired or a downstream frame with the Ack bit set has not been received, a confirmed data packet will remain in the transmission queue. The number of transmissions for confirmed messages can be set via 'DEFAULT_NUMBER_US_TRANSMISSIONS' (in lorawan.h) and is set to four

by default. Subsequent (re)transmissions are throttled based on the radio duty cycle limitations of the active sub-band. To this end there is a per LoRaWAN node singleton object, `LoRaWANMacRDC`, which keeps track of a node's duty cycle for the different sub-bands. The `LoRaWANMAC` class also supports sending unconfirmed upstream messages more than once, as per the guidelines in the LoRaWAN standard on the number of repetitions field (`NbRep`). Finally, the `LoRaWANMac` class adds a 1B `LoRaWANMacHeader` (encoding the message type) and a 4B dummy MIC to the MAC payload before passing the packet on to the PHY.

6.4.4 LoRaWAN class A end device ns-3 application

A new ns-3 application, `LoRaWANEndDeviceApplication`, was developed to represent class A LoRaWAN end devices in ns-3. The application exposes attributes for parameters such as the data rate of the end device and the packet length and message type of upstream transmissions. It also supports configurable random variables for upstream channel selection and packet generation times. The application is responsible for generating the MAC payload, as such it adds the LoRaWAN frame header to the application payload. This frame header encodes the end device address, the packet counter and the frame port of the application. Meta data about the packet transmission - such as the desired channel, data rate and code rate - are passed on to the PHY by means of a `LoRaWANPhyParamsTag` packet tag.

6.4.5 LoRaWAN gateway ns-3 application

The `LoRaWANGatewayApplication` is a simple application that is installed on gateway ns-3 nodes. Apart from passing packets to and accepting packets from the network server, it also supports querying a gateway's RDC status from the NS. Packets that are to be sent downstream are tagged with the `LoRaWANPhyParamsTag` packet tag by the network server. The `LoRaWANNetDevice` on the gateway will select the MAC/PHY pair corresponding to the PHY attributes that are listed in the packet tag (i.e. spreading factor and channel).

6.4.6 LoRaWAN Network server

The `LoRaWANNetworkServer` class is instantiated only once per LoRaWAN network simulation. This singleton object accepts upstream packets from gateways and sends downstream traffic to end devices via gateways. It exposes the following attributes to configure downstream traffic generation: packet size, confirmed or unconfirmed messages and random variable for packet generation (an `ExponentialRandomVariable` by default). The class keeps track of information such as device address, packet counters, last data rate, last known gateway(s) and last seen time for every end device. Based on the packet counters, it can detect duplicate data packets from multiple gateways.

The network server generates downstream data and acknowledgments. To this end, it contains a per end device packet queue for storing downstream traffic. For every end device, it stores RW1 and RW2 timers that are used for scheduling downstream traffic. When a timer expires, the network server goes through the list of last known gateway(s) and searches for a gateway that can send the queued downstream packet immediately. These timers are scheduled every time an upstream transmission is processed by the network server. Finally, the network server takes care of retransmissions for confirmed downstream data packets.

6.5 Scalability analysis of LoRaWAN networks

The LoRaWAN ns-3 module includes the “lorawan-tracing-example.cc” example which was used for all simulations discussed in this section. It enables automation of simulations from a CLI by setting simulation parameters and outputting ns-3 tracing results to csv files. The simulations focused on a number of different scenarios, which are detailed in the subsections below.

All simulations consist of one, two or four gateways and a configurable number of end devices deployed in a disc with a 6 100m radius ². All gateways and end devices are configured to use the same 125kHz LoRaWAN channel (868.100MHz), with the exception of the high power RW2 channel at 869.525MHz which lies in a sub-band with a 10% RDC restriction. This single upstream channel scenario is similar to that of a “the things gateway” as sold by The Things Networks. End devices employ Activation By Personalisation and as such no network join messages are exchanged. The gateways are deployed at fixed positions, which depend on the number of gateways in the simulation. In case of one gateway, it is positioned in the origin of the disc. In case of two gateways, they are positioned one radius apart on a diameter line of the disc. In case of four gateways, they are positioned on the corners of a square which is centered on the disc origin and which has a diagonal equal to the disc radius. The gateway positions are visualized in figure 6.8. The end devices are uniformly distributed in the disc (using the UniformDiscPositionAllocator in ns-3) and have a fixed position during the simulation. For all experiments the default propagation loss model was used in ns-3. This model, named ‘LogDistancePropagationLoss’, has a 3.0 exponent at a 46.6777 dB reference loss at one meter.

Simulation scenarios are run for three different upstream data generation periods: 600, 6 000 and 60 000 seconds. Each simulation is run for a simulation time equal to hundred times the upstream data generation period. For every end device, the transmission time of the first upstream packet is picked from a random variable uniformly distributed between zero and the upstream period. Subsequent upstream packets are periodically generated according to the data generation period. Upstream packets have an application payload of 8 bytes, which implies a PHY payload of 21 bytes.

²This radius was chosen as for the presented ns-3 error model, the PDR for 21B packets sent at SF12 lies close to 10% at this distance for the LogDistancePropagationLoss model.

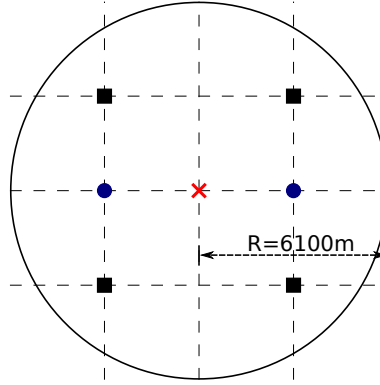


Figure 6.8: Positions for one (cross), two (circles) and four (rectangles) gateways in ns-3 simulations

Downstream data generation happens according to an end-device specific exponential random variable (representing an arrival of events, rather than periodic data transmission). The mean of this exponential random variable is set to either 60 000s or 600 000s, representing - on average - one downstream packet every ten and hundred upstream packets respectively.

For all simulations, the packet delivery ratio was measured. An unconfirmed upstream data packet is considered delivered, if it was received successfully by a gateway node. A confirmed upstream data packet is considered delivered, if one of its transmissions was successfully received by a gateway node and the end device received an acknowledgment from the network server. Note that the presented PDRs take into account all generated packets, even packets that are queued for transmission are counted towards PDR. Therefore, the PDR reflects overall network throughput (for the same number of devices and data period).

6.5.1 Assigning LoRa spreading factors to end devices

The first problem that was studied is how to assign LoRa spreading factors to end devices. Spreading factors have a major impact on packet delivery rates. Underestimating the spreading factor (i.e. assigning a SF that is too low) may lead to reception errors due to low SNR. Overestimating the spreading factor (i.e. assigning a SF that is too high) may lead to inefficient use of air time.

Three SF assignment strategies have been considered:

1. Random: assign spreading factors to end devices according to a uniform random distribution.
2. Fixed: assign the same spreading factor to end devices.
3. PER: for every end device, find and assign the lowest spreading factor for which the packet error ratio falls below a certain threshold.

For each strategy a number of simulations were performed for a six hundred seconds upstream data period and a varying number of end devices. For the PER strategy a number of different PER thresholds were tested as well: 0.001, 0.01, 0.1 and 0.25. The packet delivery ratios for the different SF allocation strategies are presented in figure 6.9. Packets are sent as unconfirmed messages.

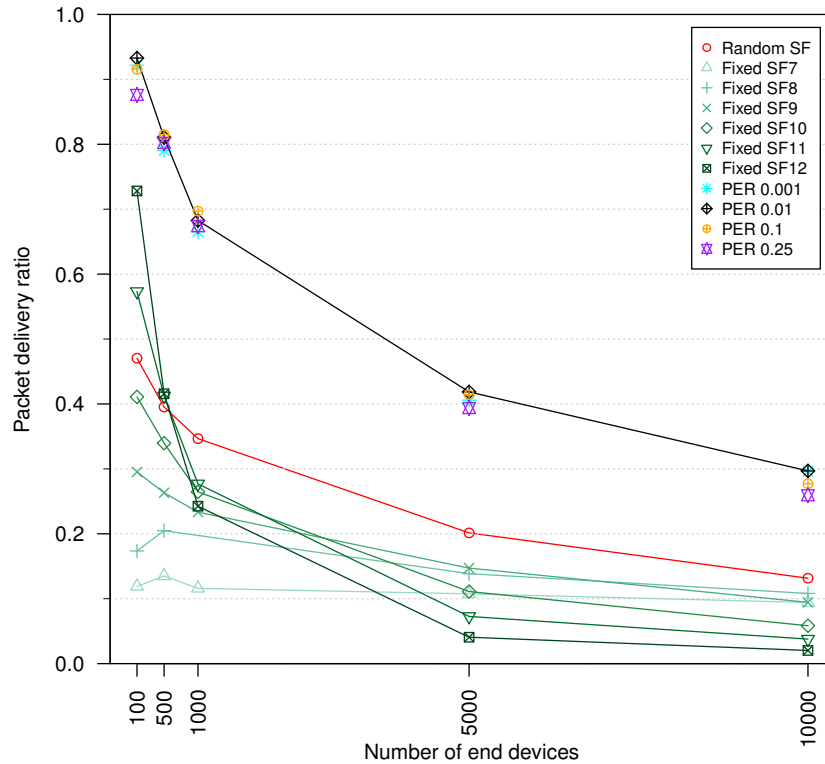


Figure 6.9: Packet delivery ratios for various spreading factor assignments strategies

Comparing the results, it is clear that the PER strategy performs the best out of three in terms of PDR. The PDRs for different PER thresholds are very similar and there is no threshold that yields the highest PDR in all considered network sizes. Note that while there exist large variations in PDRs between different spreading factors, figure 6.9 plots the global PDR across all spreading factors. A PER threshold of 0.01 is chosen for allocating spreading factors in the remainder of this paper. With this threshold, there are on average about 43% SF12, 20% SF11, 12% SF10, 8% SF9, 6% SF8 and 11% SF7 in a single gateway LoRaWAN network with radius 6 100 meters. This is represented graphically in figure 6.10.

