BILHANAN SILVERAJAN

Communication Protocol and Management Considerations for Internet of Things Gateways

Tampere University Dissertations 426

Tampere University Dissertations 426

BILHANAN SILVERAJAN

Communication Protocol and Management Considerations for Internet of Things Gateways

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Technology and Communication Sciences
of Tampere University,
for public discussion in the Auditorium TB109
of Tietotalo, Korkeakoulunkatu 1, Tampere,
on 29 June 2021, at 12 o'clock.

ACADEMIC DISSERTATION
Tampere University, Faculty of Information Technology and Communication Sciences
Finland

ISBN 978-952-03-1980-9 (print)
ISBN 978-952-03-1981-6 (pdf)
ISSN 2489-9860 (print)
ISSN 2490-0028 (pdf)
http://urn.fi/URN:ISBN:978-952-03-1981-6

PunaMusta Oy – Yliopistopaino
Joensuu 2021

Abstract

Today, it is inconceivable to think about modern life without the Internet. The Internet has transformed our lives, serving as a digital backbonefor communication, media and Web-based interaction among human users. It has, however, begun to evolve into the Internet of Things (IoT), interconnecting constrained devices such as sensors, actuators and miniaturised computing platforms. Aided in large part by advances in decreasing hardware costs, better battery, radio and computational technology, these kinds of connected IoT devices have rapidly begun to proliferate. The resulting traffic generated and exchanged by IoT devices is expected to dwarf that produced by humans.

However IoT devices exhibit significant differences in terms of computational and storage capacities, wireless radio networks and communication protocols. Integrating them into the Internet is not straightforward, in terms of device addressing, reachability as well as data representation and transfer. In many IoT domains, there are already deployed networks, devices and sensors based on legacy technology, that the IoT is compelled to integrate. All this has led to the emergence of IoT gateways in playing a central role for integration and interoperability, in ensuring interconnection and end-to-end communication with different kinds of IoT devices and networks.

IoT gateways themselves can be multi-functional; their complexity and usage depends on exactly what kinds of IoT devices and networks they need to integrate into the Internet, and what additional services are expected of them in ensuring successful interaction with edge devices. This dissertation focuses on these roles IoT gateways fulfill, enabling them to function as essential components of the IoT.

The main research question of this dissertation is: *"For various kinds of IoT network topologies, how can gateways be configured and managed, to support connectivity and communication with IoT end devices?"* To answer this question, four research areas for IoT gateways were studied, by applying Design Science Research Methodology (DSRM): Network Connectivity, Energy Consumption, Protocol Composition, and Gateway Management.

For network connectivity, the communication and reachability requirements exerted on IoT gateways by IoT devices and various IoT edge network topologies were studied. For energy consumption, the utilisation of gateway energy consumption patterns were investigated, in order to optimise communication as well as monitor operational performance. For protocol composition, the specification, implementation and deployment of protocols, gateway protocol stacks and network services was researched. Lastly, for gateway management, the research into redundancy management as well as development of IoT management practices for operational management and configuration, was performed.

The results and contributions of this dissertation are categorised into three abstraction levels of an IoT gateway. These are at the network, management and communication levels of abstractions respectively. At the network abstraction, the work is oriented towards IPv6-based networks. The results describe the various IoT topologies, and gateway configuration and design to aid with IPv6 address allocation and network connectivity for edge devices. At the management abstraction, the dissertation identifies patterns in IoT gateway management and contributes object models and communication techniques for gateway configuration, operational monitoring, redundancy management and device proxying. At the communication abstraction, the dissertation contributes an implementation framework and a specification language allowing rapid development of communication protocols. The results also describe how energy consumption in IoT gateways can be employed both to optimise data transmissions as well as detect network-based attacks.

Both proofs of concept as well as conducted field experiments were used to verify the results of the dissertation. Empirical findings and obtained results have also been adopted both for Internet and IoT device management standardisation.

Preface

"My big fish must be somewhere."
-Santiago, Ernest Hemingway's *The Old Man and the Sea*

My doctoral dissertation was in gestation for a significant amount of time. This was my big fish. I have had the distinct privilege of meeting, interacting and working with some immensely talented people throughout this process.

I wish to firstly acknowledge Tampere University for providing me the opportunities to pursue my doctoral research, and the conducive environments in which to undertake it. Prof Jarmo Harju was responsible for setting me on this journey and served as my mentor and supervisor until his retirement. Prof Kari Systä then took an active role towards my dissertation as my supervisor. I am grateful to them both for their guidance, reviews and frank discussions which became instrumental in shaping and completing my dissertation.

I wish to thank Prof Jari Porras and Prof Winston Seah, for serving as pre-examiners. Their very thorough reviews helped significantly improve my final manuscript. I also wish to thank Prof Sasu Tarkoma for graciously agreeing to be my dissertation opponent.

My gratitude goes to Ms Elina Orava from the Faculty of Information Technology and Communication Sciences and Ms Sonja Mattsson from the Library Open Science Services for their advice and assistance in the publication of my dissertation.

I have selected nine peer-reviewed publications for inclusion into my dissertation, and my deepest thanks go to my co-authors in all these papers: Karri Huhtanen, Riku Itäpuro, Nadir Javed, Jaime Jiménez, Antti Kolehmainen, Juha-Pekka Luoma, Andrea Mola, Maurizio Magarini, Mert Ocak, Teemu Savolainen, Davide Scazzoli, Jonne Soininen, Markku Vajaranta and Giacomo Verticale.

I have been, and continue to be, greatly enriched by my involvement with the IETF community, particularly the members of the CoRE Working Group, as well as with members of the OMA, and Smart Object Committee of the IPSO Alliance (which is now part of OMA SpecWorks).

No words can convey the depth of gratitude I feel towards my family members for their trust, support and encouragement. Your sacrifices meant I never gave up.

Finally I wish to acknowledge the many persons without whom my dissertation would not have come to fruition. I cannot name each and every single one of you, for you are too many. Thank you all for giving me your shoulders to lean on when I needed your aid, and to stand upon when I needed to reach higher.

Contents

Acronyms

3GPP	Third Generation Partnership Project
6LowPAN	IPv6 over Low-Power Wireless Personal Area Network
6lbr	6LowPAN Border Router
ABNF	Augmented Backus-Naur Form
APIs	Application Programming Interfaces
BLE	Bluetooth Low Energy
Bluetooth SIG	Bluetooth Special Interest Group
CBOR	Concise Binary Object Representation
CoAP	Constrained Application Protocol
CoMI	CoAP Management Interface
CoRE	Constrained RESTful Environments
DAD	Duplicate Address Detection
DALI	Digital Addressable Lighting Interface
DHCPv6	Dynamic Host Configuration Protocol version 6
DRX	Discontinuous Reception
DSRM	Design Science Research Methodology
DTLS	Datagram Transport Layer Security
EDGE	Enhanced Data rates for Global Evolution
GATT	Generic Attribute Profile
GUA	Global Unicast Address
HSPA	High Speed Packet Access
HSRP	Hot Standby Routing Protocol
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force

IID	Interface Identifier
IoT	Internet of Things
IP	Internet Protocol
IPC	Interprocess Communication
ISP	Internet Service Provider
JSON	JavaScript Object Notation
L2CAP	Logical Link Control and Adaptation Protocol
LAN	Local Area Network
LoRaWAN	Long Range Wide Area Networks
LTE	Long Term Evolution
LWIG	LightWeight Implementation Guidance
LWM2M	Lightweight Machine-to-Machine
M2M	Machine-to-Machine
MQTT	Message Queuing Telemetry Transport
MSC	Message Sequence Chart
NATs	Network Address Translations
ND	Neighbour Discovery
NB-IoT	Narrowband IoT
OCF	Open Connectivity Foundation
OECD	Organisation for Economic Co-operation and Development
OMA	Open Mobile Alliance
PI	Provider Independent
REST	Representational State Transfer
RTT	Round Trip Time
SDK	Software Development Kit
SDL	Specification and Description Language
SLP	Service Location Protocol
SLAAC	Stateless Address AutoConfiguration
SDOs	Standards Development Organisations
SDN	Software Defined Networking

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UE	User Equipment
ULA	Unique Local Addresses
UML	Unified Modelling Language
URI	Universal Resource Identifier
URN	Universal Resource Name
VRRP	Virtual Router Redundancy Protocol
WAN	Wide Area Network
WoT	Web of Things
XML	Extensible Markup Language

List of Figures

List of Tables

List of Publications

1. Savolainen T, Soininen J, Silverajan B. "IPv6 addressing strategies for IoT." *IEEE Sensors Journal.* vol 13, no 10, pp. 3511-3519, 2013, IEEE. DOI: 10.1109/JSEN.2013.225969

2. Silverajan B, Huhtanen K, Harju J. "IPv6 experiments in deploying and accessing services from home networks." *Proceedings of the 2006 IEEE Asia-Pacific Conference on Communications (APCC '06)*, pp. 1-5, 2006, IEEE. DOI: 10.1109/APCC.2006.255794

3. Silverajan B, Harju J. "Developing network software and communications protocols towards the Internet of Things". *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE (COMSWARE'09)*, pp. 9:1-9:8, 2009, ACM. DOI: 10.1145/1621890.1621902

4. Silverajan B, Harju J. "Factoring IPv6 device mobility and ad-hoc interactions into the Service Location Protocol." *Proceedings of the 32nd IEEE Conference on Local Computer Networks (LCN 2007)*, pp. 387-394, 2007, IEEE. DOI: 10.1109/LCN.2007.108

5. Silverajan B, Luoma JP, Vajaranta M, Itäpuro R. "Collaborative cloud-based management of home networks." *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 786-789, 2015, IEEE. DOI: 10.1109/INM.2015.7140376

6. Silverajan B, Ocak M, Jiménez J, Kolehmainen A. "Enhancing Lightweight M2M Operations for Managing IoT Gateways." *Proceedings of the 2016 IEEE International Conference on Internet of Things (iThings 2016)*, pp. 187-192, 2016, IEEE. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.55

7. Scazzoli D, Mola A, Silverajan B, Magarini M, Verticale G. "A Redundant Gateway Prototype for Wireless Avionic Sensor Networks." *Proceedings of the 29th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE PIMRC 2018)*, pp. 1-7, IEEE. DOI: 10.1109/PIMRC.2017.8292683

8. Savolainen T, Javed N, Silverajan B. "Measuring energy consumption for RESTful interactions in 3GPP IoT nodes." *Proceedings of the 7th IFIP/IEEE Wireless and Mobile Networking Conference (WMNC 2014)*, pp. 1-8, IEEE. DOI: 10.1109/WMNC.2014.6878863

9. Silverajan B, Vajaranta M, Kolehmainen A. "Home Network Security: Modelling Power Consumption to Detect and Prevent Attacks on Homenet Routers." *Proceedings of the 11th Asia Joint Conference on Information Security (AsiaJCIS 2016)*, pp. 9.16, IEEE. DOI: 10.1109/AsiaJCIS.2016.10

Author's Contribution to the Publications

Publication 1 analyses the suitability of different IPv6 addressing strategies for nodes, gateways, and various access network deployment scenarios in the IoT. The publication is aimed at highlighting the heterogeneity of nodes and network technologies, extreme constraint and miniaturization, renumbering, and multi-homing, present serious challenges toward IPv6 address allocation. The author was responsible in contributing to the main idea of the paper, in identifying some of the challenges, the main motivations and challenges, describing topologies for IoT deployments as well as a thorough literature survey.

Publication 2 focuses on how IoT gateways can be successfully employed to provide connectivity as well as tunnelling services, to local area networks for end-to-end reachability over the Internet. The publication provides the usage scenario of IPv6 home networks and home gateways to ensure connectivity to edge devices and data sharing from home nodes. Although the publication discussed IPv6 transitional technologies, many of the challenges regarding overlays, tunnels and advanced configuration for gateways continue to exist today to interconnect IPv6 edge networks via the global IPv4 Internet. Ease of deployments and configuration, as well as minimising any disruption to existing network services were studied. The author was the primary contributor to the paper.

Publication 3 explains the design and implementation of a network programming framework which facilitates the design and implementation of communication protocols, as well as communication protocol stacks. The author was the primary architect of all the core components of this framework as well as some of the optional application layer protocol components. The framework is especially suitable for the needs of IoT gateways, as it is lightweight, portable across many POSIX-compliant platforms and offers a clean-layered design that allows for rapid prototyping and implementation of protocols.

Publication 4 describes how advanced services and network applications can be discovered upon portable and mobile nodes changing their IPv6 network points of attachments. The approach allows movement awareness, even if mobility is shielded at the network layer, such as with Mobile IPv6. The author designed and implemented 2 kinds of agents for the Service Location Protocol (SLP), which can then be deployed into IPv6 as well as mobile IPv6 devices and gateways for performing dynamic service discovery. The SLP agents were implemented using the same framework described in Publication 3.

Publication 5 examines the issues of remote IoT network and gateway management, especially in evolving smart homes consisting of multi-vendor IoT devices which may require network segmentation and routing. The publication considers how gateway management can be jointly performed by network operators and third party experts in collaboration with the home network owners. The author was the primary architect of this REST-based architecture which separates the management and control functionalities of the gateway away from the actual connectivity, traffic forwarding and routing operations within the home network.

Publication 6 highlights how management of IoT gateways differ significantly from traditional network management practices, and the challenges that exist for remote gateway management, particularly for proxies and exposing proprietary or vendor-specific operations. The author is the primary contributor of this publication which proposes solutions based on the Lightweight Machine-To-Machine (LWM2M) protocol, gateway data models and prototypes.

Publication 7 describes how lessons learnt in the design of IoT gateways can be applied towards mission-critical systems in which the data from heterogeneous wireless avionics sensors must be initially aggregated by gateways and subsequently be transmitted to the cloud with high reliability and no loss. As gateways proved to be possible points of failure, the author proposed a redundant gateway management system in which redundant gateways are employed, and contributed to the idea of developing a heartbeat protocol to perform failure detection of an active gateway and aid rapid fail-over using a secondary gateway.

Publication 8 delves into power consumption aspects for IoT gateways and devices in cellular networks. Detailed measurements were undertaken in various 3GPP networks, to compare and contrast the energy profiles of various REST-based protocols that can conveying sensor data. The motivation for this study is to aid in optimising operation times, particularly for devices and gateways which are energy-constrained. The author contributed to the general idea of the paper, and was also responsible in the analysis and interpretation of the obtained data. Moreover, the author proposed a method of conveying CoAP messages using the WebSocket protocol, which was shown to be energy efficient, and was subsequently adopted as an Internet standard.

Publication 9 expands on the role of power consumption in mesh and Wi-Fi gateways from a security perspective. The author proposed an approach of using anomalous power consumption as a means to detect various kinds of attacks in progress in IoT gateways. By studying the energy patterns of the routers in various scenarios particularly with regards to the routing infrastructure, network owners can detect attacks in progress with a high level of confidence. The author was also responsible for the literature survey as well as presenting the key findings from the measurements taken.

1 Introduction

The Internet of Things (IoT) is characterised by interconnected, communicating smart objects and devices, such as sensors, actuators, electronic tags and miniaturised computing platforms. There are more than 10 billion IoT devices today in many diverse domains, ranging from industrial automation, connected lighting, intelligent transportation, logistics handling, digital health, to smart homes, buildings and cities. These IoT devices produce data upon which actions can be autonomously undertaken by other IoT devices and the general Internet. Other networked systems, applications and human users also interact with these devices to consume such data, particularly over the Web. In fact, the IoT is envisioned to evolve towards an architecture where IoT devices and communication become an integral, interconnected part of the Web. The realisation of this vision, called the Web of Things (WoT), calls for applying principles of the Web architecture, especially using the Representational State Transfer (REST) paradigm, so that real-world objects and embedded devices can blend seamlessly into the Web infrastructure and tools [3].

Hardware diversity, owing to differences in chip-sets, radio technology, energy efficiency and manufacturing costs has led to IoT devices exhibiting significant differences in computational, storage, communication and power constraints. In order to categorise differences in IoT devices in terms of these properties, the LightWeight Implementation Guidance (LWIG) Working Group within the Internet Engineering Task Force (IETF) devised a simple three device classification system in RFC 7228 [4] based on device processing capabilities as well as the amount of volatile and storage memory available. This was provided after performing an exhaustive study of commercially available chips and design cores for constrained devices [1]. The classification provided in RFC 7228 is as follows:

- Class 0 devices, which have (much) less than 10 KiB of RAM for data and 100 KiB of flash memory for program code. These smart objects can be considered to be severely constrained and would require the assistance of more powerful devices to connect them to the Internet.
- Class 1 devices, which have approximately 10 KiB of RAM and 100 KiB of flash memory. These constrained smart objects are capable enough to use a protocol stack specifically designed for constrained nodes, and can connect to the IoT without needing any intermediary device. However, they cannot easily communicate with other Internet nodes employing a full protocol stack.
- Class 2 devices, which have about 50 KiB of RAM and 250 KiB of flash memory. These smart objects are capable of supporting most of the same protocol stacks

[1]At the time of this writing, RFC 7228 is being updated in LWIG to reflect newer architectures. See https://tools.ietf.org/html/draft-bormann-lwig-7228bis-03

as used on notebooks or servers. However, even these devices can benefit from lightweight and energy-efficient protocols and from consuming less bandwidth.

RFC 7228 focused on constrained, low-end IoT devices in terms of their hardware. Taivalsaari and Mikkonen [5], on the other hand, proposed an alternative, software-based classification identifying seven types of IoT devices based on their software architecture. This classification not only included low-end devices such as sensors, but more powerful nodes that could run full-fledged desktop operating systems. The resulting taxonomy can then be used to understand ease of programmability and application deployment constraints for any kind of device connected to the IoT.

1.1 Gateways in the IoT

Regardless of the types of IoT devices in existence today, the IoT is composed of a multitude of such devices communicating and exchanging data with other connected devices, applications as well as human users over the Internet. To overcome the heterogeneity and constraints in hardware, software and communication protocols that various IoT devices use, intermediate nodes called gateways are often employed for interconnecting end devices in many IoT domains.

The concept of using gateways is certainly not new in computer communications. However, they remain vital to the success of the IoT, and consequently, gateway usage continues to grow in the IoT. Market projections forecast that, by 2021, the IoT gateway market will result in a billion dollar industry [6], with shipments exceeding 64 million units [7]. IoT gateways can be classified according to their capabilities, similar to IoT device endpoints. They can be limited in computation power, storage capacity and power consumption. For example, they can be battery-powered, and be designed to operate with energy efficiency in mind. In Machine-to-Machine (M2M) environments they can operate for great lengths of time without supervision. They can also be computationally powerful, and contain support for operating systems, embedded development and executing application logic. Some of the tasks that IoT gateways are used for are:

- *Multi-protocol bridging and translation.* IoT devices can use a wide variety of wireless or wired technologies. In its simplest role, a gateway acts as a protocol bridge, to interconnect an IoT device using one type of data link to another network segment using a different data link. The most common example of such a gateway today is a wireless access point or router offering 802.11-based Wi-Fi to clients, and connecting to the Internet over a wired or cellular uplink. The same principle also holds for other wireless technologies such as Long Range Wide Area Networks (LoRaWAN) or Narrowband IoT (NB-IoT). Additionally, many deployed IoT devices and networks exist today that are not natively designed to communicate using the Internet Protocol (IP). Some examples here are Bluetooth, Bluetooth Low Energy (BLE) and Zigbee-based devices. For these, an IoT gateway can serve as a communication proxy by translating application, network and data link protocols. These allow non-IP devices to be integrated into the IoT.
- *Message and payload adaptation.* Protocol translation is perhaps one of the simplest aspects of a gateway's functionality. It is also often the case that whenever there is a communication disparity among communicating IoT endpoints, payload processing needs to be performed for seamless data exchange. IoT gateways facilitate syntactic

and semantic interoperability by manipulating the serialisation, format and structure of payload contents. Payload adaptation can be performed even when no protocol translation is needed. For example, data values transmitted by an IoT device conforming to one data model, can be adapted to a different data model understood by a receiving endpoint, even if both IoT devices use the same communication protocol.

- *Data Processing.* IoT gateways are increasingly becoming important for edge computing, where, instead of processing all sensor data in centralised cloud systems, intelligence and computing is transferred in IoT edge networks. IoT gateways interact with edge devices as well as upstream services to aggregate, pre-process or filter data. This is done in order to lower latency, reduce the number of transferred bytes, or improve transmission reliability.

- *Security and Resilience* Computationally limited, as well as energy constrained IoT devices, have a restricted availability for strong authentication and encryption mechanisms for secure communication with an endpoint. In some industrial domains, the legacy protocols and networks utilised can also contain little to no security, such as lighting systems using a legacy technology standard called the Digital Addressable Lighting Interface (DALI). For these usage scenarios, computationally powerful IoT gateways can be introduced not only to improve the security and resilience of an IoT network, but also provide security auditing, logging and intrusion detection systems capable of detecting malicious activity.

These roles and properties of an IoT gateway, have a direct impact on the communication protocols, security, energy management policies, interfaces, interaction models and embedded software that need to be developed. Also, gateway management needs to be undertaken throughout the lifecycle of an IoT gateway, similar to IoT device management. In other words, the kinds of role and properties an IoT gateway has, influences how it can be commissioned and made operational into a network, how any application-level logic can be securely deployed and how its firmware and any software can be securely updated. Thus, while IoT gateways can constantly evolve in terms of hardware features, power requirements and radio communication interfaces, some of the core techniques for embedded protocol development, methods and patterns for gateway management, as well as communication mechanisms can be abstracted, improved and designed based on sound engineering principles.

This dissertation, comprising the introduction and its publications, describes these considerations and the work undertaken in developing IoT gateways to facilitate data transfer and endpoint communication in the IoT. The rest of this chapter discusses the research questions, scope and contributions, undertaken methodology for the research as well as the structure of the dissertation.

1.2 Research Questions

In this dissertation, there are four perspectives from which the research questions are derived:

1. *Connectivity and reachability.* Low powered constrained devices today require a border gateway to become reachable, while non-IP devices require gateways

capable of understanding and translating between different network technologies. Very constrained devices, such as 1-wire sensors, require gateway assistance to interconnect to the IoT. Performing bridging, translation and proxying at multiple levels of a communication stack was discussed earlier as one of the most common roles for an IoT gateway. In addition to that, IoT gateways provide other services such as naming or addressing or tunnelling between an edge network and a core network.

2. *Energy awareness in gateway communication.* Energy consumption is an important trait in IoT networks. Not only does it have implications on the constraints and operation of edge devices such as sensors and actuators, it also impacts the behaviour of IoT gateways, particularly for gateways which may themselves be battery powered. Battery drainage can occur rapidly, not only when a suboptimal combination of application layer protocol and radio technology is used, but also when the gateway is subject to misbehaving firmware or malicious attacks.

3. *Designing, implementing and deploying protocols.* The design and implementation of communications subsystems as well as protocol stacks are often used to provide advanced application level services. Implementing such services and protocols should be performed in a consistent modular manner. When rapid development, testing and prototyping are required, protocol implementation frameworks can supply many advantages as compared to creating individual custom protocols from scratch. This is particularly important when considering fast evolving protocols for which new features need to be incorporated constantly.

4. *Gateway management and configuration.* IoT device management involves the processes of installing, bootstrapping, operating and maintaining IoT devices in the network. An IoT gateway is an important component of this process, in order to allow a management server to reach the IoT devices. While some processes for device management are applicable to IoT gateways themselves, IoT management services have yet to consider the architectural impact of introducing gateways, gateway configuration and gateway management explicitly, into the overall management of an IoT network.

From these, the following research questions are addressed:

Research Question 1: What do connectivity characteristics, communication support and data reachability for current and future IoT network topologies require from IoT gateways? (Publications 1, 2 and 8)

Research Question 2: How can energy measurements be used to improve and monitor gateway operation? (Publications 8 and 9)

Research Question 3: How can the development of protocols and communication subsystems in IoT gateways be performed rapidly and consistently? (Publications 3 and 4)

Research Question 4: What are the crucial aspects of IoT gateway management that need to be considered, and how can gateway management be performed and aligned with IoT device management? (Publications 5, 6, and 7)

1.3 Scope and Contributions

The scope and contributions of this dissertation are illustrated in Figure 1.1. As the figure depicts, the scope of the dissertation lies in specifying suitable abstractions for IoT gateways as well as in practical development work in realising and validating the gateway abstractions.

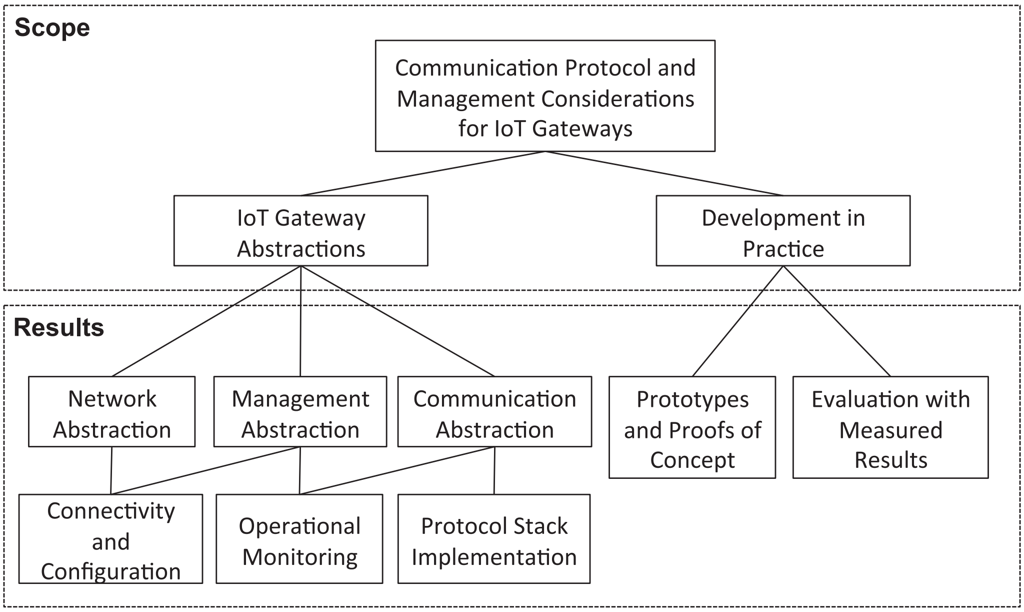

Scope
Communication Protocol and Management Considerations for IoT Gateways
IoT Gateway Abstractions
Development in Practice
Results
Network Abstraction
Management Abstraction
Communication Abstraction
Prototypes and Proofs of Concept
Evaluation with Measured Results
Connectivity and Configuration
Operational Monitoring
Protocol Stack Implementation

Figure 1.1: Dissertation Scope and the Contribution

From the IoT gateway abstractions, the results are categorised into three abstraction levels: management abstraction, network abstraction and the communication abstraction. These abstractions are augmented with prototypes, proofs of concept and measurement data. Together, the obtained results form the contributions of this dissertation.