Note that for small networks (i.e. less than 100 end devices) the 75% PDR of the fixed SF12 strategy might suffice. As this eliminates the need for active data rate (ADR) in such small networks, the downstream traffic of ADR can be avoided.

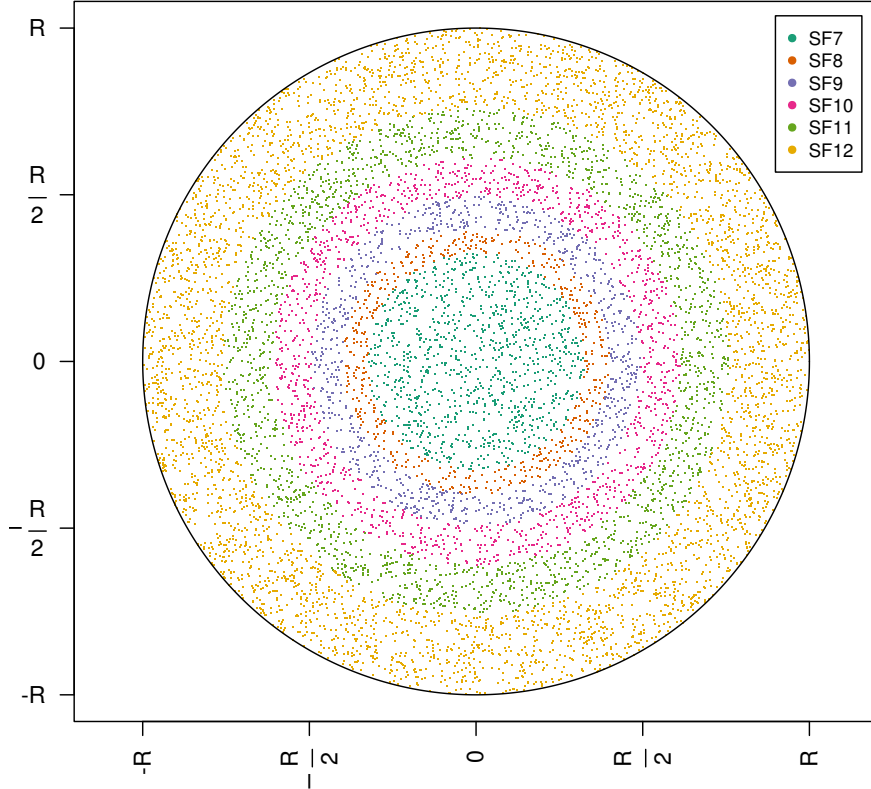


Figure 6.10: Spreading factor allocation to end devices for PER strategy (0.01)

Of course sending at an unnecessarily high data rate causes an end device to waste energy. As the number of devices increases (i.e. more than 1 000 end devices), the random strategy outperforms the fixed strategies in terms of PDR. This is explained by the fact that for larger networks the losses are mostly due to collisions and that the random strategy reduces the number of collisions (compared to a fixed strategy) by leveraging the orthogonality of the different spreading factors.

6.5.2 Unconfirmed vs confirmed upstream data

6.5.2.1 Single gateway LoRaWAN network

Next, the impact of sending upstream data as confirmed MAC messages is considered on the PDR. One would expect the LoRaWAN retransmission scheme to boost the PDR, as unacknowledged messages are retransmitted by the end device. In the case of confirmed messages, an end device attempts four transmissions before dropping the message. Unconfirmed messages are sent once for the UNC scenarios and four times for the 4UNC scenarios (i.e. $NbRep = 4$). At all times,

end devices respect duty cycle restrictions.

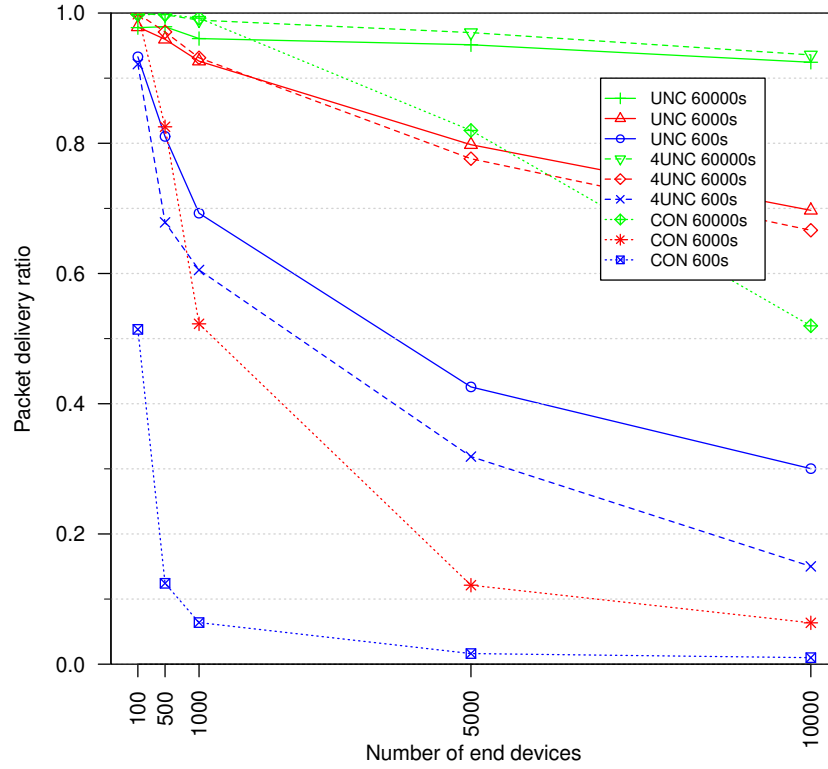


Figure 6.11: PDR for unconfirmed (UNC), $NbRep = 4$ unconfirmed (4UNC) and confirmed (CON) upstream messages in a single gateway LoRaWAN network.

The packet delivery ratios for sending upstream data as unconfirmed and confirmed messages are shown in figure 6.11 for three different data periods in case of a single gateway LoRaWAN network. The PDR decreases as data is sent more frequently and as the number of end devices increases. In case of unconfirmed MAC messages, the primary cause of undelivered packets is due to collisions where the gateway is busy receiving a transmission and therefore any other transmission with the same data rate is dropped during the ongoing reception. For the 600 seconds data period, the share of drops due to collisions in all undelivered packets is close to 90%. Another 9% of the undelivered packets are destroyed due to interference during reception. The remainder of the undelivered packets are dropped due to an SINR value that falls below the SNR cut-off point (cfr. section 6.4.1.2). While sending unconfirmed messages four times increases the PDR for low traffic scenarios, the PDR is lower in scenarios where interference becomes the limiting factor.

Note that the share of slower data rates in the undelivered packets is higher than that of faster data rates. This is partially due to the higher share of end devices with

slower data rates in the networks and partially due to the higher transmission times at lower data rates. The share of undelivered packets sent at SF11 or SF12, lies at 80.9%, 93.6% and 95.8% for the 600, 6 000 and 60 000 seconds data periods respectively.

Table 6.3: Transmission of downstream acknowledgments and upstream packets for confirmed messages in one, two and four gateway LoRaWAN network simulations.

Missed RWs: lowly saturated scenarios marked by *,
highly saturated scenarios marked by †.