The research into the network abstraction focuses on a generalised IoT architecture comprising of IoT nodes in an edge Local Area Network (LAN), an IoT gateway connecting the LAN to a geographic or organisation-specific Wide Area Network (WAN) and subsequently the Internet. The work strongly focuses on IPv6 for IoT, based on the sheer number of nodes which aim to be connected to the Internet, and the difficulties seen with IPv4 address allocation and flexibility. The contributions for the network abstraction are listed as follows:

- Categorisations of IoT network topologies based on usage scenarios and required connectivity for edge IoT nodes.
- Gateway configuration and IPv6 address allocation strategies to nodes in the LAN.
- Empirical research and measurement data demonstrating the feasibility of gateway-enabled IPv6 transition and tunnelling technologies, in the absence of native IPv6 support in backhaul networks.

The research into the management abstraction concentrates on new methods for performing IoT gateway management in a unified manner with existing open industrial standards for IoT device lifecycle management. The contributions for the management abstraction are listed as follows:

- Identification of specific gaps in IoT device management standards to allow lifecycle management of IoT gateways.
- Categorisations of IoT gateway management practices based on observed live deployments.
- Extending the Lightweight Machine-to-Machine (LWM2M) protocol, a well-known IoT device management standard, for gateway management by designing new data models and interaction methods for configuration, monitoring and operational management of edge IoT gateways. The data models designed were subsequently adopted as standard application-specific objects by the IPSO Alliance.
- Design of a redundant gateway management system using a heartbeat protocol.

The research into communication abstraction targets two subareas, which are in software-based protocol development of the gateway's communication subsystem, and in the operational monitoring of energy consumption of gateway communication. Research into protocol development for the gateway communication subsystem aims at producing tools and libraries for consistent design, implementation and deployment of application-level communication protocols. Energy awareness, especially for power-constrained gateways, aims at prolonging the lifetime of a gateway, both in terms of energy efficient communication, as well as detecting security anomalies. The contributions for the communication abstraction are listed as follows:

- A C++-based protocol implementation framework called DOORS, providing a uniform development environment encompassing C++ classes useful for protocol development, including abstractions for hardware characteristics, and operating system communication interfaces.
- A specification language describing protocol logic and messages, written in the Extensible Markup Language (XML) that facilitates code synthesis that could be compiled with code written using the DOORS framework libraries to obtain protocol implementations.
- Energy measurement results for REST-based gateway communication. Measurement data for the energy consumption of different REST protocols was obtained over live cellular networks from three different Finnish operators and subsequent activity resulting from this work became an Internet standard [8].
- Novel techniques to use IoT gateway energy measurements to detect and fingerprint different categories of network-based attacks.

1.4 Research Methodology

The Frascati Manual [9], prepared by the Organisation for Economic Co-operation and Development (OECD), provides fundamental definitions for research and development, which are internationally accepted in terms of what are included and excluded under various terminologies [10]. According to the Frascati Manual, research and development covers three activities[9]:

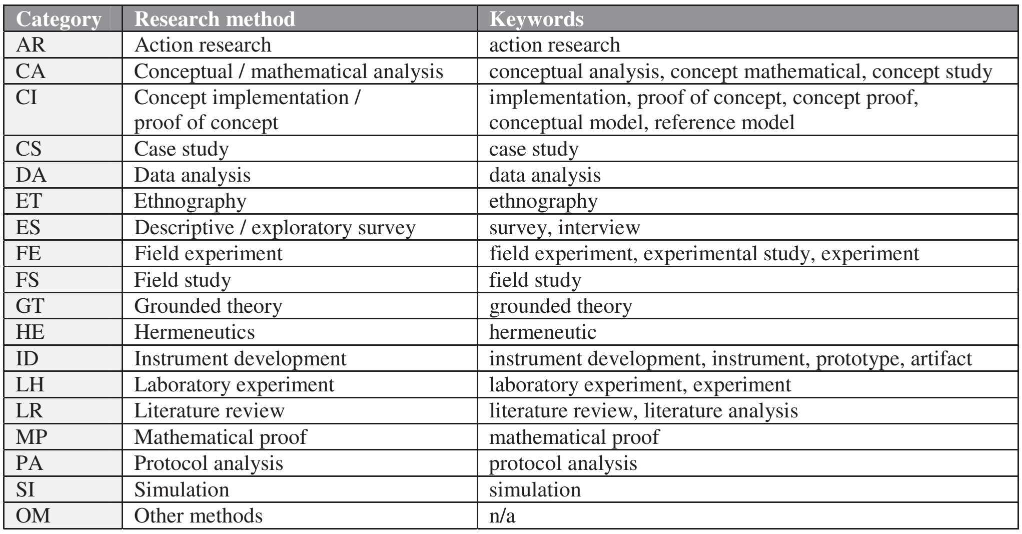
1. *Basic research*, which is experimental or theoretical work undertaken primarily to acquire new knowledge of the underlying foundation of phenomena and observable facts, without any particular application or use in view.
2. *Applied research* which is also original investigation undertaken in order to acquire new knowledge. It is, however, directed primarily towards a specific practical aim or objective.
3. *Experimental development*, which is systematic work, drawing on existing knowledge gained from research and/or practical experience, which is directed to producing new materials, products or devices, to installing new processes, systems and services, or to improving substantially those already produced or installed.

Regardless of the type of research involved, the research process involves a series of actions and steps that need to be considered [11]:

1. Formulate the research problem
2. Review existing work and literature
3. Develop a working hypothesis
4. Prepare the research design
5. Obtain the data
6. Analyse the data
7. Present findings

An analysis of papers by Ebeling et. al in five Information Science conferences from 2006 to 2010, revealed that up to 17 categories of research methods were used [1]. Each of these 5 conferences were chosen to represent Information Science research in a specific part of the world. Figure 1.2 describes these categories.

The resulting study revealed that the most frequently mentioned methods throughout all five conferences were "descriptive/exploratory survey" (24.7%) and "concept implementation/proof of concept" (21.8%). The conclusion was that although there were many methods available to researchers in the broad IS field, only a few dominating research methods were used. The differences in research preferences around the world were also highlighted: European computer science research has focused on qualitative or exploratory research methods, in which the researchers predominantly used non-empirical methods such as "conceptual" and "system development."



Category	Research method	Keywords
AR	Action research	action research
CA	Conceptual / mathematical analysis	conceptual analysis, concept mathematical, concept study
CI	Concept implementation / proof of concept	implementation, proof of concept, concept proof, conceptual model, reference model
CS	Case study	case study
DA	Data analysis	data analysis
ET	Ethnography	ethnography
ES	Descriptive / exploratory survey	survey, interview
FE	Field experiment	field experiment, experimental study, experiment
FS	Field study	field study
GT	Grounded theory	grounded theory
HE	Hermeneutics	hermeneutic
ID	Instrument development	instrument development, instrument, prototype, artifact
LH	Laboratory experiment	laboratory experiment, experiment
LR	Literature review	literature review, literature analysis
MP	Mathematical proof	mathematical proof
PA	Protocol analysis	protocol analysis
SI	Simulation	simulation
OM	Other methods	n/a

Figure 1.2: Categories of research methods (classification scheme), from [1]

Table 1.1: Research methods employed in publications

Publications	P1	P2	P3	P4	P5	P6	P7	P8	P9
Conceptual Analysis (CA)	x				x				
Concept Implementation (CI)		x	x	x	x	x	x	x	x
Data Analysis (DA)								x	x
Field Experiments (FE)								x	
Instrument Development (ID)			x			x	x		x
Laboratory Experiments (LH)				x			x		x
Protocol Analysis (PA)	x			x				x	x

The research methods used in this dissertation largely are in line with these findings. Using this categorisation, the publications included in this thesis generally fall into the category of research methods displayed in Table 1.1.

While research methods may be understood as methods and techniques that are used for the conducting of research, research methodology on the other hand, is a way to systematically solve the research problem [11]. The research methodology is perceived as the various steps that are generally adopted by a researcher in studying his research problem along with the logic behind them. It is therefore necessary to know not only the research methods, but also the methodology, as research methods constitute a part of the research methodology.

To put these concepts into perspective, the focus of this dissertation is on applied research as well as experimental development. Basically, the work undertaken for this dissertation falls into a category of applied research methodology known as Design Science Research Methodology (DSRM) [12].

DSRM is the predominant methodology used for Information Science research. It can be envisioned as a process model consisting of six activities in a nominal sequence, namely:

- Problem identification and motivation

- Definition of the objectives for a solution
- Design and development of a research artefact
- Demonstration
- Evaluation
- Communication

Problem identification and motivation. In this activity, the specific research problem is clearly defined, and the value of a solution is justified. All publications, P1 - P9 include an identification of the problem and motivate the research direction clearly towards a solution.

Definition of the objectives for a solution. The objectives of the solution developed should be clearly outlined based on the problem identification. All publications, P1 - P9 provide an objective of the research.

Design and development of a research artefact. Here, the theoretical knowledge obtained from the previous two activities is applied to the creation of an artefact. This includes both its functionality and its architecture as a means to overcome the identified problems. All publications, P1 - P9, include such a development of an artefact.

Demonstration. When the artefact is designed, it is experimented with, used in simulations, or otherwise used as a solution against the perceived problems initially identified. Publications P1, P2 and P4 - P9 provide a demonstration activity. P3, being an analysis paper identifying key problems and then offering solutions based on expert solutions, does not have a demonstration phase.

Evaluation. In this activity, an analysis is performed as to how well the research artefact meets the objectives highlighted once the problem has been identified. Improvements can be made to the artefact, if deemed necessary by returning back to the design activity. The result of evaluation can range from a comparison of the developed solutions against other existing solutions, to justifying how the developed artefact meets the research objectives. All publications, P1 - P9, provide a thorough evaluation of the solution.

Communication. The final activity is the process of dissemination of the obtained results and its effectiveness against the problems identified, to relevant forums and publishing outlets. In this regard, all publications P1 - P9 fulfil the criteria for this activity.

1.5 Thesis Structure

In addition to this introductory chapter and the inclusion of nine publications, this dissertation adds six chapters.

Background and work related to this dissertation is reviewed in Chapter 2. Chapter 3 covers the key connectivity, communication and reachability functionalities needed in IoT gateways. Chapter 4 summarises the main findings of how power consumption can be used as a metric to determine both communication methods, as well as anomalous gateway behaviour. Chapter 5 is devoted towards the work done for the development of a protocol implementation framework that could be used for the development of advanced protocols and protocol stacks for IoT gateways. Chapter 6 is focused towards findings related to IoT gateway management. Chapter 7 presents concluding remarks.

2 Background and Related Work

The aim of this chapter is to present existing work related to this dissertation. The chapter is structured to be aligned with the research questions from Section 1.2, as well as the subsequent chapters in this dissertation. Where necessary, the background of specific technologies or architectural principles are also briefly discussed, to frame the existing and related work in context.

2.1 Connectivity and Reachability

It is already well understood that the need for interconnecting smart objects into the Internet will exert tremendous strain on the existing addressing infrastructure of the Internet running on IPv4. Currently, both the top level, as well regional Internet registries have now fully exhausted the IPv4 address allocation space. The next-generation IPv6 protocol was standardised to overcome many limitations seen with IPv4, but challenges still exist in migrating the Internet fully towards IPv6 as observed by the IETF Sunset4 Working Group [13]. However, IPv6 adoption has begun to accelerate in the last few years. Live statistics on IPv6 adoption for Internet-based communication have shown exponential uptake in the past half decade, with current penetration rates today in some countries as high as 35% [14]. For the IoT, Ziegler et. al stress that IPv6 is referred to by a growing number of IoT and M2M related standards, and that recent research favours using IPv6 even on the most constrained devices. [15].

In this dissertation, IPv6-based networking and communication is seen as a fundamental technology in several of the included publications, for enabling IP connectivity and reachability for IoT devices and gateways. Simpler address allocation mechanisms and the availability of global addresses were important motivations for using IPv6 in Publications 1, 2, 4 and 9. However, other factors were also considered. These included end-to-end reachability in Publication 2 and easier network mobility in Publication 4.

This section discusses the related work around connectivity and reachability in the context of Research Question 1 in 1.2 and Publications 1, 2 and 6. Subsection 2.1.1 introduces focuses on the technologies on connecting edge IoT nodes into the Internet, which relate to the research done for the dissertation, as well as have great market relevance today. Subsection 2.1.2 then looks at existing literature and research work performed in the academia.

2.1.1 Connecting edge IoT nodes

A variety of gateway connectivity solutions are used to connect IoT devices in edge networks to the Internet. Many of these solutions require the gateway to perform functions such as network address allocation, protocol conversion, tunnelling or packet adaptation. For end-to-end IPv6 communication, in addition to the kinds of IPv6 devices present in the edge network, upstream IPv6 connectivity from a network or service provider also determines what functionality gateways play, and how IPv6 connectivity is supplied to end devices.

Two kinds of IPv6 edge networks are discussed in the following subsections. The first is the Homenet architecture for IPv6 home networks, which is a networking enabler for consumer-oriented IoT devices. Various aspects of Homenet are researched and discussed in Publications 5 and 9. The second is about IPv6 over Low-Power Wireless Personal Area Network (6LowPAN), a means of enabling constrained nodes to use IPv6 in an optimised manner. 6LowPAN is not directly discussed in the Publications, but is nevertheless an important technology for connecting IoT devices.

2.1.1.1 Homenet

Homenet was developed and standardised by the IETF Home Networking Working Group. With the increasing amount of connectivity needed for intelligent consumer electronic devices as well as the anticipated number IoT devices, sensors and actuators in the home, it was recognised that residential home networks and gateways need to evolve beyond simple Network Address Translations (NATs) and repeaters [16].

Home networks are envisioned to become complex enough to require multiple network segments and subnets within the home. This would therefore require the presence of interior gateway routing protocols, and it can therefore be assumed that Homenet supports the existence of multiple gateways that act as routers: A border router supplying overall connectivity services to the entire home and several interior routers, which, together with the border router, need to be orchestrated to perform actual routing in the home network.

Home owners may not be technically adept. Consequently, one driving goal for the design of the Homenet is to allow automatic configuration of the network in terms of IPv6 connectivity as well as discovery of services. The network must survive temporary uplink disruptions and services within the Homenet must remain reachable at all times by other nodes in the home network.

In order to meet these expectations and requirements, the Homenet architecture advocates the following steps for obtaining IPv6 adddresses into the home network:

- The Internet Service Provider (ISP) has to firstly support native IPv6 addressing. More specifically, it has to support the existence of residential IPv6 subnets, by supplying IPv6 address blocks to a requesting Homenet border router. The border router accomplishes this using an extension to the Dynamic Host Configuration Protocol version 6 (DHCPv6), known as DHCPv6 Prefix Delegation [17]. DHCPv6 Prefix Delegation allows an IoT gateway to request a DHCPv6 server for an IPv6 prefix. The obtained IPv6 prefix represents an IPv6 address block allocated to the gateway by the ISP's DHCPv6 server. Gateways within the home can similarly obtain prefixes from the border router with DHCPv6 Prefix Delegation.
- IPv6 devices within the home obtain their addresses statefully from their network points of attachment using DHCPv6, or statelessly using Stateless Address AutoConfiguration (SLAAC).

Although any internal routing protocol can be deployed in a Homenet, the Working Group has adopted Babel [18] as the mandatory to implement routing protocol. Babel comprises part of the core Homenet protocol suite [19]. Babel is a distance-vector protocol and provides fast convergence. This allows the mesh to automatically self-repair the routes should gateway failure be detected. When Babel is used, instead of a hierarchically organised routed network, gateways automatically configure themselves as an ad-hoc Wi-Fi mesh network. Many of the connectivity and routing components for Homenet, including Babel, are easily available for installation, particularly onto Linux-based embedded platforms for home routers as well as microcomputers such as the Raspberry Pi. As a consequence, Chroboczek [20] already describes some existing deployments for Babel, from hybrid networks to small unmanaged networks. Because Babel itself is a very simple protocol and can be deployed in constrained networks many IoT deployments using Homenet technology can be envisioned.

2.1.1.2 6LowPAN

6LowPAN provides a way to allow end-to-end IPv6 communication with constrained nodes with limited processing and power. 6LowPAN assumes the use of IEEE 802.15.4 radio technology: The network environment is assumed to be lossy, with a low bandwidth supporting small packet sizes [21].

The gateway for a 6LowPAN network is called the 6LowPAN Border Router (6lbr). Packets back and forth between an IPv6 backbone network and a 6LowPAN network undergo adaptation by the 6lbr. Regular IPv6 addresses undergo header compression, as well as fragmentation and reassembly [22] to adapt the sizes of IPv6 datagrams into optimised packets suitable for transmission into the 6LowPAN network.

6LowPAN has been adopted for use in the standards of various consortia such as the ZigBee Secure Energy Profile 2.0, the OpenThread Group and the Open Connectivity Foundation (OCF) [23]. In addition, the Bluetooth Special Interest Group (Bluetooth SIG) as well as the IETF have standardised the use of IPv6 over BLE [24], based on a new 6LowPAN adaptation layer.

2.1.2 Integrating Non-IP Edge Nodes

Gateways play an integral role as an enabling the interconnection of non-IP edge nodes into the IoT. IPv6 address allocations schemes for the IoT can sometimes be domain-specific, and existing research work in this area has employed an optimal combination of standardised and proprietary schemes. For example, Chakraborty and Chaki [25] propose an IPv6 address allocation scheme for Smart Grids based on hierarchical clustering, in which the addressing assignment is customised for smart grids. For Wireless Sensor Networks (WSNs), Shin et al. [26] focus on reducing address collisions during Duplicate Address Detection (DAD) using a virtual coordinate system based on locations of nodes in a 6LowPAN. Cheng at al. [27] similarly propose another location-based address allocation scheme for WSNs in a smart grid, called DSIPA, which aims at energy efficient address configuration.

Similar to address allocation schemes, there is a lack of standardisation in mapping IPv6 (or even IPv4) protocol semantics as well as datagrams to non-IP protocols. Depending on the technology involved, protocol conversion in gateways for integrating non-IP edge nodes to the IoT can either be very trivial or a very complex operation. Innovative approaches for multiprotocol gateways exist in academic research. While protocol conversion was not performed as part of this dissertation, related academic research for multiprotocol gateways is discussed here for completeness, in enabling communication between IPv6 and non-IP networks.

Zou et. al. [28] proposed three approaches for the design of a home gateway to interconnect Zigbee-based energy measurement sensors to IPv6 networks. The first is to have IPv6 over Zigbee, the second is adopt a dual-stack approach with both IPv6 and Zigbee running side by side while the third option is to consider the conversion of 6LowPAN and Zigbee addresses into standard IPv6 addresses. While the authors do not provide an exhaustive comparison of these approaches, the design of their home gateway uses the third approach. Zachariah et. al [29] proposed a general-purpose IoT gateway on modern smartphones as a software service. This provides Internet connectivity to the cloud over a cellular network, to BLE-based IoT devices. They envision that any BLE device could leverage any smartphone as a temporary IP router and act as a normal IP endpoint. Two approaches are proposed: The first approach uses the phone purely as a 6LowPAN border router to interconnect the BLE peripheral to the cloud, if it is capable of running IPv6 over BLE. The second approach considers protocol translation, in which the BLE profile data is translated and carried as a payload in an HTTP POST operation to the cloud. This is similar to an earlier work from Rouhana and Horlait [30] who proposed an architecture called the Bluetooth Web Internet Gateway, in which a gateway performed protocol translation between Transmission Control Protocol (TCP) and the Logical Link Control and Adaptation Protocol (L2CAP) layer of classic Bluetooth to allow web-based communication by a Bluetooth-enabled endpoint. For integrating legacy technologies and devices, Jara et al. [31] presents mapping techniques between legacy technologies from home automation, industrial and logistic areas to IPv6 by the use of an IPv6 addressing proxy. For wired networks, Meduna [32] looked at how to interconnect wired lighting networks based on the DALI standard to IPv6 using a protocol translation gateway running 6LowPAN.

2.2 REST and IoT

REST-based communication is an important facet of this dissertation and is a basis of Publications 5, 6 and 8. For this reason, a short background is provided about using REST in IoT in this section.

Today, interaction on the Web, as well as Web Application Programming Interfaces (APIs) are predominantly based on the REST paradigm [33], using version 1.1 of the Hypertext Transfer Protocol (HTTP). HTTP implements methods (such as GET, PUT, POST and DELETE) to manipulate the representation of resource states. A Universal Resource Identifier (URI) identifies the location of each resource served by an endpoint, as well as the communication protocol necessary to allow clients to interact with servers.

When REST-based communication is performed with constrained IoT devices such as sensors or actuators, each request or response can involve the transfer of just a few bytes expressing the state of the resource. For such operations, using HTTP can be sub-optimal, since it has been designed as a text-based protocol using a connection-oriented transport. Consequently, the IETF Constrained RESTful Environments (CoRE) Working Group designed a lightweight, binary, UDP-based REST protocol useful for IoT, called the Constrained Application Protocol (CoAP). CoAP is similar in semantics to HTTP, but optimised for constrained, IP-capable nodes [34]. Resources are described using a URI containing a ”coap://” scheme name or alternatively a ”coaps://” scheme name if security is required.

CoAP nodes exchange request and response messages using GET, PUT, POST and DELETE method calls, and the protocol ensures that message overheads are extremely small to avoid fragmentation. CoAP response codes are designed by intent to correlate with their HTTP counterpart (2.01 Created in CoAP versus 201 in HTTP, and 4.04 Not Found in CoAP versus 404 in HTTP). These features are aimed at allowing CoAP-based end points to easily integrate into the Web and communicate with HTTP-based endpoints without much difficulty. Interworking between CoAP and HTTP are covered by two documents: section 10 of RFC 7252 [34] which provides guidance for CoAP-HTTP proxying, and RFC 8075 [35] which additional guidelines with regards to how URIs, response codes, payload types are mapped between CoAP and HTTP endpoints.

2.3 Energy Measurements

Since IoT devices as well as gateways can have limited power, energy efficiency must be treated as a 1st order constraint. In this section, the related work around IoT gateway energy considerations are discussed in the context of Research Question 2 in 1.2 as well as Publication 8 with regards to balancing energy efficiency with communication needs and Publication 9 where energy consumption can be used for the detection of anomalous network and gateway activity.

For home gateways, Kaup et al. [36] suggest the usage of the Raspberry Pi as a home gateway that can be flexibly adapted to many needs to interconnect devices. They present a power consumption model called PowerPi that could be used to derive possible power savings based on CPU usage and network utilisation. The power profile of the Raspberry Pi was also studied for its suitability as a remote gateway for wireless sensor nodes by Astudillo et al. [37]. Martinez et al. developed a general methodology and framework for modelling the energy consumption of IoT devices that can be employed to forecast the impact of various application parameters on the power consumption levels [38].

In terms of general purpose measurement platforms, several research activities exist. The popular Energino hardware and software platform was developed by Gomez et al. [39] as an Arduino based low cost solution for energy monitoring in wireless networks. EMPIOT is another energy measurement platform for wireless IoT devices designed and implemented by Dezfouli et al [40].

There are a broad range of existing methodologies for taking energy measurements of different kinds of devices. Merlo et al. [41] described how, in measuring power consumption of Android-based mobile devices, they developed and implemented an app that extracts energy consumption data both from dedicated APIs of the operating system as well as low level kernel drivers to probe the battery information. Pathak et al. [42] also present their work on designing and implementing software for the energy profiling running applications in smartphones.

Power limited gateways make attractive targets for malicious activity that can compromise the gateway or cause failures. Consequently, anomalous power consumption and misbehaviour should be identified early in order not to affect the proper functioning of the network. In studying anomaly detection in wireless sensor networks, Rajasegarar et al. [43] summarised nine possible classes of attacks that could be mounted: Jamming, Tampering, Sinkhole, Denial-of-Service, Spoofing, Selective forwarding, Sybil, Wormholes and Eavesdropping. Anomaly detection in wireless sensor networks was also the focus of the research by both Jurdak et al. [44] and Kanev et al. [45] although neither considered the usage of power consumption as a metric. Oliner et al. [46] however, applied the usage of a collaborative approach of detecting energy anomalies in mobile devices by first aggregating energy data from multiple mobile devices and then performing a diagnosis on individual devices.

2.4 Composing Protocols and Protocol stacks

Development of communication protocols and application-level networking services in the IoT communication protocols and networking services is challenging, given the heterogeneity of networks and devices. Additionally, this remains an evolving area, with emerging standards describing their own protocols, as well as new functionality being incorporated to existing protocols. This has led to various development and implementation efforts for communication protocols and protocol stacks in IoT gateways, client and devices. When the required protocols or protocol stacks are not present as part of the software or firmware of an IoT gateway or device, developers have two alternatives. The first alternative is to retrieve and install any available third-party source code or architecture-specific binary packages, compiling them if necessary. Should custom protocols or more complex protocol stacks need to be developed, the second alternative is to design and implement the required communication protocols or stack. It is for the second alternative that using a protocol implementation framework, is a better proposition.

The main emphasis in this section is on how existing protocol implementation tools and frameworks can be employed to engineer communication protocols for IoT gateways. The work discussed in this section pertains to Research Question 3 in 1.2 and Publications 3 and 4 and has been further structured into two subsections. Subsection 2.4.1 looks at related work done for using domain-specific languages in creating new communication protocols. Subsection 2.4.2 describes the approaches different kinds of protocol implementation frameworks take, in providing building blocks for protocol, or protocol stack implementations.

2.4.1 Using Domain-Specific Languages

To accelerate the development of communication protocols from their initial inception based on requirements to their eventual deployment, some protocol development tools and frameworks rely on describing the the protocol in a high-level domain specific language. The commercially available IBM Rational SDL suite[47] is an instance of a framework uses domain-specific languages for protocol design. The SDL suite provides a graphical development environment that allow a developer to specify the entire system beginning with a high-level specification using the Unified Modelling Language (UML). The developer can also visualise and conceptualise protocols and network components with the Specification and Description Language (SDL) [48], a formal language that can be used for the analysis and verification of communication protocols. SDL diagrams specifies protocol designs using systems, blocks and processes. Systems encapsulate blocks which similarly are composed of processes. SDL diagrams can be augmented with usage and interaction scenarios with the Message Sequence Chart (MSC) editors provided. Such a high-level approach to develop protocols allows easy iterations between design and formal verification. Testing and simulation can also be undertaken with the framework together with components provided, to eliminate any design errors. Finally, from the validated and tested protocol design, the SDL suite enables the generation of C or C++ code and executables for some target architectures.

Bromberg et al [49] describe research work with a domain-specific language called z2z that was developed to automatically generate network protocol translators. In z2z, the developer specifies three kinds of modules: a protocol specification module, a message specification module, and a message translation module. A compiler is then used to generate running C code which is then linked to a runtime system. Burgy et al designed a domain-specific language called Zebu which is derived from the Augmented Backus-Naur Form (ABNF), to describe text-based and line-oriented IP-based application layer protocol handling [50]. From the given specifications, the Zebu compiler generates protocol-handling layers that can be tailored to the needs of the network application using the protocol. Mercadal et al [51] adopted a similar approach to develop the Zebra language, with the emphasis being to generate hardware parsers for embedded systems instead of software-based parsers.

Liu et al. designed a protocol specification language called GALANG, which forms part of their research in designing a protocol translation tool to facilitate IPv4 to IPv6 transitioning, called Generic Application Layer Translator (GALT) [52].

2.4.2 Using Protocol Implementation Frameworks

Instead of using domain specific languages, other kinds of protocol frameworks focus on the direct support for code implementations. The development methodologies employed by these frameworks, which take a protocol implementation from its conceptualisation until its final conception, can differ. In the following subsections, some dominant methodologies used by protocol implementation frameworks, as well as examples, are described.

2.4.2.1 Micro-protocol frameworks

Frameworks such as CORDS [53], Ensemble [54] and Cactus [55], provide a repository of micro-protocols, that are used to construct a protocol implementation. Each micro-protocol implements a single protocol-specific property as a building block. These properties represent, among others, features such as congestion control, flow control, error correction, sequencing and caching. Subsequently, a developer, by carefully analysing or designing a protocol based on its set of properties, uses these blocks to implement the resulting protocol. Bai et. al [56] describes the process of de-composition of routing protocols in ad-hoc networks, which were implemented using micro-protocols. A related paper from the research team reveals the usage and evaluation of ad-hoc and wireless sensor network routing protocols implemented with micro-protocol frameworks [57].

2.4.2.2 Whitebox frameworks

Whitebox frameworks provide language-specific components and objects which can be used by a protocol developer to obtain a protocol implementation, by deriving them framework-supplied objects and classes. The Adaptive Communication Environment (ACE) [58] is an example of a C++ white box framework used as a network programming toolkit. ACE can be used resource constrained environments. It also supports high performance, multiprocessing and concurrency for mission critical and real-time systems. It shields the developer from operating system and network-specific details using wrapper classes and platform adaptation layers. ACE employs software design patterns extensively in three ways. Firstly, they describe an abstraction of the construction of protocols and network services. Secondly, they explain inter-object interaction. Finally, they also are used to document the way ACE itself works.

2.4.2.3 Protocol Repositories

Protocol repositories often contain a library of ready-made protocols that the developer is immediately able to use. The biggest advantage of protocol repositories lie in the ability for rapid prototyping with minimal effort, using previously implemented protocols the framework supplies. The biggest drawback is that in order to gain any advantage of rapid prototyping over the other kinds of protocol frameworks, this approach requires an active community or support team being able to implement a wide range of protocols that a developer can subsequently use. The portable, Python-based Twisted [59] implementation framework is such a protocol repository which has an extensive collection of protocols. Twisted is event-driven, and objects called Deferreds implement callback functions. Twisted has been used in both ad-hoc as well as wireless sensor networks by the Simple Sensor Syndication project [60] in addition to the Lycaon [61] evaluation framework.

2.5 Management of IoT nodes

This section visits existing network and device management standards, practices and research work, that are important in the context of Research Question 4 in 1.2 as well as Publications 5, 6 and 7. As Publication 6 as well as chapter 5 extensively reference the REST-based Lightweight Machine-to-Machine standard, important architectural aspects of the standard are discussed. Then, an overview of other related REST-based standards pertinent to IoT device management are given. Finally, related research is presented.

2.5.1 The Lightweight Machine-to-Machine standard

Lightweight Machine-to-Machine (LWM2M) is a management protocol standard from the Open Mobile Alliance (OMA) [62]. LWM2M is perhaps the most dominant REST-based IoT device management protocol today. LWM2M was introduced by the OMA as a means for managing an entire range of IP-enabled constrained and non-constrained devices, both for device management and application data. LWM2M supports a client-server architecture, where managed endpoints are called clients. The LWM2M management server can be a physical host, or, for managing large-scale deployments, it typically resides as a cloud-based service. LWM2M leverages open standards, particularly from the IETF.

LWM2M uses a RESTful architectural style, and natively relies on CoAP for messaging. The LWM2M data model allows endpoints to expose their capabilities as standardised Objects. Each LWM2M Object is uniquely identified with a Universal Resource Name (URN) and an Object Identifier. It also contains a list of resources that are directly mapped into CoAP-based URIs and resources. Objects and resources are identified in a compact manner by a 16-bit integer. An endpoint can consist of different types of Objects and can register any number of Object types and instances to a LWM2M Server. OMA also maintains a public LWM2M Object registry with which third-party Standards Development Organisations (SDOs), private organisations as well as individual users, can register their own application-specific Objects. As an example, the IPSO Alliance maintains its own set of application-specific LWM2M Objects. LWM2M operations on managed endpoints are grouped into 4 logical device management interfaces:

- The bootstrapping interface allows either a client or a server initiated process for the provisioning of keying material, configurations and access control lists, with which the client can securely communicate with the server. For pre-provisioning, a bootstrap server can be used as a trust anchor.

- The registration interface allows a client to inform the server about the kinds of objects the endpoint is hosting.
- The management and service enablement interface is used by the server to manipulate or retrieve object and resource representations on a managed client. In LWM2M, resources can be read, written to, or executed. These basically translate into the GET, PUT and POST CoAP operations.
- The information reporting interface allows long-lived subscriptions from the Server to a resources of interest in a client. Then, whenever the state of a resource changes, the client notifies the new resource value to the server.

2.5.2 Other IoT device management SDOs

In the IETF, the CoAP Management Interface (CoMI) [63] aims at enabling the use of data models specified in the YANG modelling language [64] in constrained environments, together with the use of CoAP. The use of YANG was first introduced in NETCONF and then subsequently used in RESTCONF. Object identifiers in YANG are specified as hexadecimal strings. NETCONF and RESTCONF also encode YANG payloads in JavaScript Object Notation (JSON) or XML. Motivated by byte savings, CoMI uses compresses YANG string identifiers to numeric identifiers using base64 encoding. Also, CoMI performs YANG payload serialisation in the Concise Binary Object Representation (CBOR) format, which is a highly compact encoding format suitable for use in constrained networks.

The Open Connectivity Foundation (OCF) is an industry driven consortium to allow application interconnectivity to IoT devices, as well as machine-to-machine communication. The OCF specifications [65] describe an extensive RESTful architecture which provides language-specific API mappings to a core framework which has functionalities ranging from addressing, endpoint and resource discovery, device management, security and messaging. The primary REST protocol that OCF uses is CoAP, although bindings for certain operations such as firmware updates, can be performed using HTTP.

oneM2M is another industrial consortium established through an alliance of standards organisations to develop a single horizontal platform the for exchange and sharing of data among all applications [66]. The architecture supports a layered model which is composed of three layers: Application Layer, Common Services Layer and a Network Services Layer. Among others, the Common Services Layer provides functions such as Discovery, Registration, Communication Management, Device Management, Security and Location. For multiple device management, oneM2M supports several different protocol standards of which LWM2M is one [67].

2.5.3 Existing Work

REST-based IoT device management, and the standardisation activity around it, focus on Internet-enabled devices. As Publication 6 describes, today, device management standards do not specify how the management of non-IP devices, as well as devices with proprietary data models, are performed. Publication 6 describes how LWM2M-based device management can be extended towards the usage and configuration of gateways in supporting the management of non-IP devices. Here, similar approaches are described.

Kim et. al [68] described the design of a multiprotocol IoT home gateway that provides device management of IP and non-IP devices in the home using Message Queuing Telemetry Transport (MQTT). The design of the home gateway supports third parties to monitor the devices in the home, receive aggregated device data for reusing the data and to send messages for device control. This is accomplished by allowing the gateway to expose a REST interface using HTTP GET and POST methods. Huang et al. [69] advocated the use of a framework employing Software Defined Networking (SDN) to dynamically configure networks with IoT devices for scalability. Tayur and Suchitra [70] applied the same principles of SDN for device management in smart cities from the cloud, that also supports multi-tenancy. Also, within the domain of home networks, Perumal et al. [71] developed an IoT device management framework for smart homes to manage wireless IP and non-IP devices. The prototype can be deployed in smart phones and consumer-grade computing equipment, and enables device configuration, device discovery as well as data collection. Similarly, Jin and Kim [72] conceived an OCF-based management architecture to manage various kinds of devices by adding a proxy layer mapping OCF data models to the proprietary models of these devices. Chang et. al [73] proposed and designed a gateway through which a LWM2M server can manage legacy, non LWM2M devices. Additionally, the work demonstrates how interworking between two management frameworks can be achieved using this gateway, by incorporating the gateway to also work with the oneM2M platform. Datta and Bonnet [74] also implemented a oneM2M-based device management framework that uses LWM2M.

2.6 Summary

This chapter presented research work and the state of the art related to the work done in this dissertation. This was presented and structured according to the four research questions, in terms of connectivity and reachability, energy measurements, protocol composition and gateway management. At the same time, a background of the standards, technologies and communication paradigms important in this dissertation, was given.

It can be seen that particularly for allowing reachability between IP nodes and non-IP nodes, a variety of domain-specific address allocation, address mapping and translation technologies can be utilised. The work in this dissertation builds on these approaches, particularly in how IoT gateways can facilitate IPv6-based connectivity into edge networks, as well as communication and data transfer between IP nodes and BLE-based IoT devices. This is further employed for gateway and device management in this dissertation using REST-based communications and standard data models in LWM2M to interact with gateways to retrieve data from non-IP IoT devices.

For composing communication protocols, a large variation also exists for the tools, frameworks and methods for , and there are no directly comparable qualitative metrics amongst them. This variety is also due to specific application scenarios to which these tools and frameworks are targeted for protocol composition. In this dissertation, a hybrid approach was taken to combine the use of a domain-specific language to describe protocols, partial code generation and a white box framework to abstract message-based inter-object interaction and network services. In doing so, this presented a flexible approach for addressing the runtime computing, storage and hardware constraints in IoT gateways, as well as facilitating portability and run-time event monitoring.

The proposed approaches described in this chapter for undertaking energy measurements were performed as a means to optimise power and reduce energy consumption in end devices as well as gateways. Software-based approaches, such as smartphone apps, work well in end devices and gateways to provide fine-grained profiling, but usually require more powerful operating systems that may not always be found in IoT gateways in which firmware run in embedded systems. In addition, when considering the use of power consumption as a metric for security anomaly detection, they can either be inaccurate or ineffective. The approach undertaken in this dissertation, therefore, adopts hardware solutions and hardware-based data acquisition to address the hardware heterogeneity common in IoT gateways, in order to undertake energy measurements to perform anomaly detection as well as balancing energy efficiency with communication needs.

3 Connectivity and Reachability Considerations

This chapter presents the dissertation work performed towards IP connectivity and networking services provided by the gateway, in allocating addresses, delivering reachability and communication for IoT devices in edge networks. Primarily this focuses on the work done in Publications 1 and 2. However, work from other publications are also drawn upon. Publications 5 and 9 employ Homenet IPv6 networking, Publication 8 is used to illustrate how data reachability can be achieved with REST-based proxying, particularly the use of alternative transports, and Publication 4 for advanced service discovery.

Because of the depletion of publicly available IPv4 addresses, the research in the publications strongly emphasise IPv6-based networking and communication, as a fundamental technology that IoT gateways need to support for interconnecting IoT devices and networks. Therefore, the network connectivity and device reachability considerations for IoT gateways in this chapter focus on the usage of IPv6 for future IoT network toplogies and connected edge devices.

3.1 Gateways allocating native IPv6 addresses in edge networks

Publication 1 describes eight different kinds of edge network topologies that can impact IPv6 addressability and reachability for edge devices when native IPv6 addresses are available. From these eight topologies, six topologies are dependent on the existence of one of more IoT gateways for proper address allocation for IoT devices. In the following subsections, some specific challenges of an edge IoT network are examined, and the solutions are presented for how an IoT gateway can allocate native IPv6 addresses into the edge network.

3.1.1 Gateway solutions using DHCPv6 Prefix Delegation

IoT gateways must be able to cope and adapt to ISP prefix allocation policies and deliver announcements and possible prefix allocations reliably to other gateways and devices in the edge network. Without any pre-existing address assignment agreements between an ISP and customer organisations, DHCPv6 Prefix Delegation is the standard approach for extending networks. However, the prefix length supplied for customer LANs can vary substantially from ISP to ISP. For example, ISPs can provide a /48 prefix, which allows up to 65536 subnets per LAN, while supplying a /60 prefix restricts the LAN to just 16 subnets. In Publication 1, the gateway solution is to accomplish this by announcing routes as well as longer prefixes into the edge network. This is also demonstrated in

Publications 5 and 9. The Homenet gateway in Publication 5, for example, received a /60 prefix from the ISP, while in Publication 9, a /56 prefix was supplied instead. [1]

An IoT gateway obtaining IPv6 address blocks using DHCPv6 Prefix Delegation must also flexibly support different strategies to deliver IPv6 addresses into the edge network. Publication 1 describes how a combination of stateful and stateless address allocation strategiess can be successfully employed at the edge network. Stateless IPv6 addresses are obtained with SLAAC, while DHCPv6 can be enabled for stateful IPv6 address allocations. It is also important to note that DHCPv6 Prefix Delegation can also exist within an edge network, that can enable a downstream IoT gateway to request IPv6 address blocks from the border gateway.

In cellular networks based on the Third Generation Partnership Project (3GPP) standards, DHCPv6 Prefix Delegation has been specified for 4G Long Term Evolution (LTE) Release-10 onwards [76] . However, DHCPv6 Prefix Delegation generally remains unsupported and undeployed. Upstream bridging by the User Equipment (UE), such as a gateway, is therefore used, by announcing the same /64 prefix that was received on the radio link, to the LAN link [77]. While this allows edge devices to obtain global IPv6 addresses via the gateway directly from the ISP it currently inhibits the automatic setup and routing of IPv6 edge networks via Homenet. However the Homenet setup described in Publications 5 and 9 provide a partial solution by allowing routable IPv6 using ULAs in the edge network.

3.1.2 Solutions for minimising connectivity disruption

In the IoT, address stability is important, given the expected heavy M2M traffic and significantly large numbers of nodes in network segments. Connectivity disruption as well as resumption can adversely affect addressing in the LAN, particularly if renumbering occurs and new IPv6 prefixes are allocated. Frequent connectivity disruptions and subsequent IPv6 address configuration operations also adversely affect power consumption for energy constrained devices.

Internet connectivity disruption and resumption can be a result of several factors, such as unexpected ISP service outages, poor signal quality on cellular connections, scheduled service breaks, or nodes (and even networks) physically moving and changing their points of network attachment. Such mobility can also potentially impact running services and applications, and Publication 4 suggests possible solutions.

Homenet-based gateways for residential networks, described in Publications 5 and 9, counter address renumbering by supplying (and multihoming) all nodes in the Homenet with two kinds of IPv6 addresses: A Global Unicast Address (GUA) and a Unique Local Addresses (ULA). GUAs and ULAs are roughly akin to public and private IPv4 addresses. GUAs can change upon loss and regain of Internet connectivity. ULAs do not guarantee duplicate addresses and address collisions beyond the LAN. However, nodes and services within the homenet remain reachable to each other via their ULAs, which persist during Internet connectivity outages.

Provider Independent (PI) IPv6 addresses are another alternative, in which the organisation is directly assigned an IPv6 address block by the Regional or Local Internet Registry.

[1]To ensure no IPv6 address depletion occurs in customer networks in Finland, Traficom, the Finnish Transport and Communications Agency, discourages supplying /60 prefixes. Instead it recommends ISPs to supply a /56 prefix to their customers (including residential users), allowing up to 256 subnets per LAN, for a total of 16 million IPv6 addresses [75].

An organisation having PI addresses does not succumb to renumbering. However, PI addresses are rarely issued, owing to the overheads and maintenance upkeeps necessary at the Internet Registry. With PI addresses, no special action is needed on the part of the gateway.

3.1.3 Solutions to support node heterogeneity

IoT networks can feature a diverse mix of constrained nodes, perhaps within even a single subnet. Addressing requirements for these nodes might differ. For example, extremely constrained nodes, such as Class 0 nodes, may not be able to perform Neighbour Discovery (ND) [78] and Duplicate Address Detection (DAD) [79] operations, and consequently require IPv6 address configuration assistance from less constrained neighbouring nodes or gateways. As mentioned in Section 3 of RFC 7228 [4], Class 0 devices will participate in Internet communications with the help of larger devices acting as proxies, gateways, or servers.

Integrating non-IP IoT devices may prove challenging if they do not possess a unique Interface Identifier (IID) that can be mapped easily into IPv6 addresses, such as the IEEE EUI-64. In such cases, an IPv6 address needs to be allocated to the IoT device by the gateway. This can be done either from a DHCPv6-based pool of unallocated addresses, or by the gateway generating a 64-bit IID on behalf of the device. A range of methods for arriving at different IIDs is discussed by Cooper et al. [80]. Apart from mapping IPv6 addresses to devices, the gateway should also participate in ND and DAD operations on behalf of the device, to eliminate address collisions.

3.2 Gateway address allocation via IPv6 transition technologies

When an ISP is unable to provide IPv6 addresses that IoT devices in an edge network can obtain, several kinds of translation, as well as tunnelling mechanisms are available, that allow IPv6 address allocations to IoT devices in edge networks as well as IPv6-based communication using IPv4 core networks.

Publication 2 describes the usage of 6to4 [81] technology in a home gateway to supply IPv6 connectivity into the home. 6to4 was conceived as one of several alternative IPv4 to IPv6 transition mechanisms which permit end-to-end IPv6 communication over IPv4 intermediate links. Other examples include Teredo [82] and NAT64 [83].

Transitional IPv4 to IPv6 technologies continue to be useful for IoT for two reasons:

- Much of the deployed Internet, although transitioning to IPv6, rely on IPv4 infrasructure and have enabled IPv6 transit on a limited basis.
- In geographic regions where fixed line Internet connectivity is limited and only a cellular WAN is present, transitional technologies aid in creating an IPv6 edge network.

As opposed to native IPv6, transitional IPv6 mechanisms employ a variety of techniques to transmit IPv6 packets over IPv4 links using either translation or tunnelling. An extensive study of IPv6-over-IPv4 tunnel mechanisms was performed by Steffan et al. [84]. A similar study was performed by Kim. Ruiz et al. [85].

Compared to native communication, transitional schemes induce packet processing and transmission overheads. Mizoguchi et al. studied quality degradation of various Web services between IPv4 and 6to4, with results showing that degradation is negligible for a user, unless the link characteristics exhibit packet loss as well as higher latencies [86]. In performance evaluation studies done by Bahaman et al. [87], packet throughput for UDP transmission over 6to4 compared favourably with native IPv4 and IPv6, athough for TCP, the authors reported a 50 per cent drop. Round Trip Time (RTT) measurements showed that 6to4 was on par with native IPv4 and IPv6.

These findings are in line with in a related study of Publication 2 performed by the author [88]. In the related study, experiments were conducted in live networks in Finland, based on a content sharing prototype developed for interconnected homes. In one study, measurements were taken between 2 endpoints geographically separated by 500 km. One hundred HTTP GET and POST operations were applied on small files of approximately 400 bytes. First the tests were conducted between IPv4 endpoints. Subsequently they were conducted between a 6to4 endpoint and a native IPv6 endpoint.

The results, depicted in Figure 3.1, indicate minor variations between using IPv4 or IPv6. Although IPv4 communication did, on average, perform better, the RTT observed with using 6to4 was only marginally greater. Additionally, it can be seen that both IPv4 as well as IPv6 communication coped with retransmissions and packet loss on the live network.

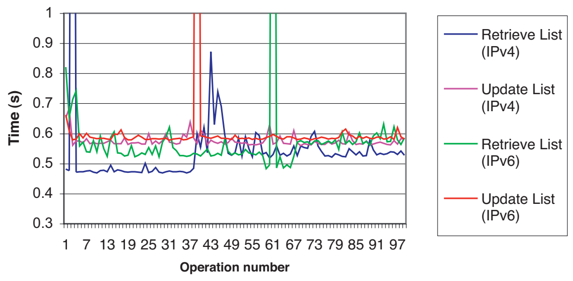


Figure 3.1: Roundtrip time comparison of IPv4, IPv6 and 6to4

One other observation for the slightly larger RTT observed with 6to4 rests with its routing mechanism. 6to4 uses anycast addresses to discover a 6to4 relay router to tunnel IP packets to, from the 6to4 border gateway. The 6to4 relay can reside in an arbitrary location on the Internet. The subsequent responses can follow a different path back to the 6to4 border gateway. Therefore, for UDP-based connectionless communication using 6to4, which can be prevalent in the IoT, there would be no measurable RTT difference with native protocols. TCP-based roundtimes would be affected, if acknowledgement packets take a different path back to the source endpoint, as was reported by Bahaman et al. [87].

While using 6to4 for highly reliable as well as high bandwidth services might be suboptimal, the same would not hold true for IoT-based communication characteristics. However, operational difficulties subsequently discovered with the use of anycasting in 6to4 [89] led to the development and deployment of its successor, 6rd [90]. Nevertheless, its usage remains popular as a transitional IPv6 technology, peaking at more than a third of the total IPv6 traffic in October 2014, as Figure 3.2 depicts. Even when the IETF formally deprecated the 6to4 anycast prefix and discouraged the inclusion of 6to4 in new host and router implementations in May 2015 [91], a small proportion of transitional IPv6 traffic continues to manifest itself.

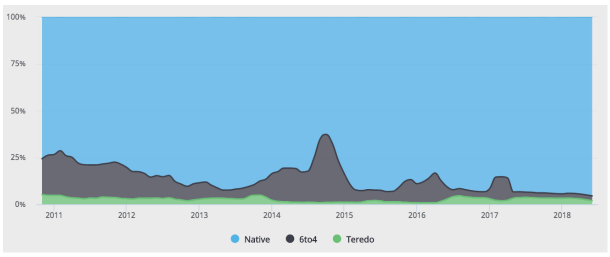


Figure 3.2: IPv6 Address Types, from [2]. Note that the figure does not show 6rd traffic as well as tunnelled IPv6 traffic, as these do not contain special prefixes.

3.3 Data Reachability

An additional dimension for reachability besides connectivity issues, stems from whether direct Internet connectivity between communicating endpoints is an essential requirement. As communication in the IoT is usually data driven, the use of a gateway for proxying and data relaying might be sufficient to fulfill the needs of communicating endpoints if end-to-end IPv6 connectivity is not required. In effect, this decouples the availability of an endpoint's data, from its network connectivity with a communicating peer.

The concept of decoupling the message payload from the communication protocol is presented in Publication 8. While the paper is oriented towards energy measurements, which is discussed in Chapter 4, it also describes how a gateway can preserve the semantics of REST-based communication while performing protocol translation to interact with resource representations. CoAP's original design was for using REST messages over the User Datagram Protocol (UDP) or Datagram Transport Layer Security (DTLS). As part of this dissertation, research was undertaken towards the design and implementation of the WebSocket protocol [RFC6455] as an alternative CoAP transport.[2] This was achieved by decoupling of its Request-Response logic, from the underlying messaging logic. Two kinds of underlying transports for CoAP are therefore discussed in Publication 8: CoAP using UDP as its transport, and CoAP using the WebSocket protocol [RFC6455] as its transport.

Websockets are an appealing transport for carrying CoAP messages for two particular reasons. Firstly, a CoAP client may be in an access network that allows communications only with HTTP or via HTTP proxies. In order to support the latter case, the constrained node hosting CoAP server would need to have a minimal TCP implementation with enough HTTP logic to negotiate a WebSocket session. While not as efficient as UDP, this approach would allow access to CoAP servers from less open networks. A client can communicate with a CoAP server by using end-to-end CoAP rather than traversing via a HTTP-CoAP proxy.

Secondly, web applications, or applications running in a web browser environment, do not have access to the underlying platform's UDP or TCP socket APIs, and the only means for communications besides HTTP is the use of WebSockets. Consequently, CoAP can be

[2]This research work was subsequently adopted into the IETF and culminated as RFC 8323[8]

implemented as a subprotocol within a WebSocket session, for example using client-side Javascript, and can be executed in a browser.