GW	DP	#ED	Ack RW1	Ack RW2	Missed RWs	$\frac{\text{\#packets}}{\text{message}}$
1	60000	100	8798	1354	0 ***	1.05
1	60000	1000	47500	53162	7968 **	1.19
1	60000	10000	91950	438343	1604427 *	3.08
1	6000	100	4741	5316	1122	1.21
1	6000	1000	9542	43767	155513	3.07
1	6000	10000	17078	52033	1465697	3.90
1	600	100	943	4315	15052 †††	3.08
1	600	1000	1623	5199	143153 ††	3.90
1	600	10000	6880	5273	262385 †	3.98
2	60000	100	9896	174	0 ***	1.02
2	60000	1000	75062	24590	1026 **	1.07
2	60000	10000	248682	631806	877387 *	2.09
2	6000	100	7926	2170	0	1.03
2	6000	1000	25400	63095	84263	2.04
2	6000	10000	42798	103371	2229616	3.79
2	600	100	2513	6477	8502 †††	2.02
2	600	1000	4838	10327	216539 ††	3.77
2	600	10000	13117	10577	645143 †	3.97
4	60000	100	10012	0	0 ***	1.00
4	60000	1000	95350	4866	201 **	1.01
4	60000	10000	646058	355183	128053 *	1.17
4	6000	100	9380	656	2	1.00
4	6000	1000	66712	33302	12972	1.16
4	6000	10000	135780	201664	2433131	3.47
4	600	100	6568	3470	1360 †††	1.15
4	600	1000	14906	20085	242954 ††	3.45
4	600	10000	26866	21163	1306882 †	3.94

Somewhat counterintuitively, the PDR of confirmed messages is not always higher than that of unconfirmed messages. The PDR of CON messages is only higher than that of UNC in cases where the traffic load is very low. In the simulations this is only the case for 100, 500 and 1000 end devices for a 60 000 s data period and for 100 end devices for a 6 000 s data period. In all other cases

the PDR of confirmed messages is lower. Recall that a confirmed message is only considered delivered if the end device receives an acknowledgment for that message. Table 6.3 shows that the number of missed receive windows (for sending an acknowledgment) goes up as the traffic load increases. Receive windows are missed because the gateway is unable to transmit at the start of a receive window due to the duty cycle restrictions that apply in the sub-band of a receive window. While CON and 4UNC have similar traffic loads for saturated networks (i.e. at the highest loads, the traffic load of CON is close to that of 4UNC³), the PDR of CON messages is lower than that of 4UNC. This difference is explained by the missing downstream acknowledgments for CON messages. Finally, when a gateway sends an acknowledgment in either RW1 or RW2, all ongoing receptions at the gateway are aborted; which further decreases the PDR.

6.5.2.2 Multi gateway LoRaWAN networks

In this section the effect of the number of gateways in a LoRaWAN network on the PDR is studied. Note that for applying the PER 0.01 SF allocation strategy, the PER to the closest gateway is calculated for every end device. Increasing the gateway density, as per figure 6.8, is anticipated to have more than one effect. Firstly, it should enable higher data rates for end devices due to an increase in link budget (as on average gateways will appear closer). Secondly, as downstream transmissions in RW1 are sent with the same data rate as the upstream transmissions, RW1 acknowledgments should also profit from the higher data rates of end devices. Finally, as duty cycle restrictions apply per gateway, the LoRaWAN network should be able to acknowledge more messages as the gateway density goes up.

Table 6.4: End devices at a specific data rate for LoRaWAN networks with one, two and four gateways

GW	SF7	SF8	SF9	SF10	SF11	SF12
1	11%	6%	8%	12%	20%	43%
2	21%	10%	17%	18%	16%	18%
4	40%	16%	23%	17%	4%	0

Table 6.4 lists the fraction of end devices at specific data rates in a 10 000 end devices LoRaWAN network with one, two and four gateways (following the PER SF assignment strategy, see section 6.5.1). The table clearly illustrates that higher gateway densities lead to faster overall data rates.

Figures 6.12 and 6.13 show the PDR for a LoRaWAN network with two and four gateways respectively. Notice how for unconfirmed messages, the PDR increases greatly as the number of gateways increases. For confirmed messages, the

³Table 6.3 illustrates that as end devices retransmit more frequently, the average number of packets per confirmed message approaches four.

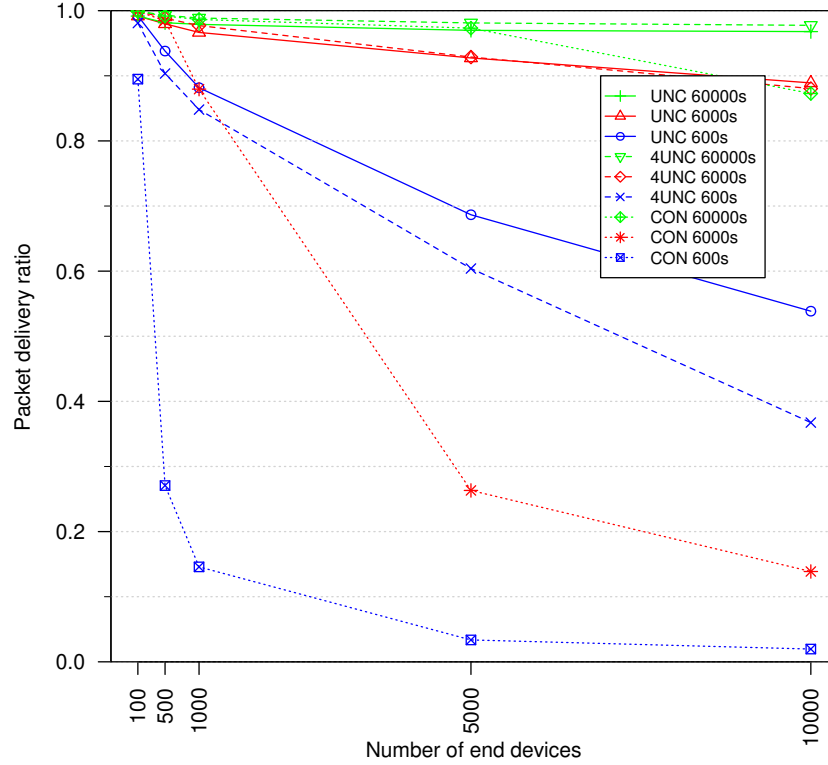


Figure 6.12: PDR for unconfirmed (UNC), $NbRep = 4$ unconfirmed (4UNC) and confirmed (CON) upstream messages in a two gateway LoRaWAN network.

increase in PDR is noticeable but is not as sharp as for unconfirmed messages. Studying table 6.3, it is clear that the number of sent acknowledgments increases as the number of gateways increases. The seemingly contradicting relation between number of missed RWs and the number of gateways is explained as follows. In saturated LoRaWAN networks (i.e. scenarios with low PDRs for unconfirmed messages), the number of sent messages that are successfully received increases with the gateway density. In case of confirmed messages, the higher number of received upstream messages means that the network server is able to identify a larger number of receive windows of end devices (as RWs are always opened after a transmission of an end device). When gateways are unable to send in these receive windows (due to duty cycle restrictions), the number of missed RWs increases. This is illustrated for the simulation scenarios marked with the \dagger symbol in table 6.3. In less saturated scenarios (marked with the $*$ symbol), the number of missed RWs goes down as increasing the gateway density does not lead to identifying more receive windows. Instead, the number of missed RWs decreases and more acknowledgments are sent (as seen in columns RW1 and RW2), which

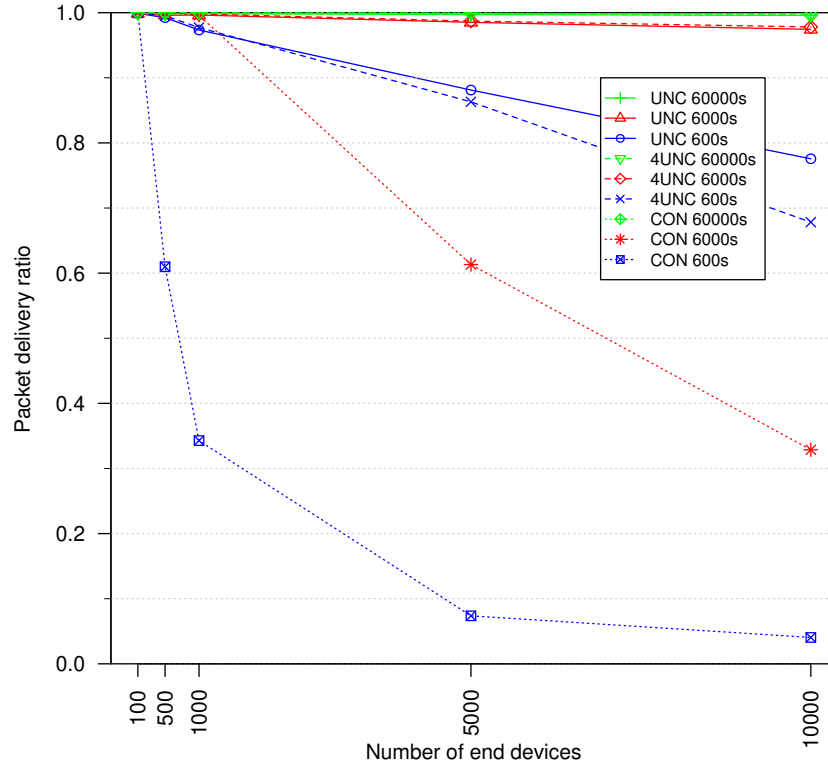


Figure 6.13: PDR for unconfirmed (UNC), $NbRep = 4$ unconfirmed (4UNC) and confirmed (CON) upstream messages in a four gateway LoRaWAN network.

benefits the PDR.

6.5.3 Downstream data traffic

In the final part of this evaluation, the impact of sending downstream data is studied. While most LoRaWAN deployments are expected to exhibit high asymmetry between the volume of upstream and downstream data, occasional downstream data messages are expected to be sent. Potential reasons for downstream data include notifying the end device of an event, end device and network management and updating application parameters (e.g. sensor sampling interval). Due to the sparseness and stochastic nature (e.g. events) of downstream data, generation of downstream data messages in ns-3 is modeled via a per-end device Poisson process with a configurable average rate λ and mean inter arrival time $\mu = \frac{1}{\lambda}$.