Additionally, BT SIG is in the process of standardising the use of a REST-based API that can be used by a client to a proxy to communicate with Bluetooth devices implementing the Generic Attribute Profile (GATT) [92] . Thus a client device can interact with a gateway either over HTTP, CoAP, or CoAP over WebSockets to interact with Bluetooth devices using this API.

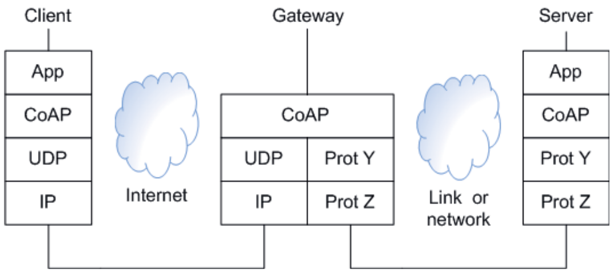


Figure 3.3: Abstraction of a simple CoAP-to-CoAP proxy over alternative transports.

The solution in Publication 8 also allows CoAP messages to be transported over other alternative transport protocols. Thus, simpler gateway proxies result, such as a CoAP-to-CoAP proxy, where only the transports differ. An abstraction of such a gateway is shown in Figure 3.3.

3.4 Summary

Depletion of publicly available IPv4 address pools has introduced significant new challenges for future public IP address allocation, particularly in the deployment, configuration and management of IoT devices. IPv6 has long been promoted to overcome the address exhaustion in IPv4 and provide significant advantages for connectivity and end-to-end reachability.

The research in this dissertation shows, however, that IPv6 deployment, specifically for IoT devices and networks continue to present significant challenges. Eight basic IPv6 network topologies for IoT deployments have been identified that IoT gateways need to consider in Publication 1. The feasibility of IoT gateways supporting mesh networking, automatic configuration and address allocation, particularly for edge networks in which different realms of control may exist, is a key factor in the successful deployment of some of the identified network topologies.

The results show that network operator support for DHCPv6 Prefix Delegation plays a key role in delivering IPv6 address blocks to edge networks via border routers and IoT gateways. Because of its scalability, the findings confirm that SLAAC is the most straightforward technology to allocate IPv6 addresses for IoT edge networks and devices. However, more complex deployment scenarios, can require additional measures to be taken by the gateway, such as DHCPv6, DHCPv6 Prefix Delegation to deliver address blocks to downstream gateways, or address mappings to support non-IP devices, in order to support end-to-end connectivity and reachability.

When network operator support for DHCPv6 Prefix Delegation is not present, such as in 3GPP networks, IoT gateways must perform bridging to deliver IPv6 connectivity. While this allows end-to-end reachability over IPv6 to IoT devices, it cannot be effectively utilised as an IPv6 address allocation strategy for the IoT network topologies described in this dissertation. Using IPv6 transition technologies continues to offer an alternative route to allow IPv6 deployment, connectivity and address allocation in such scenarios, or indeed when only IPv4 is supported in the backhaul network. As described in Section 3.2, performance measurements undertaken demonstrate the feasibility of IPv6 transition mechanism for IoT deployment.

The results of dissertation show that even when transport protocols or even networks differ in the IoT between endpoints wishing to communicate and exchange data, IoT gateways can employ other application level solutions using REST-based communication with CoAP over alternative transports. This is an effective strategy in which multi-protocol IoT gateways serve as REST proxies between disparate endpoints and still successfully preserve protocol semantics.

4 Energy awareness in gateway communication

This chapter describes the work performed in Publications 8 and 9. Publication 8 of this dissertation investigates how different protocols used for REST-based communication in the IoT, impact energy consumption in cellular networks. A common strategy used by constrained IoT sensors to conserve energy is to mostly reside in very low power, or sleep mode, sustain a very low duty cycle, and power up their radios only when necessary. This is not a realistic strategy for gateways, however. Gateways, for example, are equipped with multiple wireless interfaces and are more often required to be in active mode in order not to severely affect data readings and sensor traffic. In some deployments, they can additionally operate on battery power alone. As an example, cellular gateways exist which provide Internet connectivity to edge networks based various kinds of wireless and wired technologies. Optimising the usage of cellular radio therefore prolongs the duty lifecycle of these gateways.

Power depletion of a battery, or increased power consumption in gateways may not always be the result of sub optimal communication or increased communication from devices in the edge network to the Internet. Publication 9 studies energy consumption patterns in commodity home gateways induced with increased malicious activity. Specifically, it considers the Homenet architecture consisting of multiple gateways organised as a self-configuring mesh, and the impact of various attacks on the mesh infrastructure.

Taken together, these publications provide findings and insights as to how measuring energy consumption in the gateway can be beneficially used for both selecting the right kind of communication, and detecting communication anomalies quickly.

The rest of this chapter is as follows: Section 4.1 describes how energy consumption data was acquired. Section 4.2 discusses how different REST protocols and transports, in addition to different kinds of cellular network, significantly affect the power consumption at a gateway. Section 4.3 discusses how anomalous power consumption in a gateway can be used as a metric to detect malicious activity. Finally a summary of this chapter is provided in Section 4.4.

4.1 Undertaken Measurement Methods

Publications 8 and 9 rely upon external data acquisition equipment to measure the power consumption of the devices under test in real time, and subsequently store the data either locally, or into external storage media. The device under test in Publication 8 was a smartphone which can act as both an end-device, as well as a gateway used by both IP and non-IP edge devices. In Publication 9, commercial consumer-grade access points were used, in which the vendor's firmware was replaced with OpenWRT, a linux-based distribution for embedded system.

In Publication 8, the data acquisition device both supplied as well as measured the power to the smartphone. The device supplied highly detailed measurements in real-time via a USB interface to a PC. However, it was both costly and bulky, as it was custom ordered and made to specifications for detailed power measurements aimed at designing new smartphones. Additionally, it was not portable, as it was mains powered.

Owing to both cost and portability reasons, a different hardware solution was adopted for Publication 9, where a hardware platform was constructed with low cost, off the shelf components that could store data locally and could be powered from its own battery source reliably. Open source software was also used to drive this platform during energy measurements. Owing to its cost-effective design as well as form factor, it was easy to replicate the platform to measure multiple devices at arbitrary locations and points in time.

4.2 Optimising cellular REST communication

When REST-based communication is employed by IoT gateways, they typically either refer to HTTP or CoAP. For this purpose, the usage of these two protocols over different kinds of cellular networks were measured. As Publication 8 describes live 3GPP networks were used based on 2.5G Enhanced Data rates for Global Evolution (EDGE), 3G High Speed Packet Access (HSPA) and 4G LTE. Although it would have been interesting to compare the obtained results for energy consumption against 5G as well as NB-IoT networks, at the time of measurements, these networks were not yet commercially available. Also, the total cost of communication, in terms of power consumption of all the intermediate nodes between the communicating endpoints were not taken into account.

The envisioned use cases considered REST-based communication with remote hosts, to which data aggregated by the smartphone is sent. HTTP was selected not just as a reference point against which CoAP power consumption can be measured, but also because many Web-based platforms predominantly offer HTTP APIs to interact with.

For CoAP, two further sub cases were studied. In its traditional protocol configuration, CoAP uses UDP as its transport protocol. Because it is easy to envision using a smartphone acting as a CoAP server over the WebSocket protocol (and for instance, a web browser interacting with nearby BLE devices), usage of CoAP over WebSockets was also measured.

Two kinds of message exchanges between a client and server supporting these three protocols are depicted in Figure 4.1. Above the dotted line, the client-server messages illustrate how sessions are set up. For CoAP, the messages show how a CoAP Observe relationship is established. For CoAP+WS, the messages describe how a TCP and HTTP session is set up and upgraded to use WebSockets. For HTTP, the messages show a TCP session being established between the client and server.

Below the dotted line, message exchanges describe the transfer of the resource state from the server to the client.

For reference, the use of CoAP over Secure WebSockets secured with TLS was also measured. However session estabishment with TLS is omitted from the diagram for simplicity.

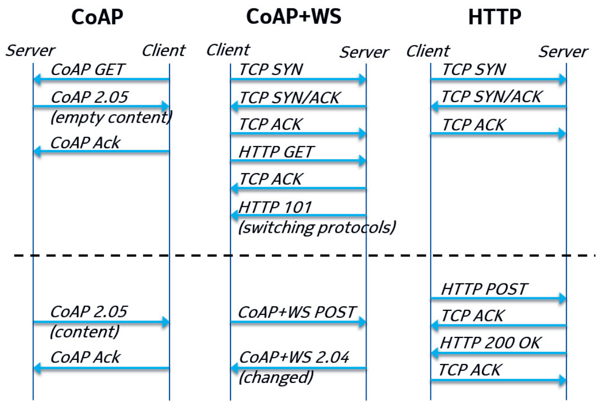


Figure 4.1: REST-based Message exchange.

The hypothesis that IoT gateways and devices can use CoAP instead of HTTP for energy efficient cellular communication was challenged. The subsequent findings did not support the hypothesis conclusively. It was discovered that no circumstances occurred where CoAP was less efficient than HTTP. In several instances, HTTP fared significantly poorly particularly when its handshake and message transmission size pushed radio transmissions into a higher energy consuming state. An example of this is the use of LTE radio, and particularly in LTE networks in which the operator has enabled Discontinuous Reception (DRX), a radio mode offering energy savings for applications that do not need constant data streams. In a DRX-enabled 4G network, CoAP-based communication, owing to its more compact packet sizes and concise message exchanges, were able to exploit energy savings benefit, but HTTP, with its more verbose payloads and greater exchanges of messages, could not. This was particularly apparent when small REST payloads were sent in rapid one-second transmissions.

When radio signalling costs dominate the cellular activity, optimisations on the transport and application layer were not as significant, and the energy consumption of HTTP was comparable to that of CoAP. This occured in EDGE networks when REST communication was performed periodically over ten-second transmissions. Thus if a deployment scenario was known in advance, the usage of HTTP-based communication between a gateway and a remote service can be feasible.

Usage of CoAP over WebSockets, as well as over Secure WebSockets, performed comparatively, with its performance closer to CoAP than to HTTP. This could be explained on how the underlying transport performs in CoAP communication: After the initial HTTP based upgrade handshake to a WebSocket connection, the communication is effectively CoAP over a bidirectional TCP channel supporting framing. The results from these measurements were taken into account when the usage of CoAP was standardised over TCP, TLS and WebSockets as RFC 8323 [8].

From the obtained results, it can be postulated that energy savings are possible for a power-constrained cellular IoT gateway if it possesses the ability to select the kind of 3GPP radio it uses, and has the ability to influence the payload size during transmissions. For small messages sizes, for example, EDGE proved to be most efficient radio.

On the other hand it can also be seen that even with just CoAP-based communication, message sizes, transmission intervals and radio selections can affect power savings significantly. For IoT gateways which wish to deploy power savings in cellular radio connections, the findings from Publication 8 reveal that two factors are therefore important as described below.

- *Aggregate RESTful interactions.* Firstly, the gateway should be able to aggregate RESTful interactions intelligently on behalf of edge IoT devices to ensure a good tradeoff between radio signalling and actual communication, as well as ensure that high powered radio states are avoided as much as possible. This means that in HSPA networks, the gateway can perform Fast Dormancy, or ensures that radio states do not enter into the high powered CELL_DCH (Dedicated Channel) state but instead maintain communication not to exceed the thresholds supplied to remain in the CELL_FACH (Forward Access Channel) state. In LTE networks, enabling DRX support allows significant energy savings for the gateway. However, the timings for both the entry into the CELL_DCH state from the CELL_FACH state, as well as DRX cycles are network operator specific, and consequently require appropriate configuration actions on the gateway.
- *Coordination with IoT devices in the edge network.* Secondly, the gateway needs to co-ordinate as much as possible, communication between the gateway and IoT devices which wish to transmit data. Providing timing hints to end devices allows synchronisation of data transmission from the edge cloud as well as packet reception. Research in this area is limited, although such a scheme was proposed in an Internet draft by Savolainen and Nieminen [93].

4.3 Anomalous Gateway Behaviour

Publication 9 specifically dealt with an advanced smart home architecture based on Homenet, where multiple gateways are present within the residential network. The work was performed in a controlled laboratory environment using four IPv6-enabled Homenet gateways interconnected as an ad-hoc mesh over a dedicated Wi-Fi radio, and which supply Internet connectivity to edge IoT devices and computers over a separate Wi-Fi radio interface in infrastructure mode. The Babel routing protocol was utilised, and energy measurements were undertaken during different kinds of attacks on the routing infrastructure.

The work was initiated with a working hypothesis that under specific kinds of attacks, a Homenet gateway, as well as the network as a whole, would exhibit significant deviation in power consumption. If the hypothesis is valid, then anomalous power consumption in a gateway can be used as a source of information for attack detection.

The energy measurements taken were based on levels of severity in terms of intrusion types. Three different kinds of attack surfaces were targetted: The wireless channel, the routing protocol, and the payload carried by the messages of the routing protocol.

The findings established that eavesdropping attacks were indistinguishable in terms of using energy measurements. However, a Jamming attack involving injecting de-authentication frames towards the link between two gateways caused a more significant surge in power consumption than injecting de-authentication frames just to cause a Denial-of-Service attack on one gateway. Similarly a Denial-of-Service attack in injecting invalid routing packets can be detected, particularly if the incoming packets force the gateways into performing computations, such as checksum verification. The third kind of attack which combined Sinkholes with Denial-of-Service, by constantly flooding new routing information into the network forcing gateways to continually update their routing tables, was extremely noticeable.

The observations can be used to conclude that route flooding attacks and some forms of de-authentication attacks can be detected by using gateway power consumption as a metric. Energy footprints obtained from these attacks were clearly visible. Additionally, route flooding was particularly effective as a battery exhaustion attack, aggressively shortening the operational lifetime of a battery-operated router by up to 50%.

On the other hand, the measurements were taken when no other traffic was present in the network. The energy footprints of simple de-authentication attacks can be masked if, in addition to routing traffic, the Homenet conveys user traffic. Thus, deploying the gateways into a live network to accurately portray network utilisation patterns as well as CPU processing loads, and then measuring the impact of these attacks, would provide additional insight.

4.4 Summary

Energy measurement undertaken in Publications 8 and 9 formed the underpinnings of this chapter. These measurements provided insights as to how energy consumption patterns in IoT can be gleaned for border as well as multi-hop mesh gateways commonly found in IoT network configurations, in two orthogonal ways.

The energy consumption patterns were firstly used towards understanding how application payloads, communication protocols as well as radio networks can significantly impact power consumption when sensor data is transported from the edge network to an external endpoint. From the obtained results, it was seen that the best protocol and radio combination allowed approximately 60% more power savings compared to the worst. Secondly, the energy consumption patterns were used towards understanding how security anomalies can be detected in an edge networks, particularly during active attacks. Here, the findings establish that incidents of eavesdropping attacks or simple Wi-Fi deauthentication attacks cannot be accurately detected in small networks, registering just 23mW in a 4-router configuration that does not have user traffic. However, a route flooding attack establishes a significant energy fingerprint. When a distance-vector routing protocol, such as Babel is used, route updates and routing information exchange among routers become significant traffic wise as the number of routers in the edge network increases. From the results seen, a route flooding attack would have a crippling effect on the routing infrastructure if not detected on time. The findings show that even with just a 4-router network, a network under attack would have each router displaying a sharp increase in average power consumption from 2192mW to 2802 mW which is almost a 28% increase in power consumption per router. Both Publications 8 and 9 show that usage of secure transports and hashing functions have a negligible impact energy-wise, when compared against insecure counterparts.

5 Composing Communication Protocols

This chapter is based on the work done in Publications 3 and 4 for prototyping and implementation of network protocols and communication software, that can reside in IoT gateways and devices. More specifically, it discusses a lightweight C++-based framework called DOORS[1], that was developed for implementing protocols and network services. These range from simple socket-based systems, to protocol stacks, proxying and translation, as well as general event-based client-server applications.

In Publication 3, the core features of the framework are described, such as reusable communication components, abstractions supporting protocols and protocol messages, as well as runtime monitoring of the deployed communication protocols and protocol stacks.

While DOORS is particularly useful for the implementation of application-layer protocols, it can also be used to implement protocols at the transport and network layers. DOORS facilitates the easy addition of experimental features to existing protocol designs. For example, Publication 3 described an implementation of an application layer discovery protocol, the Service Location Protocol (SLP) in IPv6. The advanced features offered by DOORS for protocol design were emphasised in both Publications 3 and 4, by extending SLP with experimental mobility support for automatic service discovery during movement and changes in the network points of attachment.

Section 5.1 describes how DOORS delivers information about hardware characteristics and platform-specific Interprocess Communication (IPC) mechanisms which can aid protocol development for constrained devices and gateways.

Section 5.2 provides and discusses measurements undertaken on the framework in terms of network event handling. Because DOORS works with external event handling libraries in order to perform event dispatching, several different kinds of operating system calls exist for network communication. These are compared against each other using comparable hardware platforms.

Section 5.3 describes a high-level specification language used to specify protocol messages and state machines, which are synthesised into C++ classes, that form the basis of protocol implementations. While Publication 3 provides examples of this specification, this section supplies the full schema definitions used for the protocol specification.

Section 5.4 describes the event monitoring subsystem provided by the framework, which incorporates a user-defined hierarchical symbol interface, that facilitates inspection of message and protocol parameters at run-time.

[1]currently hosted in a public Github repository http://github.com/DOORS-Framework

Finally, Section 5.5 summarizes the main findings.

5.1 Addressing IoT Gateway Hardware and Platform Heterogeneity

DOORS can be compiled and used on all UNIX and Linux variants. It is single-threaded and, with the exception of *libevent()* described in Section 5.2, has virtually no external dependencies towards third-party libraries [2]. The framework does not restrict the protocol developer from flexibly including such libraries or using advanced C++ language-level features such as smart pointers, run-time interrupt handling nor external C++ template libraries.

DOORS employs a modified build system based on the well-known GNU Autoconf tools. This modified subsystem allows fine grained options to tune the build process for specific needs, enabling and disabling the compilation of certain subsystems. This modification was also done so that builds for multiple target architectures could be performed concurrently on the same source tree, something that is impossible with a standard Autoconf configuration. The existence of a separate build directory that is unique to each target architecture provides a significant time savings advantage for testing builds as well as the behaviour of the resulting binaries.

The build system allows straightforward compile-and-execute support for 32-bit and 64-bit x86 and ARM architectures. However it is also capable of cross-compilation and being deployed to embedded systems. As an example, with the appropriate Software Development Kit (SDK) containing architecure-specific toolchains, DOORS-based protocols or network services can be cross-compiled for use in OpenWRT-based gateways and access points, and it is lightweight enough to be ported to an OpenWRT-based TP-Link TL-WR703N, a compact, low-powered, battery operated wifi router measuring 5.7cm x 5.7cm x 1.8cm having 4MB flash memory and 32MB RAM.

At compile time, platform checks are performed and a C++ header file is automatically generated containing hardware and operating system specific information, which a protocol might find useful at run-time. Example information includes the hardware architecture, endianness, storage sizes of primitive types such as integers, availability of IPv6 support and so on.

Although protocol developers can choose to obtain platform-specific information if they want to, the framework provides abstractions that aim to shield developers from operating system specific issues. This is accomplished in DOORS using the concept of virtual devices. These virtual devices typically model aspects of communication activities such as network endpoints and operating system IPC mechanisms such as sockets, pipes and queues. In effect, each virtual device represents a file descriptor at the operating system level.

Using virtual devices, it is possible for DOORS implementations to flexibly incorporate support for different kinds of technologies. For example, Publication 3 described support for Bluetooth communication. In addition, Publication 4 described the extensions made to SLP for use with Mobile IPv6 and IPv6 multicast, while in Publication 3 a more generic, protocol-agnostic mechanism was described that interfaced with an external

[2]While *libevent()* integration is extremely useful for network performance, it is not strictly essential. DOORS can also be compiled and used without it

library capable of detecting changes in network addresses. These were made possible with virtual devices.

Virtual devices belong to the I/O Handling Subsystem in DOORS, with the I/O Handler monitoring virtual devices for I/O activity and performing network event multiplexing and dispatching as necessary. The performance of such I/O multiplexing operations in DOORS is discussed in the next section.

5.2 Measuring network event handlers

For network event multiplexing and dispatching, the I/O Handler in DOORS integrates an event dispatching library, called *libevent* [94]. *libevent* is a highly portable C-language library which provides a uniform interface supporting various low-level I/O multiplexing system calls such as *kqueue()*, *epoll()*, *select()* and *poll()* [3]. Although libevent defaults to using the fastest event dispatching mechanism available to that platform, the protocol engineer is also provided the ability to select the precise operating system event-dispatching mechanism desired (as well as to be avoided). This can be a useful asset should the protocol engineer know the expected traffic characteristics in advance.

Two benchmarking experiments were conducted to measure the differences in behaviour, as well as relative performance, of the four multiplexing system calls above. Both experiments relied on measuring HTTP client-server communication. HTTP loads were generated from a client using the *ab* command-line utility. *ab* is an open-source tool provided by the Apache Foundation, and used for benchmarking HTTP servers by accepting arguments specifying the number of consecutive requests to make, as well as the concurrency of these requests. This load generator resided in a MacBook pro running Mac OS X 10.7.

For the test server, a minimal HTTP server was implemented using DOORS. This server was capable of responding to incoming HTTP GET request for a small named file. The size of this file was 100 bytes. This was pre-cached by the server upon startup, before listening for incoming requests. This minimised the effect of filesystem reads and reduced the amount of computational processing overhead for the server in completing a particular request in terms of TCP/IP segmentation and fragmentation. The server was deployed on an Intel Pentium 4 host running at 2.80GHz, 2 GB RAM, a Gigabit Ethernet network card capable of dual booting Ubuntu 10.04 LTS as well as FreeBSD 9.1.

Both client and server machines were connected via Gigabit switches and all non-essential services were terminated prior to running the experiments.

In the first experiment, the client performed 10000 requests to the server with a concurrency level of 1000 connections. No connections were kept idle. The performance of the server was measured using each event dispatch method. Initially the server was booted into the Ubuntu distribution to measure the performance of *epoll()*, *select()* and *poll()*. Subsequently it was also booted into FreeBSD to measure the performance of *kqueue()*.

Figure 5.1 shows that minor fluctuations in response times appear in all the four plots in the beginning, with *poll()* showing the greatest variance in its transient state. *poll()* also takes slightly longer to reach steady state than the other 3 mechanisms. However, all four dispatch backends used perform comparably upon reaching steady-state. Table 5.1 shows that, in the case where each connection containing an active HTTP request, over time using poll even performs marginally better than other mechanisms.

[3] *kqueue()* is specific to BSD-based operating systems such as FreeBSD, while *epoll()* is specific to Linux-based operating systems

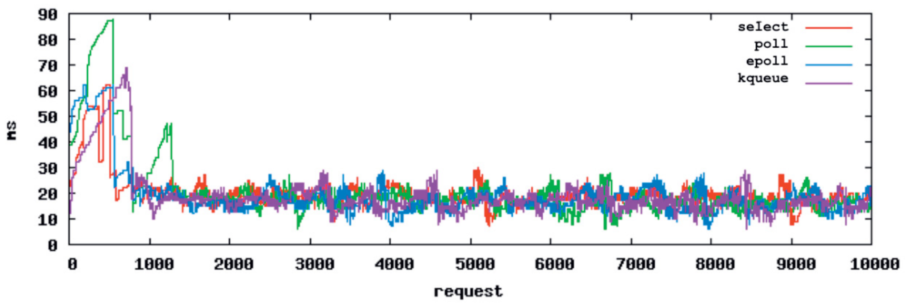


Figure 5.1: 10000 HTTP requests, concurrency level 1000, no idle connections

Table 5.1: Numerical results for Fig 5.1

	select()	*poll()*	*epoll()*	*kqueue()*
Transfer Rate (kb/s)	1142.43	1199.29	1181.18	1166.05
Serving Rate (Requests/s)	7204.61	7570.02	7446.02	7336.76
Total Completion Time (s)	1.388	1.321	1.343	1.363
Mean Completion Time (ms)	0.14	0.13	0.13	0.14

It is important to note that these measurements were conducted to monitor the performances of these mechanisms when there are no idle connections, and all incoming requests are handled. However, it is typical that for some network services residing in IoT gateways as well as end-hosts, that may not be the case. In fact, the active set of events (which represent actively communicating end-points) will be proportionally less than the entire event set of interest (which represent open sessions).

A second experiment was conducted to investigate the performance and efficiency of the event dispatch mechanisms, when the event set of interest is far larger than the active set. To do so, the HTTP server was initially preloaded with 20 000 idle connections. Then, only a small fraction of active connections were enabled and the performance of the different event dispatch mechanisms were measured. This was undertaken to obtain an insight of the impact of such network traffic, where a large number of file descriptors have to be monitored when just a small portion of active connections exist.

A small C program, called *idleconn* was used to open 20 000 connections to the server. *idleconn* is part of a suite of HTTP performance measurement tools, called *httperf*. It maintains a constant, user-defined number of idle connections to any arbitrary server and TCP port until it is explicitly shut down. If an existing connection is closed for any reason, *idleconn* attempts to reconnect and re-establish it immediately. *idleconn* was deployed onto a third connected device, which was a Raspberry Pi 2 Model B.

Figure 5.2 shows the measurement results using *poll()*, *epoll()* and *kqueue()* when 20 000 idle connections exist. Measurements were taken for 5000 requests with a concurrency level of 100 active requests. Because *select()* does not scale to support the large number of open file descriptors, it has been omitted from this experiment.

The graph confirms that the proportion of few active connections to many idle connections in the event set of interest significantly affects the response time when *poll()* is used, while *epoll()* and *kqueue()* are largely unaffected.