In terms of simulations the upstream scenario with a data period of six thousand seconds is chosen as a starting point. Two average downstream rates of one DS packet every 60 000s and 600 000s are considered, which corresponds to one DS packet every ten and hundred US packets respectively. Both confirmed and un-

confirmed downstream messages and confirmed and unconfirmed upstream messages are taken into account. Simulations were ran for one, two and four gateway networks with 100, 500, 1 000, 5 000 and 10 000 end devices. Downstream packets have a 21B size, which holds eight bytes of application payload.

Table 6.5: Packet delivery ratios of downstream data messages

Downstream Packet Delivery Ratios with 1 GW											
US	DS UNC					DS CON					μ
	98	97	94	67	40	99	97	93	59	33	10
CON	99	96	92	80	69	100	96	93	80	69	100
UNC	100	98	92	50	31	100	97	90	48	30	10
	99	98	93	63	45	100	97	92	61	44	100
Downstream Packet Delivery Ratios with 2 GWs											
US	DS UNC					DS CON					μ
	100	98	97	91	75	99	97	96	89	68	10
CON	100	98	97	93	88	99	97	97	92	87	100
UNC	100	99	98	79	59	99	99	98	78	58	10
	99	99	98	85	73	99	100	98	84	72	100
Downstream Packet Delivery Ratios with 4 GWs											
US	DS UNC					DS CON					μ
	100	100	100	98	96	100	99	99	98	96	10
CON	100	100	100	98	97	100	100	100	97	96	100
UNC	100	100	100	97	91	100	99	99	97	88	10
	100	100	100	97	94	100	100	100	97	92	100

Tables 6.5 and 6.6 present the PDRs of downstream and upstream data messages respectively for the different parameters that were tested. The five columns per quadrant in the tables represent results for 100, 500, 1 000, 5 000 and 10 000 end devices from left to right.

Studying table 6.5, the effect of saturating the available airtime at the gateway is clearly visible for simulations with one gateway and a large number of nodes (i.e. $\geq 5\,000$). As the number of gateways increases, the downstream traffic load is spread over more gateway which leads to less saturation per gateway and therefore to an increase in downstream PDR. The numbers also show that sending US data as confirmed messages negatively impacts the PDR of downstream data messages. This is because confirmed US messages require a downstream message for an acknowledgment, which increases the traffic load and therefore saturation on the gateway(s). Finally, table 6.5 also shows that for saturated scenarios the PDR for confirmed downstream messages is slightly lower than for unconfirmed downstream messages. While the cause of this is not obvious, it is probable that the 70-80% PDR of US messages in saturated scenarios (see UNC 6 000s figure 6.11) leads to losses of upstream acknowledgments which decreases downstream data PDR in the case of confirmed DS messages.

Comparing table 6.6 to the 6 000s US PDRs in figures 6.11, 6.12 and 6.13,

Table 6.6: Packet delivery ratios of upstream messages in the presence of downstream data

Upstream Packet Delivery Ratios with 1 GW												
US	DS UNC					DS CON					μ	
	98	95	90	70	60	98	95	89	70	60	10	10
CON	98	96	92	79	68	98	96	92	79	68	100	100
UNC	100	81	49	10	5	100	81	49	10	5	10	10
CON	100	83	52	12	6	100	83	52	12	6	100	100
Upstream Packet Delivery Ratios with 2 GWs												
US	DS UNC					DS CON					μ	
	99	98	96	87	80	99	97	96	87	80	10	10
CON	99	98	97	92	88	99	98	97	92	88	100	100
UNC	100	98	87	23	11	100	98	87	23	11	10	10
CON	100	98	88	26	14	100	98	88	26	13	100	100
Upstream Packet Delivery Ratios with 4 GWs												
US	DS UNC					DS CON					μ	
	100	100	99	97	94	100	100	99	97	94	10	10
CON	100	100	100	98	97	100	100	100	98	97	100	100
UNC	100	100	100	58	30	100	100	100	58	29	10	10
CON	100	100	100	61	33	100	100	100	61	33	100	100

the presence of DS data traffic leads to a negligible decrease in US PDR for low DS traffic rates ($\mu = 100$) and a small decrease in US PDR for the high DS traffic rate ($\mu = 10$) for scenarios with 5 000 and 10 000 end devices. The decrease is more profound for unconfirmed US messages than confirmed US messages, which indicates an increase in US packet loss. This increase in US packet loss is due to the gateway being unable to receive US transmissions during a DS transmission. As more gateways are deployed, the DS data transmissions occupy less time per gateway (due to overall higher data rates) which means that the gateways can spend more time listening for US messages, thereby reducing the effect of DS data traffic on US packet loss. Finally, note that there is no difference in terms of US PDR between confirmed and unconfirmed downstream data messages.

6.6 Related work

A number of works have been published in literature that study the scalability of LoRa(WAN) LPWA networks.

In one of the first works on this topic, Mikhaylov et al. [17] present an analysis of the capacity and scalability of LoRa LPWANs. The authors perform an analytical analysis of the maximum throughput for a single LoRaWAN end device, taking into account such factors as RDC and the influence of receive windows. The authors note that receive windows drastically increase the time between subsequent transmissions and that RDC restrictions reduce the maximum through-

put further. The authors applied the same methodology to determine the capacity of LoRaWAN based on ALOHA access. While it is true that the LoRaWAN MAC access is an ALOHA scheme, empirical data has shown that the assumptions made in pure ALOHA access do not adequately model a LoRaWAN network (see figure 4 in [18]). Specifically, it fails to model the interference between concurrent transmissions as pure ALOHA assumes concurrent transmissions are always lost regardless of their received power levels, timings and the presence of forward error correction. A second, but similarly lacking, pure ALOHA capacity analysis of LoRaWAN is discussed in [19]. In [20], Adelantado et al. also calculate LoRaWAN capacity as the superposition of independent ALOHA-based networks (one for each channel and for each SF). In conclusion, analyses based on pure ALOHA fail to adequately model interference in LoRaWAN networks and therefore underestimate the capacity of LoRaWAN LPWANs.

In [21], Georgiou and Raza provide a stochastic geometry framework for modeling the performance of a single channel LoRa network. Two independent link-outage conditions are studied, one which is related to SNR (i.e. range) and another one which is related to co-spreading factor interference. The authors argue that LoRa networks will inevitably become interference-limited, as end device coverage probability decays exponentially with increasing number of end devices. The authors report that this is mostly caused by co-spreading factor interference and that the low duty cycle and chirp orthogonality found in LoRa do little to mitigate this. Finally, the authors note that the lack of a packet-level software simulation is hindering the study into the performance of LoRa. It would be interesting to combine the authors' modeling of co-spreading factor interference with our ns-3 error model, as in the SINR approach all interference is treated as noise.

Reynders et al. [22] analyze the range and coexistence of two long range unlicensed communication technologies: ultra narrowband (i.e. BPSK, as used by Sigfox) and wideband spread spectrum (i.e. CSS, as used by LoRa). Through physical layer simulation, the authors find that ultra narrowband networks have larger coverage, while wideband spread spectrum networks are less sensitive to interference. The authors also extract closed form expressions for the BER of UNB and CSS from these simulations. Unfortunately, the presented results do not take into account the effects of forward error correction (as used in LoRa). Reynders et al. further compare CSS to UNB networks in ns-3 by investigating the impact of the MAC protocols used in Sigfox and LoRaWAN. Their results show that CSS networks offer higher throughput, while UNB networks support a larger number of devices (note that the analysis assumes saturated traffic conditions for CSS devices). From the manuscript the level of detail of the MAC models is unclear: e.g. the authors do mention implementing downstream acknowledgments and data rate management but other features are not discussed. In contrast, the presented LoRaWAN ns-3 module fully supports class A end devices and provides a simple network server. Finally, the authors stress the importance of data rate and frequency management in wideband networks in order to limit the contention and interference between devices. Indeed, section 6.5.1 shows that naive SF assignment strategies can severely diminish the PDR.

The work of Bor et al. [18] studies the limit on the number of transmitters supported by a LoRa system based on an empirical model. The authors performed practical experiments that quantify the communication range and the capture effect of LoRa transmissions. These findings were used to build a purpose-built simulator, LoRaSim, with the goal of studying the scalability of LoRa networks. The authors conclude that LoRa networks can scale quite well if they use dynamic transmissions parameter selection and/or multiple sinks. Our study confirms that multiple sinks drastically improve scalability, even though we use a very different approach for modeling interference. Furthermore, our study goes deeper into modeling LoRaWAN as the LoRaWAN MAC layer is modeled and the impact of confirmed messages and downstream traffic is studied.