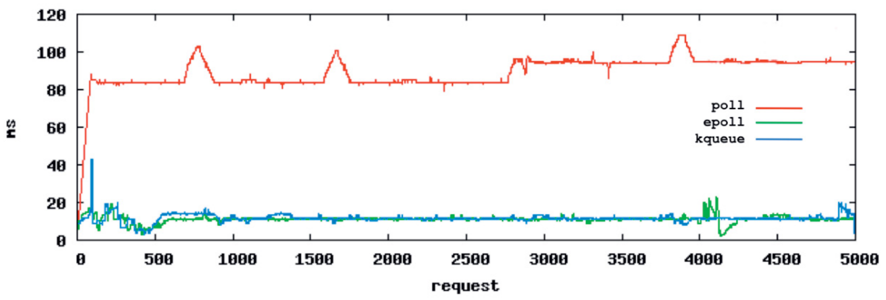


Figure 5.2: 5000 HTTP requests, concurrency level 100, 20 000 idle connections

Table 5.2: Numerical results for Fig 5.2

	poll()	*epoll()*	*kqueue()*
Transfer Rate (kb/s)	163.26	1200.22	1119.43
Serving Rate (Requests/s)	1033.27	7587.25	7082.15
Total Completion Time (s)	4.839	0.659	0.706
Mean Completion Time (ms)	0.97	0.13	0.14

As Table 5.2 shows, the serving rate, transfer rate and mean completion times of *epoll()* and *kqueue()* remain relatively constant to those found in Table 5.1. Using the *poll()* event mechanism on the other hand, performs significantly poorly. An 86% performance degradation is seen in *poll()* when large interest sets having few active connections need to be processed.

5.3 Protocol Specification and Code Synthesis

The primary aim of the DOORS framework is to enable protocol developers to derive their own protocols using the C++ object classes, as well as the tools provided in the code repository. However, several example protocols are also provided as part of the framework that could be utilised for further network or application development. For example, in addition to the IPv6-enabled SLP mentioned in the beginning of this chapter, the framework also contains code implementing other protocols such as CoAP, the Session Announcement Protocol (SAP) as well as the Real-time Transport Protocol (RTP). Example prototypes are also provided for the CoAP Resource Directory as well as application logic for both CoAP clients and CoAP servers.

Protocols implemented with the DOORS framework are initially modelled in terms of their state machines, Protocol Data Units (PDUs) as well as incoming and outgoing messages via Service Access Points (SAPs), using XML. The decision to use XML to describe message and protocol specifications was taken in order to provide a technology neutral platform for expressing many of the protocols and communications mechanisms specified by various SDOs. These XML specifications are then parsed by code generators in DOORS at compile time to produce corresponding framework specific C++ classes such as Messages, Service Access Points, Service Primitives as well as extended finite state machines. These can be directly used as part of an implementation. Three code generators have been developed in order to specify this information using XML. These are the Service Access Point generator, the Peer Generator and the State Machine Generator.

They are described in the subsections below.

5.3.1 The Service Access Point Generator

SAPs in DOORS are represented by a set of classes representing entry and exit points to the service layer or protocol. The SAP XML specifications define sets of message types that the service layer accepts or provides from other layers. These messages are classified into two categories: User messages are accepted via the Service Access Point of the layer, whereas Provider messages are supplied via the Service Access Point to other (generally higher) layers. The XML Schema Definition of the SAP specification the code generator expects, is described in Listing 5.1.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name='SAP'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:User'/>
    <xs:element ref='t:Provider'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name='User'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:Message' maxOccurs='unbounded'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name='Provider'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:Message' maxOccurs='unbounded'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name='Message'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:Parent' minOccurs='0' maxOccurs='unbounded'/>
    <xs:element ref='t:Field' minOccurs='0' maxOccurs='unbounded'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name='Parent'>
  <xs:complexType mixed='true'>
  </xs:complexType>
 </xs:element>

 <xs:element name='Field'>
  <xs:complexType mixed='true'>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Listing 5.1: SAP XML Schema Definition

5.3.2 The Peer PDU Generator

For communication with remote peers, DOORS provides a peer abstraction interface in which the message format of PDUs are specified. Apart from the definition of the message name and message fields, the XML specification of the PDUs also include 2 optional elements: The Header element and the Parent element.

The message structure of a PDU would often consist of a payload, as well as a series of octets prepended to the payload as a header. The payload contains information generated and consumed by higher layers. The header itself can have fixed and variable parts. The header typically contains information pertaining to, for example, protocol versioning, the message type, length, an identifier, the payload content type, payload encoding and possible checksums. As can be inferred, fixed headers do not generally change for different PDUs, but the variable portion of the header can be arbitrarily lengthy or ordered with various fields relevant to a specific type of PDU. Thus, should a PDU contain any fixed headers, they can be specified under the Header element of the XML specification.

When a protocol specification contains PDUs which differ only very slightly (for example, by just a single field), but are largely unchanged, then the Parent element offers a simple way to express such PDUs, by using a pre-existing C++ class as a base. In this case, the Parent would contain the message fields which repeat in every PDU, while the Message element only contains the remaining field or fields specific to that PDU type. Subsequently, in the generated C++ code, this results in a PDU class that is derived from the Parent base class.

While describing the header and payload of a PDU is straightforward in XML and renders easily towards code synthesis, describing serialisation and encoding schemes governing the way the PDU is expressed over the wire, is extremely challenging as they can vary greatly from one protocol to another. Consequently, encoding and decoding rules for each PDU have not been included into the XML specification. Instead, the PDU code generator generates template function calls, *encode()* and *decode()*, for each generated C++ class representing a specific PDU. The XML Schema Definition of the PDU specification the code generator expects, is described in Listing 5.2.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name='Peer'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:Header' minOccurs='0' maxOccurs='1'/>
    <xs:element ref='t:Message' maxOccurs='unbounded'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name='Header'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:Field' maxOccurs='unbounded'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name='Message'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:Parent' minOccurs='0' maxOccurs='unbounded'/>
    <xs:element ref='t:Field' minOccurs='0' maxOccurs='unbounded'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name='Parent'>
  <xs:complexType mixed='true'>
  </xs:complexType>
 </xs:element>

 <xs:element name='Field'>
  <xs:complexType mixed='true'>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Listing 5.2: PDU XML Schema Definition

5.3.3 The State Machine Generator

Listing 5.3 describes the XML schema for the protocol state machine, that the code generator accepts. DOORS uses a transition table-driven design approach in which *State*Input* tuples invoke appropriate event handlers. This facilitated easy synthesis of C++ code as State Machine objects containing function pointers to pre-declared event handling functions. Typically, protocol designers implement these event-handling functions as member functions for C++ objects representing the protocol itself.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name='SM'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:SAP-File' minOccurs='0' maxOccurs='unbounded'/>
    <xs:element ref='t:Peer-File' minOccurs='0' maxOccurs='unbounded'/>
    <xs:element ref='t:From' minOccurs='0' maxOccurs='unbounded'/>
    <xs:element ref='t:State' maxOccurs='unbounded'/>
   </xs:sequence>
   <xs:attribute name='HIncludeFiles' type='xs:string' use='optional'/>
  </xs:complexType>
 </xs:element>

 <xs:element name='SAP-File'>
  <xs:complexType>
   <xs:attribute name='Name' type='xs:string' use='required'/>
  </xs:complexType>
 </xs:element>

 <xs:element name='Peer-File'>
  <xs:complexType>
   <xs:attribute name='Name' type='xs:string' use='required'/>
  </xs:complexType>
 </xs:element>

 <xs:element name='From'>
  <xs:complexType>
   <xs:attribute name='Port' type='xs:string' use='required'/>
  </xs:complexType>
 </xs:element>

 <xs:element name='State'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:Interface' minOccurs='0' maxOccurs='unbounded'/>
    <xs:element ref='t:Timer' minOccurs='0' maxOccurs='1'/>
   </xs:sequence>
   <xs:attribute name='Default' type='xs:string' use='optional'/>
  </xs:complexType>
 </xs:element>

 <xs:element name='Interface'>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref='t:Input' maxOccurs='unbounded'/>
   </xs:sequence>
   <xs:attribute name='Name' type='xs:string' use='required'/>
  </xs:complexType>
 </xs:element>

 <xs:element name='Timer'>
  <xs:complexType mixed='true'>
  </xs:complexType>
 </xs:element>

 <xs:element name='Input'>
  <xs:complexType mixed='true'>
   <xs:attribute name='Name' type='xs:string' use='required'/>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Listing 5.3: State Machine XML Schema Definition

5.4 Monitoring runtime events and protocol parameters

Event-based behaviour is represented in DOORS as base objects called Event Tasks, while protocols are implemented based on a further derivative of the Event Task, called the Protocol Task. Protocol Task objects work together with other specialized objects for aiding protocol development such as finite state machine implementations, Service Access Points, and encoding/decoding of Protocol Data Units, bi-directional Ports used by Protocol Tasks for message passing and multiplexers that allow M:N communications between Protocol Tasks. As described in Section 5.3 XML code generators can be used to obtain finite state machine implementations, Service Access Point classes and messages as well as Protocol Data Units from high level specifications.

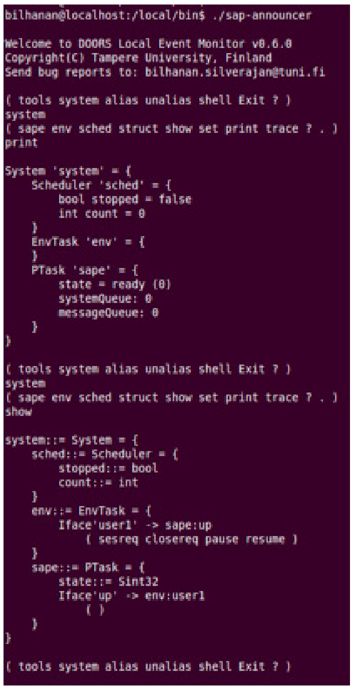
At runtime, a scheduling subsystem has been designed that consists of a scheduler handling requests from Tasks, and providing execution turns to these Tasks using a priority-based callback mechanism. The Subsystem also contains timers that can be used for providing soft real-time guarantees together with a timer manager.

Runtime event-monitoring ties in to the scheduling subsystem, with a text-based event tracing and reporting interface serving as an interaction point between the running system and the protocol developer. This interaction interface is modelled as a DOORS Task called an Environment Task. This allows the monitoring of various parts of a running system such as the state of tasks, internal variables, the number of messages processed, their types, timers as well as the scheduler load. Additionally, the interface can be used to modify data, trigger actions and send messages to interact and influence the runtime behaviour of the communication system

The event-monitoring system employs a hierarchically organized symbol-handling meta-interface in order to attach and observe the current state of running DOORS systems. Symbols are stored in a symbol table, and each symbol can arbitrarily represent message fields, tasks, C++ datatypes as well as user defined structures. Symbols also are arranged in a parent-child relationship with other symbols. For example, an Event Task symbol would contain child symbols such as those representing the state machine, the message queues, and timers. Relationship between symbols can vary dynamically at runtime as well. A good example of this would be symbols representing messages as they traverse through a protocol stack, being queued and processed by different tasks.

Figures 5.3 and 5.4 depict how the event monitoring system looks like to a protocol developer at runtime. Both these figures are based on an IPv4 implementation of the Session Announcement Protocol, a UDP-based protocol, which is used to announce multicast sessions in the network.

Figure 5.3 illustrates a simple hierarchically contextual command line interface accepting user queries and conveying results. A query for "*system print*" reveals the running scheduler, datatypes of tasks and internal variables, the messages the Environment Task accepts from the user, and the Service Access Point between the Environment Task and the PTask implementing the Session Announcement Protocol. A subsequent query for "*system show*" reveals the current runtime state of the scheduler and tasks.

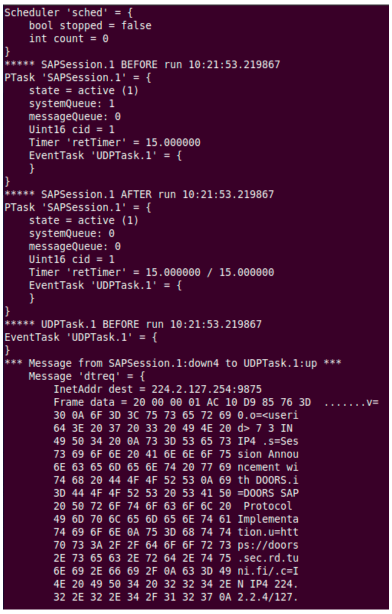


```
bilhanan@localhost:/local/bin$ ./sap-announcer

Welcome to DOORS Local Event Monitor v0.6.0
Copyright(C) Tampere University, Finland
Send bug reports to: bilhanan.silverajan@tuni.fi

( tools system alias unalias shell Exit ? )
system
( sape env sched struct show set print trace ? . )
print

System 'system' = {
    Scheduler 'sched' = {
        bool stopped = false
        int count = 0
    }
    EnvTask 'env' = {
    }
    PTask 'sape' = {
        state = ready (0)
        systemQueue: 0
        messageQueue: 0
    }
}

( tools system alias unalias shell Exit ? )
system
( sape env sched struct show set print trace ? . )
show

system::= System = {
    sched::= Scheduler = {
        stopped::= bool
        count::= int
    }
    env::= EnvTask = {
        Iface'user1' -> sape:up
            ( sesreq closereq pause resume )
    }
    sape::= PTask = {
        state::= Sint32
        Iface'up' -> env:user1
            ( )
    }
}

( tools system alias unalias shell Exit ? )
```

Figure 5.3: DOORS Event monitoring depicting system objects

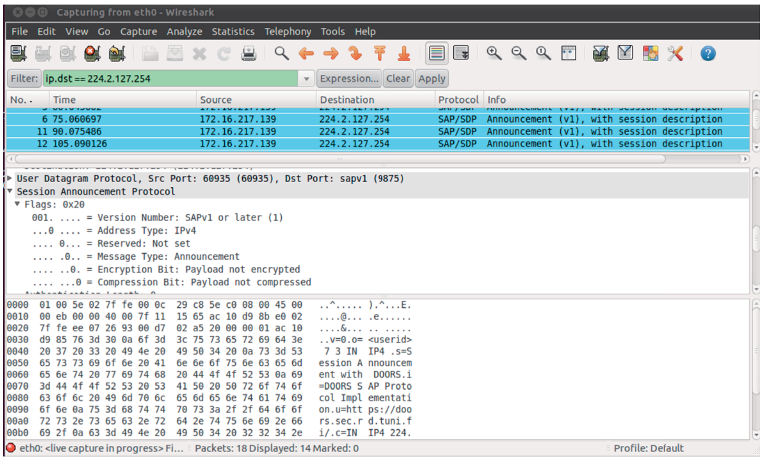
Figure 5.4 on the other hand shows runtime information pertinent to message handling and the operation of the protocol. The figure has been truncated for brevity. The Protocol

Task “*SAPSession.1*”, representing a single announcement session, is initially shown with a message in its system queue, representing a 15 second timeout value. Upon the timeout, *SAPSession.1* creates a protocol packet containing the session information and delivers it to a lower level *UDPTask* containing a UDP socket, to subsequently trasmit the announcement to the UDP multicast address 224.2.127.254.



```
Scheduler 'sched' = {
    bool stopped = false
    int count = 0
}
***** SAPSession.1 BEFORE run 10:21:53.219867
PTask 'SAPSession.1' = {
    state = active (1)
    systemQueue: 1
    messageQueue: 0
    Uint16 cid = 1
    Timer 'retTimer' = 15.000000
    EventTask 'UDPTask.1' = {
    }
}
***** SAPSession.1 AFTER run 10:21:53.219867
PTask 'SAPSession.1' = {
    state = active (1)
    systemQueue: 0
    messageQueue: 0
    Uint16 cid = 1
    Timer 'retTimer' = 15.000000 / 15.000000
    EventTask 'UDPTask.1' = {
    }
}
***** UDPTask.1 BEFORE run 10:21:53.219867
EventTask 'UDPTask.1' = {
}
*** Message from SAPSession.1:down4 to UDPTask.1:up ***
    Message 'dtreq' = {
        InetAddr dest = 224.2.127.254:9875
        Frame data = 20 00 00 01 AC 10 D9 85 76 3D  .......v=
        30 0A 6F 3D 3C 75 73 65 72 69 0.o=<useri
        64 3E 20 37 20 33 20 49 4E 20 d> 7 3 IN
        49 50 34 20 0A 73 3D 53 65 73 IP4 .s=Ses
        73 69 6F 6E 20 41 6E 6E 6F 75 sion Annou
        6E 63 65 6D 65 6E 74 20 77 69 ncement wi
        74 68 20 44 4F 4F 52 53 0A 69 th DOORS.i
        3D 44 4F 4F 52 53 20 53 41 50 =DOORS SAP
        20 50 72 6F 74 6F 63 6F 6C 20  Protocol
        49 6D 70 6C 65 6D 65 6E 74 61 Implementa
        74 69 6F 6E 0A 75 3D 68 74 74 tion.u=htt
        70 73 3A 2F 2F 64 6F 6F 72 73 ps://doors
        2E 73 65 63 2E 72 64 2E 74 75 .sec.rd.tu
        6E 69 2E 66 69 2F 0A 63 3D 49 ni.fi/.c=I
        4E 20 49 50 34 20 32 32 34 2E N IP4 224.
        32 2E 32 2E 34 2F 31 32 37 0A 2.2.4/127.
```

Figure 5.4: DOORS Event monitoring depicting message handling


Capturing from eth0 - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: ip.dst == 224.2.127.254
Expression... Clear Apply

```
No. .  Time          Source            Destination      Protocol  Info
     6 75.060697     172.16.217.139    224.2.127.254    SAP/SDP   Announcement (v1), with session description
    11 90.075486     172.16.217.139    224.2.127.254    SAP/SDP   Announcement (v1), with session description
    12 105.090126    172.16.217.139    224.2.127.254    SAP/SDP   Announcement (v1), with session description

▶ User Datagram Protocol, Src Port: 60935 (60935), Dst Port: sapv1 (9875)
▼ Session Announcement Protocol
  ▼ Flags: 0x20
      001. .... = Version Number: SAPv1 or later (1)
      ...0 .... = Address Type: IPv4
      .... 0... = Reserved: Not set
      .... .0.. = Message Type: Announcement
      .... ..0. = Encryption Bit: Payload not encrypted
      .... ...0 = Compression Bit: Payload not compressed

0000  01 00 5e 02 7f fe 00 0c  29 c8 5e c0 08 00 45 00   ..^..... ).^...E.
0010  00 eb 00 00 40 00 7f 11  15 65 ac 10 d9 8b e0 02   ....@... .e......
0020  7f fe ee 07 26 93 00 d7  02 a5 20 00 00 01 ac 10   ....&... .. .....
0030  d9 85 76 3d 30 0a 6f 3d  3c 75 73 65 72 69 64 3e   ..v=0.o= <userid>
0040  20 37 20 33 20 49 4e 20  49 50 34 20 0a 73 3d 53    7 3 IN  IP4 .s=S
0050  65 73 73 69 6f 6e 20 41  6e 6e 6f 75 6e 63 65 6d   ession A nnouncem
0060  65 6e 74 20 77 69 74 68  20 44 4f 4f 52 53 0a 69   ent with  DOORS.i
0070  3d 44 4f 4f 52 53 20 53  41 50 20 50 72 6f 74 6f   =DOORS S AP Proto
0080  63 6f 6c 20 49 6d 70 6c  65 6d 65 6e 74 61 74 69   col Impl ementati
0090  6f 6e 0a 75 3d 68 74 74  70 73 3a 2f 2f 64 6f 6f   on.u=htt ps://doo
00a0  72 73 2e 73 65 63 2e 72  64 2e 74 75 6e 69 2e 66   rs.sec.r d.tuni.f
00b0  69 2f 0a 63 3d 49 4e 20  49 50 34 20 32 32 34 2e   i/.c=IN  IP4 224.

eth0: <live capture in progress> Fi...   Packets: 18 Displayed: 14 Marked: 0        Profile: Default
```

Figure 5.5: Wireshark capture of multicast session announcements

The usage of the event monitors does not preclude the usage of other available tools for inspection of protocol packets or application profilers provided by the operating system. For example, Figure 5.5 is an independent, simultaneous capture of protocol packets on the wire using the well-known Wireshark application. The timestamps depict packets being sent by the DOORS implementation every 15 seconds, with correlating protocol headers, payloads and destination addresses.

5.5 Summary

Developing a gateway for various IoT networks requires a good understanding of the domain in which it would be deployed. Apart from bridging and supplying connectivity to the nodes in the edge network, a gateway can also engage in a variety of other functions. These include:

- Network-level protocol translation, in which an intermediary gateway enables end-to-end communication by allowing interoperability between dissimilar protocols and data
- Proxying, in which an intermediary gateway processes and manipulate packet headers without heavy interference into the payload
- Discovery support, in which a gateway actively engages in communication with edge nodes to support group-based and advanced application services
- Lookup support, in which a gateway serves as a registry of services that edge nodes can query directly for.

In all these cases, the communication subsystem and protocol stacks the gateway employs, play a key role in fulfilling these functions. When the required communication functionality needs to be designed and implemented as part of the gateway, a protocol implementation framework offers a feasible route for developers to undertake. From the work done with regards to the design and development of the DOORS framework for protocol composition in IoT gateways, this chapter concludes with the main findings which are described in the rest of this section.

Firmware and software development for IoT devices and gateways continue to pose a challenge in supporting a large heterogeneity in platform architectures. Implementation frameworks must firstly be able to remain useful to produce code which portable across different hardware and operating systems, but also, deliver platform specific information to running applications if needed. The Autoconf-based build system that DOORS has, achieves this. As described in Section 5.1, platform specific information is detected by the build system and written to a specific header file, allowing implementations to remain portable and tailor runtime behaviour to the platform. Increased portability was also achieved by limiting, or entirely eliminating, dependencies on external libraries.

How steep the learning curve of the framework is, particularly in relation to the anticipated lifetime of the resulting implementation, is important. Should the need to rapidly prototype, code and evaluate an implementation be necessary, the framework should not only be able to offer the correct kinds of protocol classes, message types and object handlers, but also be intuitive to use. DOORS facilitates this by providing an easy to understand XML-based specification language, a protocol abstraction model, as well as code generators that directly follow on to produce C++ code based on defined protocol messages and states written in XML. The event monitoring interface was also kept deliberately simple and hierarchical, allowing developers and administrators to remotely access a console on an IoT gateway to monitor running communication protocols and log events very effectively, similar to existing practices on current Internet routers and switches.

In constrained environments where computing and memory requirements can't afford the execution of multiple toools at runtime, the DOORS event monitoring subsystem provides substantial benefits for locally observing, testing and troubleshooting of protocols running in embedded systems such as gateways. The observation of variable values and operational states of executing tasks do not have any dependencies on source-level debugging formats, and hence protocol implementations do not need to be compiled with debugging support. This allows efficient optimisations for code size and execution time. Because the hierarchical symbolic interface used by the event monitor is implemented as preprocessor macros, they can also be independently and easily disabled when no longer required. While existing network protocol analysers such as Wireshark and tcpdump are prevalent for debugging well-known protocols, the DOORS event monitor is highly useful for monitoring new protocols and protocol features. Additionally, it can also be used to monitor protocols for non-IP networks that an IoT gateway can be connected to, and for which no support is available in other third-party protocol analysers.

Lastly, the capacity of the framework to interwork and interoperate with other frameworks, toolkits and applications is also important. A framework that offers a tightly coupled, highly integrated development environment may not be as effective as a framework offering a cleanly decoupled model with clear callback interfaces and communication infrastructure. By adopting this decoupled approach, DOORS remains scalable from resource constrained gateways, such as OpenWRT access points and older generation Model A and Model B Raspberry Pi boards, to more powerful Linux and UNIX-based computing platforms. A key factor for achieving this was with the DOORS I/O Handling and virtual device objects, that enable monitoring callbacks and passing execution turns with external libraries and applications. Together with the C++ classes provided, this makes DOORS a highly suitable implementation framework that can be used to develop a wide range of IP-based, as well as non-IP based communication protocols and protocol stacks that can be deployed into IoT gateways.

6 Gateway Management

The purpose of this chapter is to discuss the issues of IoT gateway management work that was performed in Publications 5, 6 and 7.

Gateways are an integral part of the IoT, and therefore occupy a crucial position in modern IoT management systems. These systems aim to use lightweight methods to perform lifecycle management of IoT endpoints, including bootstrapping, updating and decommissioning. However, the part a gateway plays in IoT network management can vary significantly, since it is closely tied to the network topology, the type of connectivity available, proxying/caching functions, as well as security requirements. The placement of management components within the overall architecture can also be affected, depending on the network topologies, traffic characteristics, usage scenarios and the types of gateways being used.

Section 6.1 provides an overall abstraction of how end-to-end management of IoT endpoints is usually perceived. Section 6.2 provides details of gateway management patterns that were observed in this dissertation. These patterns can also be perceived as a specialisation of the Device-to-Gateway communication pattern in RFC 7452 [95], with a stronger emphasis on the role of IoT gateways as explicit components in the management architecture. Section 6.3 provides an analysis of some key operational and deployment considerations that pertain to gateways in the context of end-to-end management of both IoT gateways and devices. The chapter then concludes with a summary and key findings.

6.1 Managing IoT endpoints

A simple depiction of end-to-end management between a management service and an IoT device is shown in Figure 6.1

Taken at its simplest abstraction, IoT management practices assume the following:

- A server with properties such as high scalability and availability hosts the management service that can potentially manage very large numbers of IoT devices. Typically, this can also mean a cloud-based system
- All IoT endpoints to be managed are, to a large extent, IP-enabled to allow communication with the management server. The management service interacts with the managed devices using IoT protocols (such as CoAP, MQTT or proprietary protocols) over IP.
- For applications requiring access to resources, data is retrieved from the management service instead of directly from the managed devices, using a REST-based API offered by the management service.

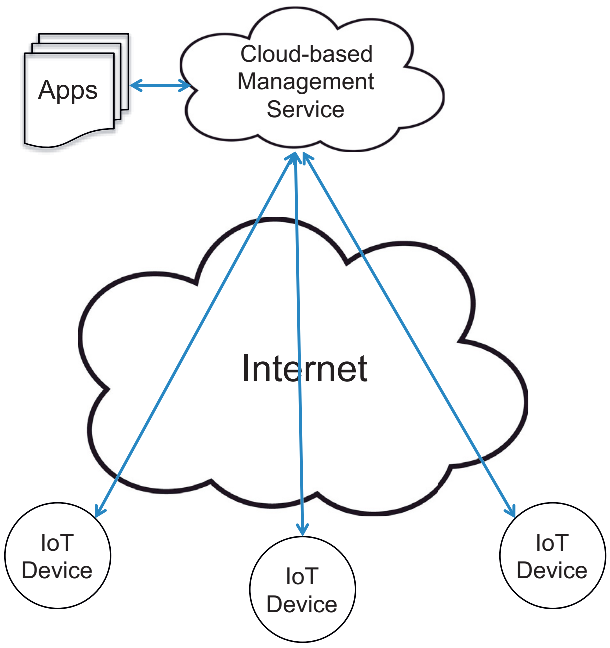


Figure 6.1: End-to-end IoT device management.

Commercial IoT device management services offered today, such as Google Cloud IoT, Amazon Web Services and the IBM Watson IoT platform [96] are built along these assumptions. They do not prescribe guidelines for device connectivity; supplying Internet connectivity and ensuring the IoT device remains reachable by the management service remains the responsibility of the device owner or network operator. Once connected, however, these cloud-based management services offer secure provisioning, monitoring as well as data retrieval from these connected IoT devices. The aggregated data is then subsequently processed, or is directly consumed by applications.

Viewed this way, gateway management is directly conflated with device management; a management service does not discern the management of a gateway with that of any other kind of IoT endpoint. However, managing the proper operation of IoT gateways extends beyond configuration and monitoring, since any changes by a management service to gateways serving edge networks can have an adverse effect on the management of IoT endpoints residing in edge networks. Thus, different types of gateway management, as well as deployment and implementation considerations are discussed in the next sections.

6.2 IoT Gateway Management Patterns

During the analysis, design and implementation work performed for Publications 5, 6 and 7, several repeatable practices for performing IoT gateway management were observed. These are detailed as six gateway management patterns described in the following subsections. In Patterns 1, 2 and 3, gateway management is influenced by the kinds of underlying network topologies selected for IoT device deployment. These are then followed by management patterns that are context-specific used to overcome certain challenges across several application domains and integration scenarios.

6.2.1 Pattern 1: LAN with IP-based IoT devices

When a managed network setup and configuration is needed to supply IP connectivity to IP-enabled IoT devices, a gateway needs to be introduced as an intermediary component into the architecture. The sensors and actuators present in this management pattern are assumed to possess the ability to use either Ethernet or Wi-Fi to obtain connectivity to the LAN. In such cases, the management service exerts control over the configuration, monitoring and updating of the gateway. Additionally organisational LAN policies may warrant the need for the gateway to perform operations related to NAT and middlebox traversal. For example, management operations supported by the IoT device can be based on UDP as the transport (which is the case with LWM2M), but the communication between the gateway and cloud may employ TCP instead. This pattern is used for gateway management in Publications 5 and 6. Additionally, IoT gateway configuration management as described in Publication 2 for connectivity provisioning, is aligned with this pattern too.

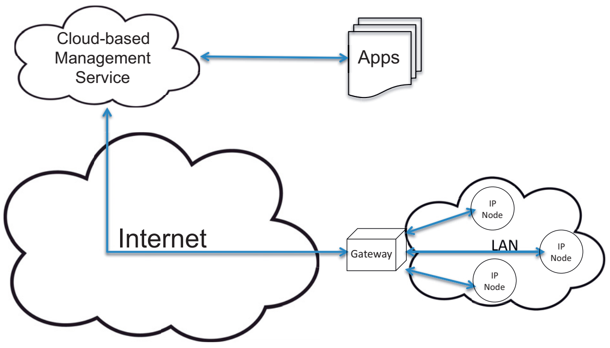


Figure 6.2: Pattern 1

6.2.2 Pattern 2: PAN with non-IP IoT devices

Figure 6.3 illustrates a common management pattern, where a gateway is introduced to interconnect devices in a Personal Area Network (PAN) to the Internet. Devices in the PAN use a different network layer technology from the one used by the management server. This can occur for example, if the IoT device uses a legacy or non-IP network, such as Zigbee, Bluetooth or BLE. To manage non-IP devices, the management server interacts explicitly with the gateway. IP-based management operations terminate at the gateway. Application-logic in the gateway translate protocol requests and responses into technology-specific operations to interact with the IoT device. Such a pattern is employed in Publication 6 to allow a LWM2M server to interact with non-IP nodes via an IoT gateway, aided by object and resource instances the gateway registers to the LWM2M server on behalf of these non-IP IoT devices.

6.2.3 Pattern 3: Mesh-based edge network

Mesh-based networks have become increasingly popular as a means to extend coverage and connectivity for IoT deployments at the edge. As opposed to traditional infrastructure networks, mesh topologies offer resilience against partial failures of wireless links if the nodes relaying traffic in the mesh are within radio coverage of at least two nodes. Figure 6.4 illustrates the gateway interconnecting the mesh to the Internet, acting as a border

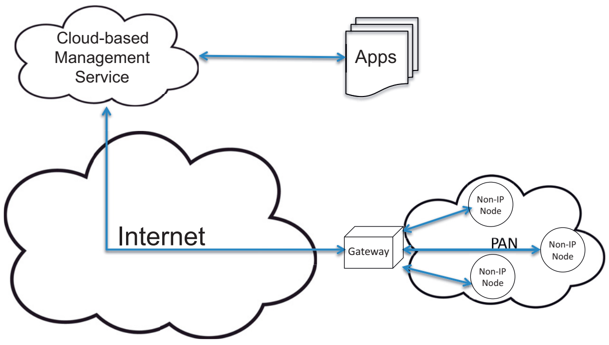


Figure 6.3: Pattern 2

router. However, gateways can take on additional roles in mesh networks as well, based on the type of mesh network present, as well as the types of network nodes present in the mesh. These types are presented below:

- A non-IP mesh network, such as a BLE mesh network in which the role of the gateway would be similar to that of Pattern 2, in order to manage all the BLE nodes in the mesh.
- A low power IP mesh network, such as 6LowPAN may have differing requirements on the gateway. When the IoT device in the mesh is so constrained to the extent that management functionality cannot be feasibly deployed on its memory, the gateway combines the role of serving as a management proxy in addition to being a 6LowPAN border router.
- A Wi-Fi mesh network, in which multiple gateways form a core, routable mesh network, offering connectivity to edge IP nodes. Multiple gateways need to be managed in the mesh network, in addition to the border router. This is the strategy employed in Publication 5.

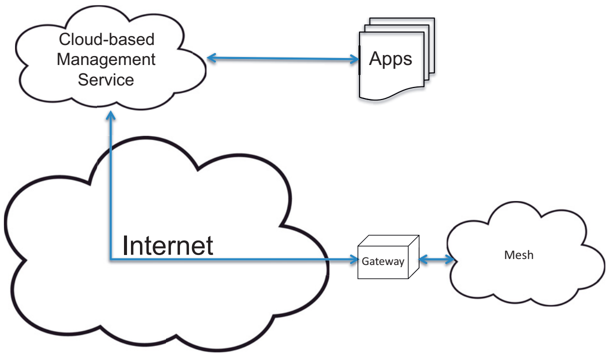


Figure 6.4: Pattern 3

6.2.4 Pattern 4: Gateway fault tolerance

Gateway downtime can adversely impact endpoint management, such as when continuous monitoring of sensor data from edge networks is required by a cloud-based management service. A common strategy used to prevent a gateway becoming a single point of failure, is to introduce redundant gateways into the architecture, as shown in Figure 6.5. This pattern is employed very effectively in Publication 7 to allow a secondary gateway to assume an active role in case of any failures with a primary gateway.

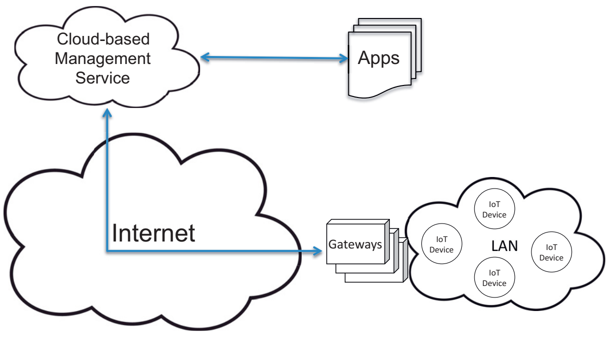


Figure 6.5: Pattern 4

Failures can be caused by misbehaving software as well as malfunctioning hardware. Additionally, this pattern is employed in systems to minimise downtime during regular maintenance and firmware update operations on gateways. Introducing gateway redundancy into the management architecture differs from Pattern 3, where multiple gateways can be active at once. The primary gateway would assume the role of the default, active gateway while secondary gateways, with independent uplinks to the management service, would be configured to be in standby mode.

6.2.5 Pattern 5: Disruption tolerance

Previous management patterns have assumed constant connectivity between the management service and the edge network's gateways and devices. Connectivity disruption can have an adverse effect on management operations. This can occur for several reasons such as:

- Mobility: The sensors and gateway form part of a geographically mobile network in which intermittent connectivity is the norm
- Power savings: The gateway can be energy constrained based on available battery power, and economises radio usage by powering down its network interfaces on occasion
- Link characteristics: The link between the gateway and the Internet is inherently unreliable, or is lossy.

In this context, the design of a disruption tolerant management system is necessary. Disruption tolerance refers to the ability of the various components in the LWM2M

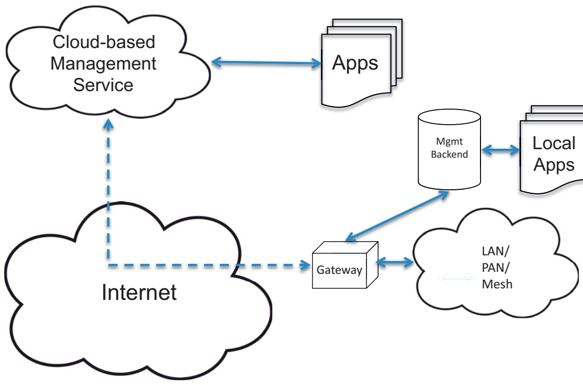


Figure 6.6: Pattern 5

architecture to cope and be resilient when the uplink of the border gateway exhibits intermittent connectivity and continue operating even when the edge network is disconnected from the Internet.

As an example, gateways, sensors and end devices in a personal area network may need to continue and communicate with a management service without service disruption, or applications may need to be supplied information constantly.

The dominant management pattern to allow disconnected operations is to collocate a primary management service within the same network topology as the managed endpoints, to ensure reliable communication with managed entities in the network. A secondary service resides in the cloud as a resource cache.

Application logic is therefore required to cope with synchronisation of managed data between the two services. Hence, the primary management service usually resides as a software component on the gateway, to facilitate easy detection of connectivity resumption with the Internet. This application-level logic would then be responsible for communicating with cloud-based applications interested in receiving endpoint data.

Publication 5 provides an example in which this pattern is employed in which the primary REST-based management service resides as part of the edge network, with a secondary service in the cloud. Synchronisation between the two is performed between the two in a best-effort manner to provide eventual consistency between the resource states in the primary service and the secondary service.

6.2.6 Pattern 6: Legacy platform integration

Many organisations have existing investments in proprietary management platforms and contain deployments that cannot be easily migrated to newer interoperable management solutions that integrate into Web-based platforms today. Consequently, a common way to integrate vendor-specific or legacy management systems is to use an application-level gateway that performs the high-level translations necessary to deliver management operations from the management service and the specific IoT Device residing within a different management architecture. This is depicted in Figure 6.7.

It can be seen that this management pattern restricts the ability of the management service to perform end-to-end management to the physical devices present in the legacy

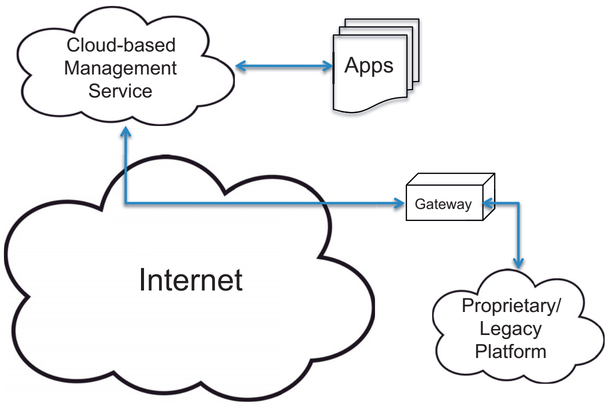


Figure 6.7: Pattern 6

platform. Operations terminate at the gateway, and consequently, actual management operations at the last hop are opaque to the management service. Additionally, as opposed to other patterns introduced in this section, the gateway in this case can be deployed as a software service, either residing in the cloud, or as a value-added service for the vendor-specific platform.

The work done for the dissertation does not specifically investigate legacy integration of external management systems and frameworks. However, this specific management pattern is nevertheless an important facet of how gateways are used today.

6.3 Operational considerations

Owing to the wide variety of usage scenarios, devices and topologies present in the IoT, several of the described management patterns can also be combined. Minor variations on the same patterns can occur too. Furthermore, the patterns discussed are not exhaustive. It is anticipated that, from new developments in the lifecycle management of gateways and device, additional patterns will emerge. Advances in hardware design, computational capabilities and storage capacities in future gateway would also have a major influence in IoT management in terms of edge computing, lightweight virtualisation, and deployment of application logic into gateways. In this section, some key considerations are presented for gateway management in IoT. Where possible, the discussion is presented in the context of using LWM2M as the gateway management protocol.

6.3.1 Similarities and differences with IoT device lifecycle management

While some aspects of the gateway's lifecycle management are analogous to IoT device lifecycle management, additional steps need to be taken, particularly in the pre-operational phase of the gateway. The pre-operational management phase for a gateway refers to the measures that need to be taken in order to transition the gateway into an operational state. More specifically, onboarding, bootstrapping and staging operations need to be performed.

- *Onboarding.* From the perspective of an IoT device, onboarding refers to obtaining connectivity into the LAN. However, from the perspective of an IoT gateway, the onboarding operation refers to the steps undertaken in order for the gateway's northbound network interface to be connected to the Internet and reachable from the management service. Consequently, a pre-established trust relation as well as authentication credentials need to exist, initially between the ISP and the IoT gateway for IP connectivity, and subsequently between IoT management service provider and the IoT gateway.

- *Bootstrapping.* The bootstrapping operation can be either gateway or server initiated upon obtaining Internet connectivity. A dedicated bootstrap server ensures additional security credentials, such as keying material or certificates, are configured into the gateway. The result of the bootstrapping operation is an assurance of the identities of both the gateway and the management server to each other, as well as possible communication privacy based on the use of an encrypted channel.
- *Staging.* Upon completion of bootstrapping, IoT gateways can be managed as if they are standard IoT devices. Staging of the gateway can be necessary before the gateway can enter into its operational phase. Staging in the context of gateway management refers to the ability for the management service to deliver additional operations to the gateways, in order to extend their role towards supplying connectivity for the LAN. Between the bootstrapping and staging, a soft reset of the gateway might be necessary. Some of the operations during staging include:
  - Configuring the properties of one or more south bound radio or wired interfaces to supply connectivity to edge devices.
  - Configuration and deployment of advanced services allowing configuration of edge devices.
  - Configuration of security settings such as access control lists, passwords and firewall rules.

Current IoT device management standards to support onboarding, bootstrapping and staging are still evolving. For example, the data model for LWM2M supplies the Security and Access Control Objects which are used for bootstrapping general IoT devices, but data models and operations for onboarding as well as staging did not exist. Consequently the development and design of the Gateway System Object, Gateway Fixed and Wireless Interface Objects as well as the Gateway Firewall Object, undertaken as part of this dissertation work, were adopted for usage by the IPSO Alliance as extensions to the LWM2M object model.

6.3.2 Moving from Device Masquerading to Device Proxying

A distinguishing feature of an IoT gateway is its ability to communicate and serve its resources over two or more connection endpoints. When Pattern 2 is implemented using LWM2M, the gateway performs registration of the LWM2M Objects representing the IoT endpoint to the LWM2M server. All subsequent LWM2M operations performed by the server are then translated at the gateway into the operations to manipulate or retrieve data in the IoT endpoints. Results are relayed to the server.

Publication 6 described new LWM2M objects which would allow the registration of endpoints by IoT gateways, but preserve the semantics of the gateway serving as a proxy to an endpoint, instead of masquerading the gateway itself as the IoT endpoint.

Migrating an IoT gateway away from device masquerading to provide endpoint visibility to a management server, can be accomplished at three levels:

- Using protocol semantics that enable the management server to perceive the gateway as an intermediary component. For example, CoAP, which is used both by LWM2M as well as OCF, explicitly supports the notion of a Proxy URI. A proxy URI can be used to separate the identification of a resource at an endpoint as well as its location, from an intermediate network device employed to communicate with the endpoint.
- Providing application logic or a commissioning context to a management server or a lookup facility during registration of an endpoint to be managed. The CoRE Resource Directory, which supports CoAP-based registration support for endpoints and resources, offers support orthogonally for both the usage of a commissioning tool (which aids the registration of nodes too constrained to perform registration on their own) as well as a proxy (which can be used at run time to retrieve and manipulate resources on a registered endpoint).
- Establishing explicit Proxy objects within the data model supported by the management platform. Gateways registering proxy objects can then contain resource fields or web links which then describe the data models of the endpoints that are being assisted.

6.3.3 Undertaking Gateway Failovers and Redundancy Management

Pattern 4 described redundancy as part of a gateway management strategy to reduce intermediate points of failure in managing sensors and obtaining sensor data from edge networks. Redundant gateways provide high availability to sensor data by allowing failover with one or more secondary gateways on standby. Such failovers can be either initiated by the management service, or by the gateways themselves. Using LWM2M as an example, it is initially assumed that all gateways register themselves to the LWM2M server. The IPSO Gateway System object, described in Publication 6, of each gateway can also be extended with a new resource field, “Gateway State”, which reflects the role of each gateway in the redundancy model.

For a low latency edge network in which it is essential to minimise data loss from IoT devices to the management service, a gateway initiated failover model,as described in Publication 7, can be used. For this model, a secondary gateway remains in hot standby state and monitors the state of the primary gateway using a heartbeat protocol. Should a suspected failure on the primary gateway occur, the secondary gateway immediately performs failure recovery by announcing itself as the default gateway at the edge network. Combining the approaches of using the IPSO Gateway System Object from Publication 6 with a "Gateway State" resource, with the solution outlined in Publication 7, the secondary server can change the value of its “Gateway State” resource into “primary”. The LWM2M Server can subsequently either modify the value of the previous gateway’s “Gateway State” resource field to “secondary”, or remove its information entirely should the registration become stale after a given time.

When the scenario also includes the use of non-IP devices in the edge network (as with Pattern 2), the steps outlined above for redundancy management are insufficient to minimise data loss. This is primarily due to the fact that in Pattern 2, from the perspective of a LWM2M server, the gateway also registers LWM2M objects which map to the native or proprietary data model of the non-IP devices in the edge network. Data transmitted from the sensors are first intercepted by the gateway before being adapted, serialised and transmitted by the gateway over the Internet, using a content type format that the LWM2M server accepts. Consequently, in the event of a failure, not only does the secondary gateway need to ensure that connectivity between the cloud and edge network is restored, it also needs to re-register the LWM2M objects specific to the non-IP devices as well as resuming protocol translation and serialisation of the sensor data. Thus, when non-IP devices are present in an edge network, not only does the secondary gateway need to monitor the liveliness of the primary gateway, it also has to keep track of the timestamps of the last sensor data values successfully transmitted by the primary gateway. This is described in Publication 7.

In constrained networks where redundant gateways serve as border routers, a server initiated failover strategy can also be employed. In this scenario, secondary gateways are kept in cold standby, where no monitoring or heartbeat protocols are performed between the redundant gateways. Should a primary gateway fail to update its registration to the LWM2M server within the allocated time, the server elects a secondary gateway with an active registration, changing the status of its Gateway State resource into “primary”. This then initiates failover operations by the newly elected gateway to become the default active border router of the edge network.

6.3.4 Multiparty Gateway and Device Management

As greater numbers of gateways and IoT devices become integrated into the Web of Things, more complex IoT topologies will become inevitable. This is already evident in current smart buildings in which smart lighting systems, utility networks, environmental and ventilation systems can be federated with the IT network infrastructure. In such environments, if the gateways and devices belonging to multiple stakeholders and owners need to interact with each other, authorisation mechanisms and proper access rights to sensor data is required.

Publication 5 describes how, in the context of a smart home, sophisticated gateway management and configuration cannot always be performed by a home network owner. The work in the paper stems from proposed future networking architectures for the home, such as the Homenet architecture [16], which requires advanced configuration and deployment of network services. When the Homenet becomes coupled with multi-tenancy by introducing third-party owned fire alarms, remote surveillance and smart metering sensors, expert management becomes necessary to collaboratively co-manage the home network.

However, existing IoT management practices do not easily cater for performing collaborative management or management of multi-tenant networks and devices at the edge. IoT gateways that natively support multi-tenancy are not common although both research prototypes and commercial solutions exist [97]. In such cases, gateway configuration still requires a single point of control and administration.

Research activities in cloud computing and cloud architectures on the other hand have led to the development of multi-tenant systems in the cloud: Data residing in a cloud-based service can be presented as well as visualised differently depending on the needs as well as access rights of the stakeholder. Thus, while gateways and IoT devices at the edge can be directly configured and managed by a centralised service, Publication 5 proposes that it is more feasible for collaborative management to be performed as a cloud-based application service which interacts with the management service instead.

6.4 Summary

This chapter focused on research and work done for the management of IoT gateways. A high-level architecture was first presented for how IoT endpoints are typically managed today in Section 6.1. Then gateway management patterns were presented in 6.2 as findings, based on studies done for this dissertation. Table 6.1 summarises the usage of these patterns in the three main publications described in this chapter. As integration with legacy management frameworks was not explored in this dissertation, the column representing Pattern 6 is blank.

Table 6.1: Summary of IoT gateway management patterns used in publications

Gateway Patterns	**1**	**2**	**3**	**4**	**5**	**6**
Publication 5	x		x		x	
Publication 6	x	x				
Publication 7				x		

Section 6.3 identified the key issues for IoT gateway management and how this dissertation provided solutions for them. There are still numerous challenges that need to be solved in order to have a standardised way of performing gateway management. Because existing network management practices used in enterprise-level networks to manage enterprise-grade routing and switching equipment is infeasible for IoT networks, the work done in this dissertation is to align gateway management practices with IoT device management.

REST-based communication, using CoAP and HTTP, between a management server and managed gateways is a key factor towards this alignment. Additionally, it was shown that LWM2M is extremely suitable as a management standard for gateway management. However, since the data models that exist for LWM2M supported only management of end devices, this dissertation developed data models that could model the functionality of IoT gateways. These could then be incorporated into existing LWM2M management systems to monitor both gateways as well as connected sensors and devices.

Another finding in this dissertation is that it is vital to model the gateway accurately enough for configuration management, and perhaps even more so than device management. A misconfigured and malfunctioning gateway, particularly one that serves as a border gateway or an entry point to an entire edge network, can have serious repercussions for the operation of all IoT devices and sensors. In addition to ensuring that a gateway has been configured properly, LWM2M allows the monitoring the operational performance of gateways in terms of observing various parameters such as CPU loads, bandwidth characteristics, memory and storage. When the operational thresholds are exceeded, such as with energy consumption as described in Publication 9, anomalous behaviour can be rapidly detected.

It was also seen that it would be beneficial to integrate software-based event management systems, such as the DOORS event monitor described in Chapter 5, as it would be easy to model the hierarchical symbolic interface as a specific set of data models, to expose the running state of communication protocols. However, that work has not been performed as part of this dissertation.

7 Conclusion

The IoT is a continuously evolving environment which calls for protocols, network designs, management and service architectures that can cope with billions of IoT entities. IoT gateways are vital components in enabling this to happen, by interconnecting the sensing, computing and actuating systems with the networking domain.

IoT gateways need to cope with new radio technologies, sensors, embedded systems and heterogeneous communication strategies to securely connect the suppliers of the data with the consumers. By recognising and abstracting the many roles that IoT gateways need to perform, the main goal of this dissertation is simplifying the integration and interoperability of edge IoT devices to the Internet, and facilitating remote configuration and management. Where possible, the practical approaches developed in the dissertation also placed heavy emphasis on the use of standardised communication protocols and device management.

7.1 Results

The research problem of this dissertation was divided into four research questions as described in Section 1.2. These research questions and the answers presented in this dissertation are summarised next.

Research Question 1

What do the connectivity characteristics, communication support and data reachability for current and future IoT network topologies require from IoT gateways?

Connectivity intermittence, network address renumbering, communication and addressing support for device heterogeneity and scalability were identified as key challenges in IoT networks. Network topologies for IoT were described and solutions were given for how IoT gateways can be deployed to meet the connectivity requirements expected by IoT devices. Deploying a self-healing mesh network of IoT gateways as advocated by the Homenet architecture, provided solutions for automatic address configuration, network disruption tolerance and scalability to support large scale connectivity for IoT edge devices. IPv6 networking was particularly considered an important step towards enabling globally unique identifiers for IoT devices. Hence, solutions were provided for how an IoT gateway can facilitate the deployment of IPv6 to edge devices. In Publication 1, the IoT edge gateway was shown to work in tandem with a connectivity provider or ISP to deliver IPv6 addressing and reachability to edge networks and devices. In the absence of such support from the ISP, support for an IPv6 transitioning or tunnelling mechanism must be configured to an IoT edge gateway. A solution to employ IoT gateways to decouple end-device address reachability and data reachability was also described. This equips the

IoT gateway to facilitate REST-based proxying, where RESTful communication occurs between two endpoints using different transports or REST protocols.