The recent work presented by Pop et al. in [23] studies the impact of bidirectional traffic in LoRaWAN by extending the LoRaSim simulator to include bidirectional LoRaWAN communication. The resulting simulator is named LoRaWANSim. Both our ns-3 module and LoRaWANSim allow to study the scalability of LoRaWAN networks. Both works find that duty cycle limitations at the gateway limit the number of downlink messages (Ack or data) a gateway can send. This problem grows worse as the end device density increases, but can be partially mitigated by increasing gateway density (see section 6.5.3). The authors of [23] correctly identify that the absence of an acknowledgement does not necessarily mean that the link quality has decreased and that a node should decrease its data rate for subsequent retransmissions. Actually, decreasing the data rate might exacerbate this problem as detailed in [23]. Notable differences between the two simulators include that the LoRaWANSim manuscript is limited to single gateway network, while the ns-3 module provides support for multi-gateway LoRaWAN networks. Secondly, the collision models are quite different. The ns-3 module builds on the error model derived from the complex baseband BER simulations, while LoRaWANSim reuses the empirical model from LoRaSim. Both collision models support the capture effect as well as modeling interference. Under capture effect, we understand the ability to receive an interfered transmission in the presence of one or more interferers as long as the SNR of the interfered transmission is sufficiently high for the transmission to be received error-free. The LoRaWANSim collision model incorrectly assumes perfect orthogonality between spreading factors, while the ns-3 module counts every transmission on the same channel with a different spreading factor as interference. Furthermore, the LoRaWANSim manuscript does not mention the 10% RDC restriction that applies in the sub-band of the RW2 channel in the EU. This underestimates the downlink capacity in RW2. Thirdly, the SpectrumPhy model for the LoRa PHY in ns-3 enables modeling inter-technology interference, which could facilitate studies on the interference between 802.11ah on LoRaWAN. Finally, the LoRaWANSim simulator does not appear to be open source although the manuscript is still under revision at this time.

In another recent work, Magrin et al. [24] evaluate the performance of LoRa networks in a smart city scenario. The authors propose a link measurement and a link performance model for LoRa and implement a LoRaWAN system-level simulator in ns-3. Their results show that LoRaWAN provides a higher throughput than

a basic ALOHA scheme and that LoRaWAN networks scale well as the number of gateways increases. Our work confirms these two results, while also studying the impact of downstream traffic. Modeling the LoRa performance by means of the SINR threshold matrix presented in [24] is quite different than our AWGN BER performance model of LoRa. As both studies have their merits, it would be interesting to try and merge both works into a single lorawan module in ns-3 in the future.

6.7 Discussion

In this section a number of findings from our scalability analysis in section 6.5 are discussed. The results show that confirmed messages severely impact the packet delivery ratios of upstream messages. While increasing the number of gateways helps to alleviate this problem somewhat, the results of the six hundred seconds data period show that the PDR remains low even in a four gateway network. The impact of downstream data messages on upstream messages was found to be negligible due to the sparseness of the tested downstream data traffic load. Additionally, little difference was found between sending downstream data as unconfirmed messages vs confirmed messages in terms of the DS PDR. Only for the single gateway and $\mu = 10$ scenario a significant difference was found.

As every study has its limitations, a number of points that could be improved as part of future work are discussed here. As discussed in the related work study, the approach of modeling all interference as noise has its drawbacks. Specifically, literature has shown that while interference between different spreading factors can be accurately modeled as noise, co-spreading factor interference may be modeled more accurately via a stochastic approach. Future studies may opt to fine-tune the path loss model in ns-3 in order to more closely match the radio environment under study. An interesting point for future work is to study the impact of the downstream data rate in RW2. By default, this is set to the lowest data rate in the LoRaWAN standard. However when downstream data messages are not delivered due to RDC limitations (rather than low link quality), a faster RW2 data rate might increase the capacity of the LoRaWAN. Another interesting research topic would be to introduce structure to the LoRaWAN medium access. While this will come at a cost in terms of traffic overhead and power consumption, it might lead to higher network capacity by reducing interference. Additional MAC features such as adaptive data rate (ADR) and network management (e.g. joining) could be added to the ns-3 module. This would allow a more in-depth study of the LoRaWAN standard. Extending the lorawan module to track the energy usage of end devices would enable future studies to take power consumption into consideration. For example, studying the trade-offs between energy usage, reliability and network scalability when sending unconfirmed messages multiple times (i.e. LoRaWAN NbTrans > 1) would be an interesting topic. Finally, it would be interesting to study how LoRaWAN networks are affected by the presence of other sub-GHz (LP)WAN technologies.

6.8 Conclusion

In this work, a comprehensive model of LoRaWAN LPWANs in the ns-3 network simulator is presented. This model includes an error model used for determining range as well as interference between multiple simultaneous transmissions. All spreading factors and code rates found in LoRaWAN are supported by the PHY layer model of LoRa in the ns-3 module. The ns-3 module models the MAC layer for class A end devices and supports both upstream and downstream (un)confirmed messages via a simple network server. Furthermore, LoRaWAN networks with multiples gateways are supported.

The ns-3 module forms the basis for a scalability analysis of single channel multi gateway LoRaWAN LPWANs. The results of this analysis show that allocating network parameters to end devices is hugely important for the performance of LoRaWAN networks. Furthermore, the capacity for different types of traffic is studied. The results confirm recent findings from literature that the limited downstream capacity highly deteriorates the packet delivery ratio of confirmed upstream messages. Increasing the gateway density can delay the onset of this effect, but it cannot be eliminated completely. Finally, it is the hope of this work to encourage future work on all aspects of LoRaWAN networks by means of the publicly available ns-3 module. To this end, a number of interesting topics are presented as well.

Acknowledgment

This work was carried out in the context of following projects. MoniCow is a project realized in collaboration with imec. Project partners are DeLaval, Metagam, Multicap, NXP Semiconductors N.V. and snapTonic, with project support from VLAIO (Flanders Innovation & Entrepreneurship). IDEAL-IoT (Intelligent DENSE And Long range IoT networks) is an SBO project funded by the Fund for Scientific Research-Flanders (FWO-V) under grant agreement #S004017N. ‘Processing visual sensor data in low-power wide area networks’ is a project funded by the Fund for Scientific Research-Flanders (FWO-V) under grant agreement #G084177N.

References

- [1] P. Middleton, T. Tully, J. F. Hines, T. Koslowski, B. Tratz-Ryan, K. F. Brant, E. Goodness, A. McIntyre, and A. Gupta. *Forecast: Internet of Things - Endpoints and Associated Services, Worldwide, 2015*, 2015. Available from: <https://www.gartner.com/doc/3159717/forecast-internet-things--endpoints>.
- [2] U. Raza, P. Kulkarni, and M. Sooriyabandara. *Low Power Wide Area Networks: An Overview*. arXiv:1606.07360v2 [cs.NI], 2017. Available from: <http://arxiv.org/abs/1606.07360>.
- [3] Cisco. *Cisco visual networking index: Global mobile data traffic forecast update 2015-2020*. Technical report, Cisco, 2016. Available from: https://www.cisco.com/c/dam/m/en/_in/innovation/enterprise/assets/mobile-white-paper-c11-520862.pdf.
- [4] Nokia White Paper. *LTE evolution for IoT connectivity*. Technical report, Nokia, 2016. Available from: <http://resources.alcatel-lucent.com/{%}0Asset/200178>.
- [5] L. Alliance. *LoRaWAN Specification*. LoRa Alliance, 2015.
- [6] Proximus NV. *LoRa network coverage map (January 2017)*. Available from: <https://www.proximus.be/resources/iportal/ect/maps/coverage-en.html{#}lora>.
- [7] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena. *Network simulations with the ns-3 simulator*. SIGCOMM demonstration, 14(14):527, 2008.
- [8] M. M. S. Kowsar and S. Biswas. *Performance improvement of IEEE 802.11n WLANs via frame aggregation in NS-3*. In 2017 International Conference on Electrical, Computer and Communication Engineering (ECCE), pages 1–6. IEEE, feb 2017. Available from: <http://ieeexplore.ieee.org/document/7913039/>, doi:10.1109/ECACE.2017.7913039.
- [9] L. Tian, S. Deronne, S. Latré, and J. Famaey. *Implementation and Validation of an IEEE 802.11ah Module for ns-3*. In Proceedings of the Workshop on ns-3 - WNS3 '16, pages 49–56, New York, New York, USA, 2016. ACM Press. Available from: <http://dl.acm.org/citation.cfm?doid=2915371.2915372>, doi:10.1145/2915371.2915372.
- [10] O. B. A. Seller and N. Sornin. *Low power long range transmitter*, 2014. Available from: <https://www.google.com/patents/EP2763321A1?cl=fi>.
- [11] M. Knight and B. Seeber. *Decoding LoRa: Realizing a Modern LPWAN with SDR*. Proceedings of the GNU Radio Conference, 1(1), 2016. Available from: <http://pubs.gnuradio.org/index.php/grcon/article/view/8>.

- [12] P. Robyns, E. Marin, W. Lamotte, P. Quax, D. Singelée, and B. Preneel. *Physical-Layer Fingerprinting of LoRa devices using Supervised and Zero-Shot Learning*. In Proceedings of the 10th ACM Conference on Security & Privacy in Wireless and Mobile Networks, Boston, MA, USA, 2017. doi:10.1145/3098243.3098267.
- [13] N. Baldo and M. Miozzo. *Spectrum-aware Channel and PHY layer modeling for ns3*. In Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools, page 2. ICST, 2009. Available from: <http://eudl.eu/doi/10.4108/ICST.VALUETOOLS2009.7647>, doi:10.4108/ICST.VALUETOOLS2009.7647.
- [14] M. Lacage and T. R. Henderson. *Yet another network simulator*. In Proceeding from the 2006 workshop on ns-2: the IP network simulator - WNS2 '06, page 12, New York, New York, USA, 2006. ACM Press. Available from: <http://portal.acm.org/citation.cfm?doid=1190455.1190467>, doi:10.1145/1190455.1190467.
- [15] G. Piro, N. Baldo, and M. Miozzo. *An LTE module for the ns-3 network simulator*. In Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, page 527. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011. Available from: <https://dl.acm.org/citation.cfm?id=2151129>.
- [16] T. Henderson. *Low-Rate Wireless Personal Area Network (LR-WPAN) - Model Library ns-3*, 2011. Available from: <https://www.nsnam.org/docs/models/html/lr-wpan.html>.
- [17] K. Mikhaylov, J. Petaejaervi, and T. Haenninen. *Analysis of Capacity and Scalability of the LoRa Low Power Wide Area Network Technology*. In European Wireless 2016; 22th European Wireless Conference, pages 1–6, 2016.
- [18] M. C. Bor, U. Roedig, T. Voigt, and J. M. Alonso. *Do LoRa Low-Power Wide-Area Networks Scale?* In Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pages 59–67, 2016. Available from: <http://doi.acm.org/10.1145/2988287.2989163>, doi:10.1145/2988287.2989163.
- [19] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley. *A Study of LoRa: Long Range & Low Power Networks for the Internet of Things*. Sensors, 16(9), 2016.
- [20] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia, and T. Watteyne. *Understanding the limits of LoRaWAN*. IEEE Communications Magazine, pages 8–12, 2016. arXiv:arXiv:1607.08011v1.

- [21] O. Georgiou and U. Raza. *Low Power Wide Area Network Analysis: Can LoRa Scale?* IEEE Wireless Communications Letters, 2017.
- [22] B. Reynders, W. Meert, and S. Pollin. *Range and coexistence analysis of long range unlicensed communication*. In 2016 23rd International Conference on Telecommunications (ICT), pages 1–6, Thessaloniki, Greece, 2016. IEEE. Available from: <http://ieeexplore.ieee.org/document/7500415/>, doi:10.1109/ICT.2016.7500415.
- [23] A.-I. Pop, U. Raza, P. Kulkarni, and M. Sooriyabandara. *Does Bidirectional Traffic Do More Harm Than Good in LoRaWAN Based LPWA Networks?* arXiv:1704.04174v1 [cs.NI], 2017. Available from: <https://arxiv.org/pdf/1704.04174.pdf>, arXiv:1704.04174.
- [24] D. Magrin, M. Centenaro, and L. Vangelista. *Performance Evaluation of LoRa Networks in a Smart City Scenario*. In 2017 IEEE International Conference on Communications (ICC), Paris, 2017. IEEE.

7

Conclusions and perspectives

When asked about his prediction for the future of the web, Eric Schmidt replied the following: “I will answer very simply that the Internet will disappear. There will be so many IP addresses, so many devices, sensors, things that you are wearing, things that you are interacting with that you won’t even sense it. It will be part of your presence all the time. Imagine you walk into a room, and the room is dynamic. And with your permission and all of that, you are interacting with the things going on in the room.”

– Eric Emerson Schmidt (1955 -)

The words of Eric Schmidt allude to the ubiquity of Internet technology in our lives and how the Internet of Things (IoT) will lead to an immense growth in pervasive computing. This omnipresence of computing, sensing, communication, etc. is a common theme among the many visions that exist of the IoT. Computing is gearing up for this pervasive future by the ever decreasing cost of embedded systems, the continuing miniaturization and integration of Integrated Circuits (ICs) and the increasing complexity of embedded software. The same applies to connectivity, with novel network technologies such as Low Power Wide Area Networks (LPWANs) and low-power cellular networks hitting the market. Industry has not been dormant either, as innovation drives products to be Internet-connected which has led to many (new) tech suppliers offering IoT solutions to small and large business.

Another interesting quote about the future of the IoT comes from Bob Harden, a senior IT consultant, and states: “*purchasing 10-20 different services from 10-20 different vendors using 10-20 different apps with 10-20 different user interfaces. If that’s the way IoT goes, it will be a long tough slog to Nirvana*”. Harden warns

about what is sometimes referred to as the ‘Intranet of Things’, where things only talk to other things of the same vendor and ignore other systems in their surroundings. Such an approach is problematic, as in the broad scope of the IoT many products from different vendors will co-exist and failure to interoperate will severely restrict the abilities of IoT systems as functions, data, networks, etc. would not be shareable between different systems. For many recently released IoT products, this ‘Intranet of Things’ is exactly what has happened with the proliferation of vertical stacks of integrated products. All of the (big) tech companies have tried to gain a competitive advantage by developing their individual and proprietary operating systems, equipment and protocols. As a result, an IoT product is part of a closed ecosystem and every user is limited to the walled garden of their chosen vendor. Unfortunately, this fragmentation in IoT is likely to continue to exist for a while.

Nevertheless, open technology - in the form of open protocols, data formats, open source software and open frameworks - is gaining popularity among more and more companies and developers. AllSeen, initially launched by Qualcomm and now developed by the AllSeen alliance, is an example of an open source software framework that makes it easy for devices and apps to discover and communicate with each other. Similarly, Ikea has recently launched a smart lighting solution, Trådfri, which builds on open standards (DTLS, CoAP and OMA LWM2M specifically) for implementing device management and interactions.

This dissertation is situated against the backdrop of moving from closed and proprietary IoT systems to open and interoperable IoT systems. As such the main goal of this work has been to remove barriers in the adoption of open standards in Constrained RESTful Environments (CoRE), which is the subset of the IoT that focuses on resource-constrained devices employing the Representational state transfer (REST) paradigm. In this subset, diverse things host RESTful web services with open data formats which dramatically improves integration and interoperability, which is why this is sometimes referred to as the Web of Things (WoT). This PhD has approached these adoption barriers from a number of different angles. Firstly, we have studied issues related to efficient resource usage in an open WoT as one has to be mindful of the impact of deploying open standards on the limited availability of resources in constrained devices and networks. Usability of CoRE for end users and software developers is a second important topic addressed in this dissertation, where the lack of (user) interfaces, security primitives and functionality are important barriers to adoption. Heterogeneity in connectivity technologies, communication models and patterns in resource-constrained systems built on open standards is the third subject handled in this dissertation. Finally, as open standards gain traction in new constrained networks, namely LPWANs, designed for massive IoT deployments, it is interesting to study the scalability and reliability of these LPWANs. The efforts lead by the IETF IPv6 over Low Power Wide-Area Networks (Ippwan) working group to bring open standards to these types of networks, indicate that in the future LPWANs will be part of the open WoT and as such will face issues similar to the ones studied here. The remainder of this chapter summarizes the most important work and the main conclusions of this PhD research. It is closed with potential directions for future work.

7.1 Summary and conclusions

Throughout this dissertation we have applied the concept of Distributed Intelligence (DI) to overcome adoption barriers of open standards in CoRE. DI recognizes that processing and communication are pervasive and that they may reside anywhere on the Internet. By deploying functionality on distributed systems effectively, we may attain an intelligence of sorts that is spread over multiple computing systems (hence the term). In the context of an open, resource-constrained WoT, resource-constrained devices and networks are limited in the functionality they can implement and offer. Therefor, DI is an effective concept as it helps to overcome the limitations of these resource-constrained systems by relying on more capable systems distributed over the Internet. As such, this dissertation has applied DI to address limitations specifically related to deploying and using open standards in CoRE. Specifically, this dissertation has applied Sensor Function Virtualization (SFV), a technique where constrained devices are extended with new functionality through device virtualization, to overcome these limitations.

The Secure Service Proxy (SSP) presented in chapter 3, applies device virtualization as a means towards realizing DI. Virtualizing resource-constrained devices on more capable systems allows the SSP to extend devices with new functionality and capabilities. While this dissertation has focused device virtualization on improving the performance, scalability, usability and security of CoRE, the concept of device virtualization may be applied to other problems as well. A modular system, named adapters, has been presented that implements such performance, scalability or usability enhancing functionality. Adapters process CoAP requests and generate CoAP responses, thereby supporting any functionality to be deployed on virtual devices. This dissertation has designed and implemented adapters for offering virtual resources, enforcing congestion control policies, filtering CoAP requests according to Access Control Lists (ACLs), caching responses, rewriting discovery responses and proxying requests as a reverse proxy or a CoAP mirror server. Additionally, the presented work has applied device virtualization for improving security in CoRE. As virtual devices are deployed on more capable systems, they can support strong security primitives unattainable in resource-constrained systems. Specifically, virtual devices are able to offer Public Key Infrastructure (PKI) cipher suites which provide stronger and more scalable authentication when compared to Pre-Shared Key (PSK) or Raw Public Key (RPK) cipher suites. By additionally employing a long-lived secure session between the virtual device and the constrained device, some of the overhead issues with session initialization in conventional Datagram Transport Layer Security (DTLS) are circumvented. Evaluation shows that the SSP reduces the load on constrained devices by response caching and avoiding unnecessary traffic in the constrained network. Additionally, it also improves response times in constrained networks by employing long-lived security sessions. Finally, the concept of device virtualization has also been applied to LoRa Wide Area Networks (LoRaWANs) in appendix A. It shows that the reverse proxy approach can hide LoRaWAN-specific integration APIs and offer standard-based interfaces for data access and control of LoRaWAN

end devices. This may aid in the integration of LoRaWAN networks in the WoT.