References: Publications 1, 2, 5 and 8, dissertation chapter 3.

Research Question 2

How can energy measurements be used to improve and monitor gateway operation?

Energy consumption patterns were measured for IoT gateways in two different contexts. When optimising communication for energy efficiency, an IoT gateway can achieve power savings of up to 60% by selecting the correct cellular radio and REST-based protocol for communication. This indicates that IoT gateways should co-ordinate and aggregate RESTful interactions with edge IoT devices to achieve better energy consumption. Energy consumption of an IoT gateway can be used as a run-time metric to detect certain types of anomalous or malicious activity when compared against baseline measurements. The dissertation showed this was a particularly effective out-of-band strategy in distinguishing jamming, battery draining, denial of sleep, and denial of service attacks, with each producing distinct energy footprints from an IoT gateway under attack.

References: Publications 8 and 9, dissertation chapter 4

Research Question 3:

How can the development of protocols and communication subsystems in IoT gateways be performed rapidly and consistently?

A lightweight protocol implementation framework that facilitates the rapid creation of protocols, protocol stacks and network services called DOORS was developed. Communication protocol logic and messages are specified using in XML from which C++ code is generated. As opposed to working at the programming language level, high level protocol specifications allow easy additions and modifications to existing protocol behaviour. The building and compilation tool-chain aims at portability in order to deploy the resulting implementation across several hardware architectures for IoT gateways. The framework supports native and cross-compilation tool-chains across several different hardware architectures and UNIX- and Linux-based desktop and embedded systems. This allows consistent development of network protocols and communication subsystems that could be implemented across heterogeneous IoT gateways. At run-time, the framework is able to supply consistent diagnostic and operational information of the running protocol stack or network service, as well as flexibly migrate among different low-level network event dispatch mechanisms if needed.

References: Publications 3 and 4, dissertation chapter 5

Research Question 4:

What are the crucial aspects of IoT gateway management that need to be considered, and how can gateway management be performed and aligned with IoT device management?

Misconfigured and mismanaged gateways can negatively impact IoT edge networks and device reachability several orders of magnitude greater than misconfigured or misbehaving IoT devices. The role an IoT gateway plays in the network greatly influences different aspects of its management. In terms of IoT gateway lifecycle management, this dissertation focused on solutions for gateway configuration during bootstrapping, operational monitoring, and failure and redundancy management. Six architectural patterns for IoT gateway management were described. REST-based gateway management and integration with device management standards, particularly LWM2M were deemed important. Consequently, solutions for extending LWM2M for gateway management with new gateway-specific data models were implemented. IPSO data models were designed that allowed LWM2M servers to interact with proprietary data models, as well as with non-IP devices by using IoT gateways as communication proxies, to minimise device masquerading. A REST-based solution for multiparty collaborative management of IoT gateways and networks, using a cloud-based management service was developed. A redundancy management solution for IoT gateways was described, that can be used to prevent loss of critical communication and data transfer between a management service and edge devices.

References: Publications 5, 6, and 7, dissertation chapter 6

7.2 Summary of Contributions

The dissertation scope and contributions were described in Chapter 1.3. As a summary, however, this dissertation contributes to the scientific body of knowledge in the following ways:

- Eight basic IoT network topologies were identified for end devices, with respect to IPv6 addressing needs. Of these, the dissertation details IPv6 address allocation steps that IoT gateways must provide for seven of these scenarios.
- The dissertation shows that IPv6 transition mechanisms can be used without any significant impact on IPv6 reachability and communication with IoT end devices, in the absence of operator support for IPv6.
- The dissertation provided empirical evidence that when IoT gateways engage in REST-based communication over a 3GPP network, undertaken measurements indicate that energy consumption can be markedly reduced if the deployment scenario is known in advance. While the best performing combination was the use of CoAP over a 2G network, energy consumption was significantly influenced by the choice of RESTful protocol, payload sizes and trade-offs with radio signalling costs.
- The feasibility of using energy consumption data to detect security anomalies and attacks on IoT gateways was demonstrated.
- A lightweight event-driven framework called DOORS was designed with which communication protocols and protocol stacks can be implemented, deployed and monitored in IoT gateways.
- Six IoT gateway management patterns were identified from the studies done in the dissertation. The dissertation produced standardised data and interaction models for IoT gateways, which allow REST-based management of both gateways and end devices using well-known IoT device management standards.

7.3 Discussion and Future Directions

The research and resulting work of this dissertation have aimed to facilitate IoT gateway development at the device, networking and application levels of abstraction. The work done in the dissertation is also applicable to many of the IoT application domains and vertical segments, with promising future directions. In this section, some final discussions and future research directions are given.

7.3.1 IPv6 and IoT

IPv6 has clear benefits compared to IPv4 in terms of a larger unique global address space, easier address allocation, mobility, renumbering and overall routing efficiency. For constrained networking, 6LowPAN networking is advocated as an efficient form of IPv6 supporting header compression, energy efficiency and the ability to be processed by devices with limited processing capabilities. Thus, delivering and supporting IPv6 in edge networks are featured strongly in this dissertation. For backhaul connectivity, SDOs including the IETF and the Thread Group, are looking towards IPv6 networking as a foundation to meet anticipated challenges wrought by the IoT. The Thread Group's networking protocol, in fact, is built entirely over 6LowPAN. In terms of deployments, DHCPv6 Prefix Delegation is already supported by the firmware of certain commercial residential broadband gateways while operator deployments of IPv6 over 3GPP networks has become a reality. On the other hand IPv6 adoption and deployment has not been as rapid as expected. Despite address exhaustion, IPv4 still remains in popular and active use today. The use of network address translation, extensive private subnetting and regional Internet registries reclaiming and reusing unused IPv4 address spaces, have formed the basis of mitigation efforts to continue IPv4 usage. Dualstack approaches and the use of IPv6 transition mechanisms are expected to play increasingly important roles for Internet connectivity for IoT.

7.3.2 Impact of Application State

A related discussion pertaining to NAT utilisation and private IPv4 addresses in edge networks, is that usage of NAT and NAT tables creates application state in an edge gateway, as opposed to stateless IPv6 addressing, forwarding and routing into edge networks. Minimising application state can be of significant benefit to reduce application-level processing and memory consumption, both for constrained IoT gateways as well as to lower packet forwarding latencies for IoT gateways in topologically flat edge networks consisting of large numbers of IoT devices.

Nevertheless, as Publications 6 and 7 show, IoT gateways are also extensively used to carry application-layer state. Thus, in addition to packet forwarding, the specific integration challenges that IoT presents, means that gateways will be compounded with application state to interconnect for data sharing with legacy back-end systems, translate packets and data for the management of non-IP devices and networks, or, as described by RFC 7228, assist Class 0 devices. In IoT domains where gateways are based on COTS hardware, the likelihood of gateway failure, and subsequent loss of application state, is higher than with enterprise-grade gateway equipment. Thus, gateway redundancy and redundancy management such as that described in Publication 7, would become essential. However, research is still needed for the design and development of a reliable mechanism to synchronise application states and protocol interaction among redundant gateways in a standardised manner. As an example, this is still an ongoing issue for enterprise networks and enterprise-grade routers; High Availability routers using redundancy protocols such as the Hot Standby Routing Protocol (HSRP) [98] or the Virtual Router Redundancy Protocol (VRRP) [99], offer limited support for stateful NAT failovers.

7.3.3 Gateway Management

IoT gateway management would continue to develop towards several interesting directions. Semantic interoperability is an area of active research in the IoT. Today, the various SDOs pursuing device management standards have largely converged towards using REST-based communication between a management server and managed IoT devices. However, the data models, serialisation and content formats utilised by these standards for retrieving and manipulating resources and objects are still largely divergent and remain domain-specific. Also as Publication 6 demonstrates, in addition to data models conforming to a specific management standard, devices and gateways can also be shipped with proprietary or vendor-specific data models. Future research into semantic interoperability can aim at preventing a proliferation of disparate data models and standards, and instead aim towards harmonising them such as via automatic runtime translation or the use of hypermedia for dynamically generating client-server interaction models. An initial foray into this research has already been performed by the author of this dissertation, towards a semantic repository allowing the discovery, publishing and distribution of multiple kinds of data models for IoT device and gateway management [100].

A further point of note worth mentioning regarding IoT gateway management is that, IoT gateways are increasingly becoming more intelligent, with advanced application logic and computational capabilities. As edge and fog computing become more prevalent, software-based communication systems, network services and data processing applications would be deployed from centralised servers into IoT gateways, either as lightweight container and virtualisation technologies, or as native executables. Future work in IoT gateway management would need to consider the deployment, monitoring, updating and decommissioning of such containers and virtual machines running within the gateways as well.

7.3.4 IoT Gateways in Home Networks

The home network, which Publications 2, 5 and 9 focus upon, offers a rich testing and proving ground for IoT experimentation and deployment. It is an evolving environment with a limited physical space that can contain diverse sensors, embedded systems, powerful computing nodes and smart consumer electronic devices, all of which communicate using different IP and non-IP wireless technologies, and sometimes need multiple gateways for connectivity and interoperability. Additionally, although it is increasingly common to find Commercial Off-The-Shelf (COTS) hardware in industrial IoT domains, as Publication 7 shows, it is commonly in the smart home domain that low-cost COTS equipment are prevalently found. Publications 5 and 9 employ the use of multiple COTS gateways within the home in a self-organising and self-healing multi-hop mesh over Wi-Fi, to scalably support connectivity for a large number of edge IoT devices. Solutions for mesh-based sensor networks have been widely deployed in smart cities and smart building, with proprietary technologies as well as based on standards such as Zigbee and BLE mesh networking. However, Wi-Fi based mesh networking such as the Homenet architecture described in Publications 5 and 9 is a recent phenomenon. ISPs as well as commercial vendors have begun rolling out residential Wi-Fi mesh gateways for home owners wishing to geographically extend Internet coverage, strengthen signal strength or counter connectivity blind spots. The Thread Group, focusing on IoT technologies for the home, is standardising low power IP-based wireless mesh networking standards and protocols over the IEEE 802.15.4 wireless radio [101]. As a future direction, mesh-based connectivity to the Internet within the home would become commonplace and home gateway implementations must consider issues such as autonomic networking, simplified gateway management and deployment, in order to cater towards non-technical home owners and users.

7.3.5 REST-based Proxying

The work described in Publication 6 focused on IoT gateways possessing multiple network interfaces acting as proxies between entirely different protocol stacks and radio technologies. However, gateways can also perform proxying between two disparate transports for the same management or application protocol. As an example, the LWM2M protocol relies on using CoAP for obtaining management and resource information, with CoAP itself has been standardised to use UDP and DTLS. However LWM2M is also specifying TCP and TLS as viable bearer transports for cellular networks, as network management functionality in IoT can often be hampered owing to the existence of middleboxes such as firewalls and NATs between the management server and the managed nodes. Additionally, the use of Websockets has also been proposed for consideration as an alternative transport for CoAP, for similar reasons. A LWM2M gateway thus can function as a proxy for LWM2M/CoAP clients over UDP in its LAN, while using a TCP or Websocket endpoint in its WAN for communication with a LWM2M server. However, the LWM2M standard does not really address the proxying of management operations. Therefore, with the currently available data model, the gateway needs to masquerade the objects and resources of each endpoint on its LAN as if they were its own, to an external management server.

Additionally, as Publication 6 discusses, the discovery and usage of proxy functionality should be incorporated into future data models which unambiguously assert the location and type of managed endpoints residing behind such a gateway. Although the CoAP protocol supports a *Proxy-URI* option, how this can be achieved and defined in a Data Model remains an open issue. However, in doing so, the semantic model becomes expressive enough to cleanly separate end-to-end resource retrieval and interaction at runtime, from any underlying transport-specific reachability issues. As the gateway resides in the path between a management server and managed endpoints, a similar principle can also be applied, in which the gateway functions as a translator for very resource constrained endpoints, providing semantic interoperability for connected end devices. In terms of caching and validation, an added benefit of this role is that the gateway then possesses and provides a cached representation of the schema itself, and is able to locally validate the translation of objects and resources exposed by a very constrained endpoint. Additionally a gateway acting as a CoAP proxy may be requested to retrieve the same resource from a CoAP endpoint over multiple transports. This could be possible if such an endpoint exposes its resources over both UDP and DTLS (or UDP and TCP), for example. Instead of retrieving the same requested resource representation multiple times, the gateway can ideally returned an already cached, valid representation if one exists. Although some work exists in understanding retrieval of CoAP-based resources over multiple transports from an endpoint, it is still in its infancy.

7.3.6 Acquiring Energy Measurements

From the work performed in obtaining energy measurements in Publications 8 and 9, it became apparent that there is significant benefit in obtaining and utilising energy measurements to understand IoT gateway behaviour in several different dimensions, either with hardware or software. Acquiring energy measurements using a hardware-based approach as opposed to software, has certain drawbacks. The approach is not fine grained enough to pinpoint power consumption information of all the various running components in the measured device. Consequently, it is vital that during data acquisition, non-essential applications and services are shut down in the measured device to avoid false positives. Secondly, it is also less convenient in terms of experimental setup, compared to software-based systems, and calibration is often necessary before any measurements are undertaken.

Having an external platform for measuring power consumption has several benefits. External measurement equipment are device agnostic and the same equipment can be used to measure a wide range of IoT gateways and devices. The approach is non-invasive and eliminates any measurement bias caused by a software-based energy profiler, typically as a result of a computational or I/O skew. Particularly for a compromised gateway in which the firmware may have been maliciously replaced, a software profiler may not be accurately detect any rogue activity. Additionally, measurements can be taken reliably even when the measured device is under heavy stress or close to power depletion. The dissertation shows that such measurement equipment can be built to be portable, with low cost and common off-the-shelf components. Consequently, it is feasible to deploy a large number of such equipment to simultaneously measure the energy consumption of all gateways and devices in a network.

The measurement devices could themselves be managed in a standardised manner, such as with LWM2M either over a different or the same backhaul network from the IoT devices being measured. Using external measurement devices also is advantageous for the future in being able to derive meaningful comparisons of different emerging wireless radio technologies without having to develop custom measurement software.

7.3.7 Security and Privacy

Finally, security and privacy research have a significant role in ensuring the success of the IoT. This is an extensive field of study in its own right. Some of the work performed in the dissertation, particularly that of Publication 9, shed light into how network-based attacks on gateways could be identified, based on specific energy footprints. Also, energy comparisons in Publications 8 and 9 show that usage of secure transports and hashing functions have a negligible impact energy-wise, when compared against insecure counterparts. For the future, work would focus on the secure lifecycle management of IoT gateways, particularly with secure bootstrapping, secure firmware updates as well as trusted execution of application gateway logic.

Bibliography

[1] B. Ebeling, S. Hoyer, and J. Bührig, "What are your favorite methods?-an examination on the frequency of research methods for is conferences from 2006 to 2010." in *ECIS*, 2012, p. 200.

[2] IPv6-test.com, "IPv6 test statistics," [Online; accessed 24-July-2018]. [Online]. Available: http://web.archive.org/web/20180724155456/http://ipv6-test.com/stats/

[3] 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.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 97–129. [Online]. Available: https://doi.org/10.1007/978-3-642-19157-2_5

[4] C. Bormann, M. Ersue, and A. Keranen, "Terminology for constrained-node networks," Internet Requests for Comments, RFC Editor, RFC 7228, May 2014, http://www.rfc-editor.org/rfc/rfc7228.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7228.txt

[5] A. Taivalsaari and T. Mikkonen, "A taxonomy of iot client architectures," *IEEE Software*, vol. 35, no. 3, pp. 83–88, 2018.

[6] Technavio, "Global industrial iot gateway market - drivers and forecast from technavio," [Online; accessed 21-March-2019]. [Online]. Available: http://web.archive.org/web/20170420190456/https://www.businesswire.com/news/home/20170420005801/en/Global-Industrial-IoT-Gateway-Market---Drivers

[7] ABI Research, "Gateways Power Nearly Every IoT Market as ABI Research Forecasts Global Shipments to Exceed 64 Million Units in 2021," [Online; accessed 21-March-2019]. [Online]. Available: http://web.archive.org/web/20190321141121/https://www.abiresearch.com/press/gateways-power-nearly-every-iot-market-abi-researc/

[8] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets," RFC 8323, Feb. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8323.txt

[9] O. for Economic Co-operation and Development, *Frascati manual 2015: guidelines for collecting and reporting data on research and experimental development.* OECD Publishing, 2015.

[10] D. Aksnes, G. Sivertsen, L. T. Van, K. Wendt *et al.*, "Measuring the productivity of national r&d systems: challenges in cross-national comparisons of r&d input and publication output indicators," *Science and public policy*, vol. 44, p. 13, 2017.

[11] C. R. Kothari, *Research methodology: Methods and techniques.* New Age International, 2004.

[12] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *Journal of management information systems*, vol. 24, no. 3, pp. 45−77, 2007.

[13] IETF sunset4 mailing list, “Closing sunset4,” 2018, [Online; accessed 25-October-2018]. [Online]. Available: https://web.archive.org/web/20181025051445/https://mailarchive.ietf.org/arch/msg/sunset4/KgD6anjNnqK5i6KWcaP8PQ4y7No

[14] Google IPv6, “Per-country ipv6 adoption,” [Online; accessed 25-October-2018]. [Online]. Available: https://www.google.com/intl/en/ipv6/statistics.html#tab=per-country-ipv6-adoption&tab=ipv6-adoption

[15] S. Ziegler, C. Crettaz, L. Ladid, S. Krco, B. Pokric, A. F. Skarmeta, A. Jara, W. Kastner, and M. Jung, “Iot6−moving to an ipv6-based future iot,” in *The Future Internet Assembly.* Springer, 2013, pp. 161−172.

[16] T. Chown, J. Arkko, A. Brandt, O. Troan, and J. Weil, “Ipv6 home networking architecture principles,” Internet Requests for Comments, RFC Editor, RFC 7368, October 2014.

[17] O. Troan and R. Droms, “Ipv6 prefix options for dynamic host configuration protocol (dhcp) version 6,” Internet Requests for Comments, RFC Editor, RFC 3633, December 2003.

[18] J. Chroboczek, “The babel routing protocol,” Internet Requests for Comments, RFC Editor, RFC 6126, April 2011.

[19] J. Chroboczek, “Homenet profile of the Babel routing protocol,” Internet Engineering Task Force, Internet-Draft draft-ietf-homenet-babel-profile-07, Jul. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-homenet-babel-profile-07

[20] J. Chroboczek, “Applicability of the Babel routing protocol,” Internet Engineering Task Force, Internet-Draft draft-ietf-babel-applicability-05, Nov. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-babel-applicability-05

[21] N. Kushalnagar, G. Montenegro, and C. Schumacher, “Ipv6 over low-power wireless personal area networks (6lowpans): Overview, assumptions, problem statement, and goals,” Internet Requests for Comments, RFC Editor, RFC 4919, August 2007, http://www.rfc-editor.org/rfc/rfc4919.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4919.txt

[22] J. Hui and P. Thubert, “Compression format for ipv6 datagrams over ieee 802.15.4-based networks,” Internet Requests for Comments, RFC Editor, RFC 6282, September 2011, http://www.rfc-editor.org/rfc/rfc6282.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6282.txt

[23] Internet Society, “State of IPv6 Deployment 2017,” [Online; accessed 2-February-2019]. [Online]. Available: http://web.archive.org/web/20190327143648/https://www.internetsociety.org/wp-content/uploads/2017/08/IPv6_report_2017-0606.pdf

[24] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, and C. Gomez, "Ipv6 over bluetooth(r) low energy," Internet Requests for Comments, RFC Editor, RFC 7668, October 2015.

[25] M. Chakraborty and N. Chaki, "An ipv6 based hierarchical address configuration scheme for smart grid," in *2015 Applications and Innovations in Mobile Computing (AIMoC)*, Feb 2015, pp. 109–116.

[26] H. Shin, E. Talipov, and H. Cha, "Spectrum: Lightweight hybrid address autoconfiguration protocol based on virtual coordinates for 6lowpan," *IEEE Transactions on Mobile Computing*, vol. 11, no. 11, pp. 1749–1762, Nov 2012.

[27] C. Y. Cheng, C. C. Chuang, and R. I. Chang, "Lightweight spatial ip address configuration for ipv6-based wireless sensor networks in smart grid," in *2012 IEEE Sensors*, Oct 2012, pp. 1–4.

[28] Z. Zou, K.-J. Li, R. Li, and S. Wu, "Smart home system based on ipv6 and zigbee technology," *Procedia Engineering*, vol. 15, pp. 1529 – 1533, 2011, cEIS 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877705811017851

[29] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The internet of things has a gateway problem," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '15. New York, NY, USA: ACM, 2015, pp. 27–32. [Online]. Available: http://doi.acm.org/10.1145/2699343.2699344

[30] N. Rouhana and E. Horlait, "Bwig: Bluetooth web internet gateway," in *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*, 2002, pp. 679–684.

[31] 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*, vol. 13, no. 5, pp. 6687–6712, 2013. [Online]. Available: http://www.mdpi.com/1424-8220/13/5/6687

[32] J. Meduna, "A lighting interface to wireless network," in *Proceedings of the IAB Workshop on Interconnecting Smart Objects with the Internet.* Internet Architecture Board, 2011. [Online]. Available: https://www.iab.org/wp-content/IAB-uploads/2011/03/Meduna.pdf

[33] W. Jung, S. I. Kim, and H. S. Kim, "Ontology modeling for rest open apis and web service mash-up method," in *The International Conference on Information Networking 2013 (ICOIN)*, Jan 2013, pp. 523–528.

[34] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," Internet Requests for Comments, RFC Editor, RFC 7252, June 2014, http://www.rfc-editor.org/rfc/rfc7252.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[35] A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk, "Guidelines for mapping implementations: Http to the constrained application protocol (coap)," Internet Requests for Comments, RFC Editor, RFC 8075, February 2017.

[36] F. Kaup, P. Gottschling, and D. Hausheer, “Powerpi: Measuring and modeling the power consumption of the raspberry pi,” in *39th Annual IEEE Conference on Local Computer Networks*, Sept 2014, pp. 236–243.

[37] F. Astudillo-Salinas, D. Barrera-Salamea, A. Vázquez-Rodas, and L. Solano-Quinde, “Minimizing the power consumption in raspberry pi to use as a remote wsn gateway,” in *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, Nov 2016, pp. 1–5.

[38] B. Martinez, M. Montón, I. Vilajosana, and J. D. Prades, “The power of models: Modeling power consumption for iot devices,” *IEEE Sensors Journal*, vol. 15, no. 10, pp. 5777–5789, Oct 2015.

[39] K. Gomez, R. Riggio, T. Rasheed, D. Miorandi, and F. Granelli, “Energino: A hardware and software solution for energy consumption monitoring,” in *2012 10th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, May 2012, pp. 311–317.

[40] B. Dezfouli, I. Amirtharaj, and C.-C. Li, “Empiot: An energy measurement platform for wireless iot devices,” *arXiv preprint arXiv:1804.04794*, 2018.

[41] A. Merloa, M. Migliardib, and P. Fontanellia, “Measuring and estimating power consumption in android to support energy-based intrusion detection,” *Journal of Computer Security*, vol. 1, pp. 1–7, 2014.

[42] A. Pathak, Y. C. Hu, and M. Zhang, “Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof,” in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys ’12. New York, NY, USA: ACM, 2012, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168841

[43] S. Rajasegarar, C. Leckie, and M. Palaniswami, “Anomaly detection in wireless sensor networks,” *IEEE Wireless Communications*, vol. 15, no. 4, pp. 34–40, Aug 2008.

[44] R. Jurdak, X. R. Wang, O. Obst, and P. Valencia, *Wireless Sensor Network Anomalies: Diagnosis and Detection Strategies.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 309–325. [Online]. Available: https://doi.org/10.1007/978-3-642-17931-0_12

[45] A. Kanev, A. Nasteka, C. Bessonova, D. Nevmerzhitsky, A. Silaev, A. Efremov, and K. Nikiforova, “Anomaly detection in wireless sensor network of the smart home system,” in *2017 20th Conference of Open Innovations Association (FRUCT)*, April 2017, pp. 118–124.

[46] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma, “Carat: Collaborative energy diagnosis for mobile devices,” in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys ’13. New York, NY, USA: ACM, 2013, pp. 10:1–10:14. [Online]. Available: http://doi.acm.org/10.1145/2517351.2517354

[47] IBM, “5724-x71 ibm rational sdl suite,” [Online; accessed 21-March-2019]. [Online]. Available: http://web.archive.org/web/20190328112105/http://www-01.ibm.com/common/ssi/printableversion.wss?docURL=/common/ssi/rep_sm/1/897/ENUS5724-X71/index.html&request_locale=en

[48] R. Z. ITU-T and Z. Recommendation, “100: Specification and description language (sdl),” *International Telecommunication Union*, 2000.

[49] Y.-D. Bromberg, L. Réveillère, J. L. Lawall, and G. Muller, “Automatic generation of network protocol gateways,” in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing.* Springer, 2009, pp. 21–41.

[50] L. Burgy, L. Reveillere, J. Lawall, and G. Muller, “Zebu: A language-based approach for network protocol message processing,” *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 575–591, 2011.

[51] J. Mercadal, L. Réveillere, Y.-D. Bromberg, B. Le Gal, T. F. Bissyandé, and J. Solanki, “Zebra: Building efficient network message parsers for embedded systems,” *IEEE Embedded Systems Letters*, vol. 4, no. 3, pp. 69–72, 2012.

[52] C. Liu, Y. Cui, C. Zhang, and J. Wu, “Generic application layer protocol translation for ipv4/ipv6 transition,” in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[53] F. Travostino, E. Menze, and F. Reynolds, “Paths: Programming with system resources in support of real-time distributed applications,” in *Proceedings of WORDS’96. The Second Workshop on Object-Oriented Real-Time Dependable Systems.* IEEE, 1996, pp. 36–45.

[54] M. Hayden, “The ensemble system,” Cornell University, Tech. Rep., 1998.