This thesis has also applied the concept of SFV to improving user interactions with constrained devices. As end users will be confronted with many, different constrained devices in their environment, it is, as mentioned by Bob Harden, unfeasible to use one mobile app per end device. Instead, users expect intuitive and uniform ways to interact with devices. As web technology is widely supported, this work proposes a web template based system for rendering Graphical User Interfaces (GUIs). In this approach users are offered rich, responsive web pages that render representations of RESTful resource and that enable users to update RESTful resources. The GUI system was demonstrated for rendering discovery responses, where web links were rendered in a table, for temperature reading, rendered in a time series graph, and for controlling a light switch, rendering as a button. As web templates allow scripting, they can also do more than just render static information. One demonstrated example was the use of non-blocking response retrieval via asynchronous javascript (ajax) in order to speed up the template rendering.

In the context of heterogeneous IoT technologies, a cloud platform for integrating heterogeneous devices and communication models has been proposed in chapter 5 of this dissertation. The viewpoint of service developers wanting to combine diverse IoT technologies is adopted. To achieve this integration, the design of the platform is split into two layers: an abstraction and an access layer. The abstraction layer offers a uniform interface for interacting with the heterogeneous devices on the platform, as such this layer abstracts the heterogeneity for service developers. The access layer integrates with technology-specific interfaces and data models, this is where the mapping is made between a specific technology and the device abstraction of the platform. Three testing scenarios evaluated different aspects of the cloud platform. One scenario shows how two different communication models (push and pull) are abstracted by the platform. In another scenario, four different IoT products, employing different connectivity technologies and protocols, are abstracted by the platform. The evaluation further shows that it is relatively straightforward to build a control and management dashboard on top of the proposed platform using conventional web technology that is well-known to service integrators.

The final topic addressed in this doctoral thesis is the scalability of emerging LPWANs. A comprehensive model of LoRaWAN networks was built in ns-3, a network simulator, supporting class A end devices sending (un)confirmed upstream messages to a network server and vice versa. In the LoRa physical layer model, interference is modeled as noise by applying the well-known signal-to-interference-plus-noise ratio (SINR) modeling technique. To this end, an error model was constructed from baseband bit error rate simulations for different signal-to-noise ratios (SNR) and modulation settings of the LoRa signal. Extensive simulations have uncovered that the radio duty cycle restrictions of gateways are detrimental to the delivery ratio of confirmed messages, due to the inability to send timely downstream acknowledgments. The lack of acknowledgments leads to end devices, which assume that the transmission was not received correctly,

unnecessarily retransmitting messages which leads to an increase in congestion and wasting energy. While increasing the gateway density does show a significant increase in packet delivery ratios of confirmed messages, it can not compensate completely for the stringent duty cycle restrictions of gateways.

To conclude, this dissertation has presented several methods for improving the integration of resource-constrained devices into applications and services. The research focused on the topics of efficiency, usability and scalability of resource-constrained environments. The dissertation can be approached as a cooking book with extensive recipes to address operational concerns of Constrained RESTful Environments. Specifically, the distributed intelligence concept and the device virtualization approach are the two cornerstones of this work. This dissertation describes and shows the value they can add in (secure) CoRE by filtering (unwanted) traffic, combining resource requests, adding virtual resources and other functionality, improving authentication and reducing resource usage. In doing so, this PhD has made a modest but significant contribution to the goal of an open, secure WoT, where many, heterogeneous (resource-constrained) devices co-exist and interoperate.

7.2 Outlook

The fragmented state of today's IoT market is threatening its exponential growth and limiting innovation. This situation is comparable with the early days of the Internet, where many non-interoperable networking technologies competed for market share. In order to address this fragmentation, IoT vendors ought to adopt open technology that target interoperability. In the near future, the World Wide Web Consortium (W3C) will continue to standardize web technology for things in its WoT working group. It is hoped that the role of open web standards in the development of the IoT will be similar to that of web technology in the development of the widely used World Wide Web (WWW). On the networking side, the 6lo IETF working group will continue the adoption of Internet Protocol version six (IPv6) in new link layer technologies (e.g. RFCs on IPv6 over Bluetooth Low Energy and DECT have been released already). On the protocol side, the healthy and active state of the CoRE IETF working group is a testament of the global interest in working on a RESTful architecture for resource-constrained devices and networks. The future will tell whether all these different initiatives can break the fragmentation of the IoT and lead to an open WoT.

Apart from the promising future of the WoT, we also expect the concept of distributed intelligence to gain importance. Already today, a lot of research is ongoing on the role of cloud computing and fog computing in the IoT.

One example is the growing interest in increasing the flexibility of constrained devices, which is apparent from adopted IETF drafts that describe resource linking and observe parameters. Both mechanisms provide novel ways for shaping direct, RESTful interactions between clients and servers. Similarly, the WoT W3C working group is looking into scripting APIs for use on Things. Such a standard-

ized API means that Things could be extended, at run time, with new functionality which has parallels to the adapter approach for extending virtual devices presented in this dissertation. Next to scripting, there is also ongoing research on the efficient (partial) reprogramming of resource-constrained devices.

Network intelligence may be considered as another form of distributed intelligence and is also on the rise in recent years. One example is the move from random access networks to networks with deterministic access for certain IoT use cases. The time-slotted channel hopping mode of IEEE 802.15.4e is a popular example that has been around for many years. Despite its age, efforts to bring IPv6 to TSCH have only begun recently at the IETF with the formation of the 6tisch working group in 2014. Nevertheless, the work at 6tisch is promising as it brings standards-based solutions for organizing channel access in 802.15.4e networks in a sector that has historically been dominated by closed standards such as WirelessHART and ISA100.11a. The novel IEEE 802.11ah (HaLow) Wi-Fi standard introduces another example of network intelligence with the concept of 'Restricted Window Access' (RAW) for structuring wireless medium access. By grouping stations, limiting contention to the stations in a group and limiting medium access to one group at a time, RAW decreases the collision probability due to congestion and increases the maximum throughput in dense networks. For both these technologies, it is necessary to build strategies that use the mechanisms provided by the standards for structuring medium access to the requirements of a specific deployment. Research is undergoing on how such strategies might look like and how the requirements of dynamic networks can be learned. An interesting enabler for network intelligence is the development of RESTful interfaces for managing constrained devices and networks at the IETF. This effort, CoAP Management Interface (CoMi), by the CoRE WG defines network access to management information embedded in YANG data models through CoAP resources. It should help to manage resource-constrained devices and networks (where traditional management protocols are unsuited) by retrieving operational information about the network which may be used as input to network intelligence methods.

Finally, in the world of Low Power Wide Area Networks (LPWANs) the move to IPv6 is also underway with the standardization of compression and fragmentation techniques in the IETF lpwan working group. This evolution could lead to LPWANs joining the WoT paradigm as standardized by the W3C, thereby greatly increasing their potential for integration. This also means that many of the existing research into the resource-constrained WoT could be applied to the LPWAN world, with the caveat of the very low throughput and datagram sizes and high delays common in LPWANs. A second interesting topic is replacing ALOHA with alternatives that structure medium access in order to improve the scalability of LPWANs. Promoting alternatives Another interesting evolution pertains to the medium access of LoRaWAN gateways. Recently, there have been tests with hardware-assisted 'Listen Before Talk' (LBT) medium access on gateways. If LoRaWAN gateways implement LBT, then this could drastically increase the downstream capacity of LoRaWANs and thereby mitigate some of scalability issues raised in this dissertation. Multimodal wireless communication is another

promising trend for LPWAN devices. Such devices combine multiple wireless connectivity technologies in order to get the best of multiple worlds, e.g. they may benefit from the high range offered by LPWANs and switch to high data rate communication at specific locations in order to perform firmware updates or other high throughput tasks. The concepts presented in this dissertation (e.g. device virtualization) may be applied to hide which communication technology a device is using and to offer seamless network access to a device regardless of its active connectivity technology.



Integrating LoRaWAN networks into the Web of Things via device virtualization

This appendix presents unpublished work on integrating LoRaWAN end devices into the Web of Things (WoT) via device virtualization. It can be considered as an extension to Chapter 5, with LoRa Wide Area Network (LoRaWAN) as an example of network heterogeneity.

A.1 Introduction

One issue with the LoRaWAN specification is the sharing of data in a distributed system. Messages from LoRaWAN end devices are received by gateways, which forward all messages to the LoRaWAN Network Server (NS). At the NS messages are deduplicated, acknowledged (if necessary) and decrypted. However, the LoRaWAN standard does not specify how data at the NS should be shared outside of the LoRaWAN infrastructure. It also does not specify how the NS should accept traffic that is to be sent downstream towards end devices. One popular option in existing LoRaWAN networks is a message brokers where messages from end devices are published and downstream messages may be queued. Troublesome issues with this approach include that it is difficult to discover the available end devices and that every deployment has its own variation on the message broker pattern.

Instead, we propose to employ open standards to facilitate this sharing of data in a web service oriented architecture. One benefit of open standards is the wide availability of interoperable software that can interact with open standard-based interfaces. Additionally, open standards typically follow a certain philosophy (or method) for designing interfaces. This means that the resulting interface is familiar to those experienced with the standard (e.g. RESTful philosophy). Finally, open standards also free us from re-inventing the wheel, allowing us to focus on the integration.

Our approach follows the RESTful paradigm as applied in popular web technology nowadays. It dictates a clear compartmentalization of functionality into multiple services. We have chosen the Constrained Application Protocol (CoAP) as it is popular in resource-constrained environments for developing RESTful web services. Benefits of CoAP include that it is highly usable in resource-constrained environments and that many requirements of M2M communication have been taken into account during the design phase of the protocol. Additionally, CoAP was also designed to support easy discovery of services and to interface easily with HTTP.

A.2 RESTful Web services for data sharing and control of LoRaWAN end devices

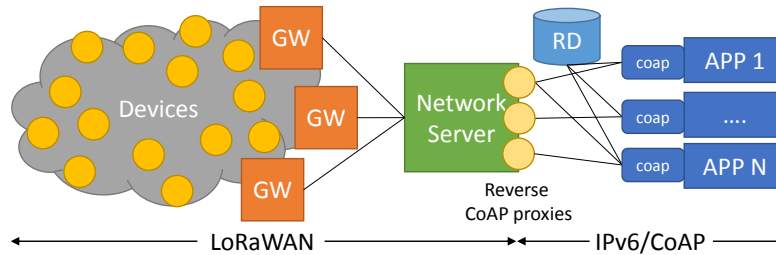


Figure A.1: LoRaWAN end devices are abstracted as virtual CoAP servers to facilitate data exchange and control in the WoT

Figure 1 presents our approach for sharing upstream messages and accepting downstream messages between LoRaWAN end devices (on the left) and applications external to the LoRaWAN infrastructure (on the right). The approach hosts a CoAP server on behalf of every LoRaWAN end device in the network, which is known as a reverse proxy approach. Upstream application data from an end device is offered as a CoAP resource on the reverse proxy corresponding to the end device. One CoAP resource is hosted per LoRaWAN FPort in use. This enables parties with interest in the upstream data to observe the upstream resource on the CoAP proxy and retrieve upstream messages. Similarly, the reverse proxy hosts a CoAP resource that accepts downstream messages for the end device in question.

Once the LoRaWAN end device opens a receive window, the value of this resource is retrieved and sent to the end device as a LoRaWAN frame payload.

The reverse proxy is responsible for the communication with the the Network Server and as such is dependent on the LoRaWAN deployment. In our case, we used the ‘The Thing Networks’ LoRaWAN network which hosts a message broker as its main interface. On this message broker there are different message topics for upstream and downstream traffic per end device. The reverse proxy subscribes for upstream messages at a topic exchange. When a LoRaWAN message is received at the network server, it is published to the upstream topic corresponding to the end device. The reverse proxy receives the message and updates the CoAP upstream resource of the CoAP server corresponding to the end device. For downstream traffic, the reverse proxy publishes CoAP resource representations on the downstream topic corresponding to the end device. This enables the NS to send the message once a receive window is opened by the end device.

The integration also hosts a ‘Resource Directory’ to facilitate discovery of the available CoAP reverse proxies (i.e. LoRaWAN end devices) and their resources. The integration also offers an HTTP/CoAP cross protocol proxy to enable HTTP access to the CoAP resources.

A.3 Binary data encoding over LoRaWAN

One issue with the approach presented above is the limited usability of upstream and downstream messages. Typically, these messages combine multiple data sources in a binary encoding in order to increase the efficiency of communication (i.e. send more information in shorter chunks of data). While highly efficient data formats with minimal data size are important in the LoRaWAN network, they are not so important on the CoAP reverse proxy where network resources with third parties are much less limited than in LoRaWAN networks. In fact, when sharing data it can be cumbersome for third parties to interpret the binary encoding used by LoRaWAN end devices.

There are multiple approaches to tackle this issue. One approach defines a mapping between a binary encoding and CoAP resources for every type of LoRaWAN application message. Depending on the type of binary encoding, this mapping may be described in a domain specific language or syntax or it may be implemented as part of the CoAP proxy for binary encodings that cannot be formalized. Examples notations that define binary encoded messages include ASN.1 and protocol buffers. These mappings are used to deconstruct binary messages into their data fields. When an upstream message arrives at the reverse proxy, it can be deconstructed into its individual fields (using the mapping) and the proxy may offer one CoAP resource for every field in the message. Note that a similar approach can be applied for downstream traffic, where multiple CoAP resources are combined as data fields of one binary message. In this case, it is also the mapping that defines which fields (i.e. resources) will make up the downstream message. As a result, external systems are freed from (de)combining multiple data fields from or

into one message.

In our approach we chose to use Google's protocol buffers for the binary mapping as it generates compact binary messages (important in the LoRaWAN network), while supporting verbose message structures. Additionally, there exists a lightweight protobuf implementation¹ with a small memory footprint that is suited for use on resource-constrained LoRaWAN end devices. The CoAP reverse proxy can use the protocolbuf library as made available by Google Inc.

A.3.1 Proof of concept demonstration

In order to demonstrate our approach, we have setup a small proof of concept with an IMST iM-880A-L LoRaWAN demo board, a gateway, network server and the CoAP++ framework. The nanopb software processes the message definition (the so-called proto file) and outputs two files that contain the struct definition and default initializer of the message. The embedded program uses these two files, as well as the nanopb runtime library to serialize a message struct to its binary message and deserialize a binary message in its message struct.

An example is presented below for the APIPort3 message, which is a LoRaWAN message type offered by the LoRaWANMac SDK. On the left the protobuf message definition of the APIPort3 message is shown, while the struct definition produced by nanopb for the APIPort3 message definition is shown on the right. In this case the APIPort3 message contains information on the peripherals available on the demo board: is the LED turned on, what is state of the potentiometer and what is the voltage measured at the power source?

<pre>message APIPort3 { required uint32 AppLedStateOn = 1; required uint32 PotiPercentage = 2; required uint32 VDD = 3; }</pre>	<pre>/* Struct definitions */ typedef struct _APIPort3 { uint32_t AppLedStateOn; uint32_t PotiPercentage; uint32_t VDD; } APIPort3;</pre>
---	---

The listing below shows the protobuf encoding of the integer values of the _APIPort3 fields on the left. They are encoded as so-called variable integers (var ints), where shorter bytes sequences are assigned to small integers. On the right, the serialized APIPort3 message is displayed for the value of the data fields on the left. The underlined bytes are the headers added by protobuf to encode the type, the field number and length (if using variable length types) of the field in the message. In this case three additional bytes are added to transport the meta data of the message.

<pre>0x00 0x64 0xF5 0x2D</pre>	<pre>0x08 0x00 <u>0x10</u> 0x64 <u>0x18</u> 0xF5 0x2D</pre>
---------------------------------	---

On the CoAP reverse proxy a protocol buffers module was added, based on the protobuf library. The protobuf module accepts a proto message definition and

¹nanopb: <https://github.com/nanopb>

a URI to the upstream resource hosted on the reverse proxy. The module uses protobuf reflection (using the message definition) to parse protobuf messages at run time into their fields. For every known field, the protobuf module will create a corresponding CoAP resource on the reverse proxy of the LoRaWAN end device (at the moment the name of the resource is set to the field name). When an upstream message is received from the LoRaWAN backend at the reverse proxy, it is deserialized into its fields and these fields are used to update the resources on the reverse proxy. The result for the APIport3 example is shown below:

```
coap://lwdevice1.test/AppLedStateOn      -> AppLedStateOn: 0
coap://lwdevice1.test/PotiPercentage     -> PotiPercentage: 100
coap://lwdevice1.test/VDD                -> VDD: 5877
```

A.4 Conclusion

This appendix illustrated how device virtualization can be applied to share data coming from LoRaWAN end devices and to send data towards end devices. The CoAP reverse proxy hides the details of the API offered by the LoRaWAN backend from third-party developers, thereby aiding in integration of LoRaWAN devices. Compared to the efforts at the IETF of bringing IPv6 into the LoRaWAN, the proposed method can be considered as an alternative for when the compression mechanisms cannot be implemented (e.g. updating end devices in existing deployments is unfeasible) or for when the overhead after compression of IPv6 remains too large. For external systems there is no difference between the end-to-end IPv6 and reverse proxy cases, as they see a CoAP server in both cases.

While the protocol buffers encoding is not the most efficient in terms of size (e.g. the AppLedStateOn is a flag that is encoded into 2 bytes in the message), it does strike a good balance between compactness and usability. The protobuf language allows formalizing many different types of messages. Additionally, implementations of the protobuf encoding are available on a wide range of platforms. The proof of concept proved that – due to a compact implementation – protocol buffers are also suitable for the type of embedded devices that are used in LoRaWAN networks (the iM-880A-L contains an ultra-low-power STM32L151, with 128KB flash memory and 16KiB RAM memory).