[55] M. A. Hiltunen, R. D. Schlichting, X. Han, M. M. Cardozo, and R. Das, “Real-time dependable channels: Customizing qos attributes for distributed systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 600–612, 1999.

[56] F. Bai, N. Sadagopan, and A. Helmy, “Brics: A building-block approach for analyzing routing protocols in ad hoc networks-a case study of reactive routing protocols,” in *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)*, vol. 6. IEEE, 2004, pp. 3618–3622.

[57] F. Bai, G. Bhaskara, and A. Helmy, “Building the blocks of protocol design and analysis: challenges and lessons learned from case studies on mobile ad hoc routing and micro-mobility protocols,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 3, pp. 57–70, 2004.

[58] D. C. Schmidt, “The adaptive communication environment: An object-oriented network programming toolkit for developing communication software,” 1993.

[59] A. Fettig, *Twisted network programming essentials.* " O’Reilly Media, Inc.", 2005.

[60] M. Colagrosso, W. Simmons, and M. Graham, “Demo abstract: Simple sensor syndication,” in *Proceedings of the Fourth ACM SenSys Conference*, 2006, pp. 377–378.

[61] P. Martı-Gamboa, “A framework for the evaluation of protocols and services in ad-hoc networks,” Ph.D. dissertation, University of Dublin, 2006.

[62] Open Mobile Alliance, “Lightweight Machine to Machine Technical Specification Version 1.0.2,” Feb 2018. [Online]. Available: http://web.archive.org/web/20190327233643/http://openmobilealliance.org/release/LightweightM2M/V1_0_2-20180209-A/OMA-TS-LightweightM2M-V1_0_2-20180209-A.pdf

[63] M. Veillette, P. V. der Stok, A. Pelov, and A. Bierman, “CoAP Management Interface,” Internet Engineering Task Force, Internet-Draft draft-ietf-core-comi-04, Nov. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-comi-04

[64] M. Björklund, “The YANG 1.1 Data Modeling Language,” RFC 7950, Aug. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7950.txt

[65] Open Connectivity Foundation, “OCF Core Specification Version 2.0.1,” Feb 2019. [Online]. Available: https://openconnectivity.org/specs/OCF_Core_Specification_v2.0.1.pdf

[66] oneM2M, “oneM2M: The Interoperability Enabler For The Entire M2M Ecosystem, White Paper,” Jan 2015. [Online]. Available: http://www.onem2m.org/images/files/oneM2M-whitepaper-January-2015.pdf

[67] oneM2M, “oneM2M Functional Specification Version 3.15.0,” Mar 2019. [Online]. Available: http://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=29382

[68] S. M. Kim, H. S. Choi, and W. S. Rhee, “Iot home gateway for auto-configuration and management of mqtt devices,” in *2015 IEEE Conference on Wireless Sensors (ICWiSe)*, Aug 2015, pp. 12–17.

[69] H. Huang, J. Zhu, and L. Zhang, “An sdn_based management framework for iot devices,” 2014.

[70] V. M. Tayur and R. Suchithra, “Software defined unified device management for smart environments,” *network*, vol. 121, no. 9, 2015.

[71] T. Perumal, S. K. Datta, and C. Bonnet, “Iot device management framework for smart home scenarios,” in *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*, Oct 2015, pp. 54–55.

[72] W. Jin and D. H. Kim, “Iot device management architecture based on proxy,” in *2017 6th International Conference on Computer Science and Network Technology (ICCSNT)*, Oct 2017, pp. 84–87.

[73] W. G. Chang and F. J. Lin, “Challenges of incorporating oma lwm2m gateway in m2m standard architecture,” in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct 2016, pp. 1–6.

[74] S. K. Datta and C. Bonnet, “A lightweight framework for efficient m2m device management in onem2m architecture,” in *2015 International Conference on Recent Advances in Internet of Things (RIoT)*, April 2015, pp. 1–6.

[75] Traficom, “Suositus IPv6:n käyttöönotosta kuluttajalaajakaistaliittymissä (in Finnish),” [Online; accessed 02-April-2019]. [Online]. Available: http://web.archive.org/web/20190402095056/https://www.traficom.fi/sites/default/files/media/regulation/200-2014-S-Suositus-IPv6-n-kayttoonotosta-kuluttajaliittymissa.pdf

[76] T. Savolainen, J. Korhonen, K. Iisakkila, B. Patil, J. Soininen, and G. Bajko, "IPv6 in 3rd Generation Partnership Project (3GPP) Evolved Packet System (EPS)," RFC 6459, Jan. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6459.txt

[77] C. Byrne, D. Drown, and V. Ales, "Extending an IPv6 /64 Prefix from a Third Generation Partnership Project (3GPP) Mobile Interface to a LAN Link," RFC 7278, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7278.txt

[78] W. A. Simpson, D. T. Narten, E. Nordmark, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861, Sep. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4861.txt

[79] D. T. Narten, T. Jinmei, and D. S. Thomson, "IPv6 Stateless Address Autoconfiguration," RFC 4862, Sep. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4862.txt

[80] A. Cooper, F. Gont, and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms," RFC 7721, Mar. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7721.txt

[81] B. E. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds," RFC 3056, Feb. 2001. [Online]. Available: https://rfc-editor.org/rfc/rfc3056.txt

[82] C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," RFC 4380, Feb. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4380.txt

[83] P. Matthews, I. van Beijnum, and M. Bagnulo, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers," RFC 6146, Apr. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6146.txt

[84] S. Steffann, I. van Beijnum, and R. van Rein, "A Comparison of IPv6-over-IPv4 Tunnel Mechanisms," RFC 7059, Nov. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc7059.txt

[85] P. S. Kim, "Analysis and comparison of tunneling based ipv6 transition mechanisms," *International Journal of Applied Engineering Research*, vol. 12, no. 6, pp. 894–897, 2017.

[86] M. TOMOHIKO, I. YOSHIHIRO, and K. RYO, "Effect of communication quality degradation on web usability over 6to4 networks," *Electronics and Communications in Japan*, vol. 100, no. 4, pp. 3–14. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ecj.11941

[87] N. Bahaman, E. Hamid, and A. S. Prabuwono, "Network performance evaluation of 6to4 tunneling," in *2012 International Conference on Innovation Management and Technology Research*, May 2012, pp. 263–268.

[88] B. Silverajan, S. Kinnari, A. Vekkeli, and T. Vartiainen, "Beyond connectivity," *IEEE Vehicular Technology Magazine*, vol. 4, no. 3, pp. 55–61, Sept 2009.

[89] B. E. Carpenter, "Advisory Guidelines for 6to4 Deployment," RFC 6343, Aug. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6343.txt

[90] M. Townsley and O. Trøan, “IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) – Protocol Specification,” RFC 5969, Aug. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5969.txt

[91] O. Trøan and B. E. Carpenter, “Deprecating the Anycast Prefix for 6to4 Relay Routers,” RFC 7526, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7526.txt

[92] Bluetooth Special Interest Group, “GATT REST API White Paper,” Apr 2014. [Online]. Available: http://web.archive.org/web/20190403084316/https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=285910&_ga=2.245581764.1383925978.1554280404-1893575694.1554280404

[93] T. Savolainen, “Optimal Transmission Window Option for ICMPv6 Router Advertisement,” Internet Engineering Task Force, Internet-Draft draft-savolainen-6lo-optimal-transmission-window-00, Jan. 2014, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-savolainen-6lo-optimal-transmission-window-00

[94] N. Mathewson, “Fast portable non-blocking network programming with libevent,” 2012.

[95] H. Tschofenig, J. Arkko, D. Thaler, and D. McPherson, “Architectural considerations in smart object networking,” Internet Requests for Comments, RFC Editor, RFC 7452, March 2015.

[96] IBM, “Watson IoT Platform Overview,” [Online; accessed 24-July-2018]. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSQP8H/iot/overview/overview.html

[97] R. Morabito, R. Petrolo, V. Loscri, and N. Mitton, “Legiot: A lightweight edge gateway for the internet of things,” *Future Generation Computer Systems*, vol. 81, pp. 1–15, 2018.

[98] Cisco Systems, “Hot Standby Router Protocol Features and Functionality,” [Online; accessed 23-February-2019]. [Online]. Available: http://web.archive.org/web/20190617060257/https://www.cisco.com/c/en/us/support/docs/ip/hot-standby-router-protocol-hsrp/9234-hsrpguidetoc.html

[99] S. Nadas, “Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6,” RFC 5798, Mar. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5798.txt

[100] B. Silverajan, H. Zhao, and A. Kamath, “A semantic meta-model repository for lightweight m2m,” in *2018 IEEE International Conference on Communication Systems (ICCS)*. IEEE, 2018, pp. 468–472.

[101] Thread Group, “Thread Overview,” [Accessed 10-September-2019]. [Online]. Available: https://www.threadgroup.org/Portals/0/documents/support/ThreadOverview_633_2.pdf

Publications

Publication I

IPv6 Addressing Strategies for IoT

T. Savolainen, J. Soininen and B. Silverajan

IEEE Sensors Journal. vol 13, no 10, pp. 3511-3519, 2013, IEEE.

DOI: 10.1109/JSEN.2013.225969

IPv6 Addressing Strategies for IoT

Teemu Savolainen, Jonne Soininen, Bilhanan Silverajan

***Abstract*—In this paper we analyze the suitability of different IPv6 addressing strategies for nodes, gateways and various access network deployment scenarios in the Internet of Things. The vast numbers of things being connected to the Internet need IPv6 addresses, as the IPv4 address space was effectively already consumed prior to the introduction of the Internet of Things. We highlight how the heterogeneity of nodes and network technologies, extreme constraint and miniaturisation, renumbering and multihoming, present serious challenges towards IPv6 address allocation. By considering the topologies of various types of IoT networks, their intended uses as well as the types of IPv6 addresses that need to be deployed, we draw attention to allocation solutions as well as potential pitfalls.**

IPv6, IoT, addressing, renumbering, low-power, topology.

I. Introduction

In 2011, nearly 1 billion smart connected devices, comprising of PCs, tablets and smartphones were shipped, with estimates indicating that the number would almost double by 2016 [1]. The total number of Internet connected devices, however, exceeded 8.7 billion in 2012 [2]. Within 7 years, expectations are rife that the present number of connected nodes would be significantly dwarfed, as estimates from standardization bodies, network equipment vendors as well as network operators range from 25 to 50 billion connected devices [2][3][4]. These nodes comprise both smart devices as well as complexity and resource limited nodes such as sensors and actuators. While commonly available connectivity technologies such as cellular, fixed Ethernet and Wi-Fi networks would continue to be used, billions of resource constrained nodes are expected to also utilise low power communication over technologies such as Bluetooth Low Energy (BLE), DASH7, Insteon, 1-Wire, as well as IEEE 802.15.4-based technologies such as ZigBee.

The phrase "The Internet of Things" (IoT) was the title of the seventh report in a series of Internet reports the International Telecommunications Union ITU-T issued to address challenges to the network[5]. This report was published in 2005 and envisioned interconnected Internet-enabled networks providing ubiquitous connectivity which had far reaching implications for machine to machine communications, delivering content to users as well as allowing everyday household objects to be connected to the digital world. Advances in Radio Frequency Identification (RFID), nanotechnology, sensors and smart technologies (such as wearable computing, intelligent homes and vehicles and robotics) were identified as technology enablers for IoT. While the origins of the term "IoT" predate the ITU-T report, the vision became a reality in 2010 when Internet connected devices began outnumbering the world's human population [2].

One of the first steps in allowing such large numbers of nodes to co-exist and communicate in the Internet is the existence of efficient, scalable and federated architectures and schemes for unambiguous naming and addressing. This is particularly so in IoT, where there is an abundance of sleeping nodes, intermittent connectivity, mobility and non-IP devices. In addition to reachability, this provides the ability for unique identification, facilitation of active mechanisms for service discovery as well as passive lookups.

We examine how IPv6 can be effectively deployed for various IoT topologies particularly with emphasis on address allocation approaches. The motivation and contributions of this paper are outlined in the next section. The terminologies used to describe various concepts, topologies and solutions are supplied in Section 3. Section 4 describes various relevant facets of IPv6 for IoT node addressing while Section 5 outlines and explains possible network topologies for realistic deployment as well as upstream connectivity. Mobility implications are then discussed in Section 6 while solutions for address allocation are presented in Section 7. We then arrive at a conclusion of how well current and future addressing needs for IoT devices are met by efforts towards IPv6 adoption for the Internet of Things.

II. Motivation

IPv6 address allocation schemes for constrained nodes have been covered extensively in academic literature, particularly by the wireless sensor networks (WSN) research community. Some of the issues surrounding address allocation in WSN are similar to IoT. The challenges of combining sensor networks with IP access are discussed in [6], particularly with low power networks such as 6LoWPAN. A scheme called MPIPA is presented in [7] that uses three-dimensional location coordinates to assign unique IPv6 addresses to sensor nodes in a smart grid. A lightweight stateless IPv6 address autoconfiguration for 6LoWPAN using color coordinators is discussed in [8]. An IPv6 address allocation scheme applicable to MANETs is outlined in [9] that allows the acquisition of unique IPv6 addresses from neighbouring proxy nodes by mobile nodes. Three ways in which WSNs can be integrated with the Internet are introduced in [10] along with a discussion of the complexities and routing requirements of each approach.

Many of these contributions view IPv6 as a replacement to IPv4 owing to its vast globally unique address space and the simplicity of network configuration, therefore focusing on parameters for optimal IPv6 address allocations to reduce the chances of address collisions during Duplicate Address

Teemu Savolainen (teemu.savolainen@nokia.com) is with the Nokia Research Center.
Jonne Soininen (jonne.soininen@renesasmobile.com) is with the Renesas Mobile.
Bilhanan Silverajan (bilhanan.silverajan@tut.fi) is with the Tampere University of Technology.

Detection (DAD). We do not intend to revisit old ground, but underline how the topology of IoT networks and the types of IoT nodes impose new requirements and limitations on node addressing, address configuration, reachability and naming needs. Consequently this paper's aim is to look at how well current efforts in IPv6 research and standardisation cope with technical challenges in the IoT for various scenarios, bearing in mind the expected properties of IoT nodes and topologies.

The motivation of this paper is to highlight IPv6 addressing schemes for nodes and gateways in various topologies with the emphasis on issues such as:

Node Miniaturisation: Extremely severely constrained nodes may not even have the ability to dynamically configure their addresses. There will be nodes which broadcast or communicate only unidirectionally and extremely miniaturized nodes such as nano machines communicating with the Internet (the Internet of NanoThings [11]) need to be considered for the future.

Renumbering Challenges: A smart vehicle and its associated networked sensors and nodes need to be renumbered upon entering specific locations (such as during car servicing and connecting to a service network), or in the absence of Internet connectivity without adverse disruptions to the operation of the intravehicular network, or communication with external nodes

Multihoming Challenges: Smart homes in which multiple stakeholders, which includes home owners as well as 3rd party operators (utility meters, smart grids and device vendors) may wish to access readings and data without being dependent on a single Internet uplink from the home. An electricity meter in the home may be simultaneously accessible from within the firewalled home network by the home owner and from the electricity provider's own Internet uplink and downlink. Multihoming scenarios also consider mobility scenarios with multiple interfaces where vertical handovers occur or when nodes migrate back and forth between IPv6 and non-IPv6 networks.

Proxying and Tracking non-IP nodes: The ability for an IoT node to serve as a bridge or a proxy towards non-IP technologies expected to be prevalent. BLE nodes and objects tracked with RFID provide unique IDs or tag values that can be used to generate IPv6 addresses. However, BLE IDs can be regenerated by the node from time to time, while the points of association for RFID tags change with respect to their readers or writers. Considerations that IPv6 addresses may not be permanent for such node types must be taken into account.

III. IoT Terminology

The architectural elements that form the building blocks for IoT can, and have been classified in many ways. The authors of [12] presented a high level taxonomy of IoT to illustrate the ubiquity of the architecture. They defined three IoT components necessary for seamless ubiquitous computing: *Hardware* such as sensors, actuators and embedded communication systems, *Middleware* such as analytic frameworks for data computation and on-demand storage and *Presentation* which provides a visual perspective with the aid of interpretive, multi-platform tools. Node classifications have also been performed in various ways. The IEEE 802.15.4 low-rate wireless personal area network (LR-WPAN) standard classifies any participating, networked node as either a full-function device (FFD) which can participate in any topology, implement the entire protocol set and act as a PAN co-ordinator, or it can be a reduced-function device (RFD) for more limited nodes that have minimal implementations allowing them to paticipate as leaf nodes in various topologies without the ability to perform any co-ordination activities [13]. In [14] authors discuss nodes constrained by power and memory capacity and classifies them as belonging to *Class 0* for very constrained and simple sensor-style devices, *Class 1* for nodes containing approximately 10 Kilobytes of RAM memory and 100 Kilobytes of Flash memory or *Class 2* for nodes containing approximately 50 Kilobytes of RAM memory and 250 Kilobytes of Flash memory. A different kind of taxonomy is presented when discussing ambient energy harvesting sensors: IoT installations are classified as *Trivial*, *Classic*, and *True IoT* based on the number of nodes and communication technologies, to understand their energy needs [15].

To understand the presented concepts and topologies in this paper, we provide definitions of the various parts of an IoT network and the types of node present in the IoT.

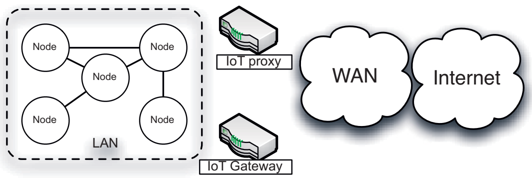


Figure 1: Internet of Things, generalised architecture

Figure 1 shows a high-level layout of the Internet of Things. On the surface, the architecture does not differ from any normal Internet network structure. The explanation of the different building blocks are in the following.

IoT Local Area Network (LAN) is the network connecting IoT nodes in a local - relatively short range - configuration. Different network technologies can be used for connecting the IoT LAN, both wireless, and wireline, and be present in one IoT LAN. However, the technology used is relatively short range, and the administration is under one entity (person, or organization). IoT LAN topologies can differ from low-power, short-range configurations, to building or organization wide configurations. IoT LAN may or might not be connected to the Internet.

IoT Wide Area Network (WAN) is a network that covers geographically, and organizationally a wide area. While IoT LANs may be connected directly to the Internet, the IoT can be perceived also as a network of networks, with LANs connected and aggregated via WANs which are subsequently connected to the Internet. Hence, a WAN is usually a combination of network segments administered by different players reaching potentially a very large geographical area.

IoT node is a node in the IoT LAN, and through that connected to the other nodes in the IoT LAN. If the IoT LAN is connected to the Internet, the IoT node may also be connected to the Internet directly. However, this is not always the case. In addition, there are cases where the IoT node is connected to the Internet though a WAN connection without an IoT LAN being present.

IoT gateway is a router connecting an IoT LAN with a WAN, and to the Internet. An IoT gateway is a layer 3 device, which forwards IP packets between the IoT LAN and WAN implementing both the network technologies. An IoT gateway can potentially implement several LAN and WAN technologies (wireless or wireline) depending on the configuration. The IoT gateway forwards packets between the IoT LAN and the WAN on the IP layer without performing application layer tasks. In one IoT LAN, there may be zero or more IoT gateways depending on the scenario.

IoT proxy is an entity that performs an active application layer function between IoT nodes, and other entities. The application layer functionality can range from relatively simple application protocol conversion to more active application functions. The IoT proxy can be collocated with the IoT gateway.

IV. Background to IPv6

Recent development of IoT technologies, including the transport of IP over low-power radio technologies, have concentrated almost solely on IPv6. For example, only IPv6 support for 802.15.4 has been defined. An IPv6 address is a 128-bit fixed length numerical address consisting of a *subnet prefix* and an *Interface Identifier* (IID) portions. The length of the prefix and the IID can vary based on link type, but typically a 64-bit prefix, and hence also 64-bit IID, is used for address configuration. The split at 64-bit boundary has become the industry standard in the most common IPv6 implementations. Thus, most if not all of the current implementations have been designed with the assumption of 64-bit IID length.

The bits for the IID can be selected in several different ways depending on the use-case and deployment scenario, as described below. Additionally, the relatively large size of 64-bit IID has fostered innovations where meaningful information is encoded within IIDs.

- **Modified EUI-64-based IIDs**: A globally unique IID generated from network interface's globally unique identifier, such as IEEE 802 48-bit MAC address.
- **Privacy Addresses**: An IID generated using pseudo-random algorithms, if a globally unique identifier is not available, or if a host wishes to improve privacy by making IID-based tracking impossible.
- **Cryptographically Generated Addresses**: For securing IPv6 Neighbor Discovery procedures, IIDs may be derived from public keys and signed using private keys. These are seldom used.

A. IPv6 Addressing Architecture

The IPv6 address architecture defines two *scopes* for unicast addresses: link-local and global. *Link-local addresses* are used for auto-discovery and auto-configuration, and at least one is always configured for each interface of a node. IPv6 packets using link-local addresses will not be forwarded by routers to other links, as the link-local addresses are not guaranteed to be unique over a larger network. The global scope addresses, on the other hand, are expected to be globally unique and can be used in the scope of the whole Internet. A node needs a global IP address to be able to communicate over the Internet.

Unique Local Addresses (ULA) [18] are designed to be used in local networks larger than a single link, but not for communications through the Internet. However, ULA are designed to provide adequate uniqueness in order to have extremely small risk of address collision. These addresses are intended to allow routing over a network that expands over multiple links and routing hops, and even can expand over multiple networks. The address independence from the Internet's global routing system, and address administration, is a desired characteristic in some deployments. ULA may provide address stability and independence from an outside provider such as the operator, but come with the cost of limiting the communications' scope.

Globally Unique Addresses (GUA) are globally administratively guaranteed to be unique and routable in the Internet. The administration is done by the Internet Assigned Number Authority (IANA), which administers the global pool of addresses, and by the Regional Internet Registries (RIRs) who administer address space received from IANA regionally. The RIRs provide address space to the Local Internet Registries (LIRs) - operators, companies, and other organizations, who require address space for themselves, and possibly for further allocation to their customers.

B. Host Address Configuration

The IPv6 protocol suite defines a set of well-known mechanisms for address autoconfiguration on an attached link. These are Stateless Address Autoconfiguration (SLAAC) and Stateful Address Autoconfiguration, the latter being nowadays synonymous with the Dynamic Host Configuration Protocol version 6 (DHCPv6). In addition, in the case of Virtual Private Networks (VPN), Internet Key Exchange version 2 (IKEv2) can be used for address configuration.

SLAAC has been designed to provide simplest possible, yet dynamic, way for nodes to configure IPv6 addresses for themselves. With SLAAC, hosts must configure *link-local addresses* for all interfaces they use IPv6 on. The link-local address can be configured even in absence of routers. The routers on networks transmit ICMPv6 Router Advertisement (RA) messages that may include IPv6 prefixes, which nodes can use to configure one or more ULA or GUA addresses. Once a node receives RA, it will parse it and select one or more 64-bit IPv6 prefixes for combination with selected 64-bit IIDs in order to create one or more 128-bit IPv6 addresses. SLAAC is the most scalable of the mechanisms, as it does not require the network to know which nodes exist and which addresses they have configured.

DHCPv6 can be used to explicitly configure IPv6 addresses to nodes, thereby providing network administrators with added control over the nodes on their networks. Hence DHCPv6 is popular in environments where stricter control is required, such as in enterprise networks. In addition, DHCPv6 can be used for prefix delegation[16]. In prefix delegation, a router is given the responsibility over a shorter prefix from which it can advertise longer prefixes to the network segments under its responsibility. DHCPv6 requires the DHCP server to keep state on the allocated addresses. Hence, it provides more control on the addresses, but less scalability than SLAAC.

Obviously IPv6 protocol suite supports manual configuration of addresses, and this can include provisioning of addresses with mechanisms other than SLAAC or DHCPv6, including proprietary out-of-band tools. Provisioning or manual configuration is the least scalable of these approaches.

C. Remote Address Anchor Points

IPv6 hosts will always configure addresses from the point of network attachment, but additionally hosts may have addresses configured from remote anchor points. These addresses belong topologically to locations other than the hosts' direct points of network attachment. In order for the hosts to be able to use these addresses, tunneling of sorts is is required. These tunneling solutions include client-based Mobile IPv6 (MIPv6), Network Mobility (NEMO)[17], or Dual-Stack Mobile IPv6 (DS-MIPv6)[20], but also gateway-based solutions exist, such as Proxy Mobile IPv6 (PMIPv6)[19].

D. Network Prefix Translation

If hosts do not require direct visibility for global addresses, it might be feasible to number a network with ULA and utilize experimental IPv6-to-IPv6 Network Prefix Translation (NPTv6)[21] at the gateway. NPTv6 differs from traditional port and address translating NAT in that it translates in checksum neutral way and only the prefix part and not the transport layer protocol port number. If reachability from Internet to nodes numbered in such a setup is needed, the nodes need to register their public IPv6 address, the address on the Internet side of the gateway, to the used rendezvous system. Other forms of IPv6 address translation, namely NAT66, have also been discussed and speculated, but not adopted into use even in an experimental manner.

V. Topologies for IoT Deployments

The network topologies for which addressing needs are considered in this paper and that are explained in this section are illustrated in Figures 2 and 3. These topologies can be extended to more complex topologies with variations, as we will discuss in the end of the section.

A. Case A: Disconnected IoT Network Without a Central Node

A disconnected IoT LAN may have no Internet connectivity, but only connectivity within a link itself. The underlying medium may provide mesh, star, or shared connectivity, but nevertheless IPv6-wise nodes are in a single link without any router. Hence, there is no entity in the network providing numbering services - ULA or GUA prefixes. In this kind of topology, the main requirement for addressing is that IoT nodes must be able to automatically number themselves. As the automatic numbering has to be in very simple in some cases, the IPv6 addresses may be statically configured. To be clear, routable addresses are not needed, as the network is not connected to other networks, such as to the Internet. The absence of routers restricts the nodes to be on the same link in this scenario. Therefore, the most suitable address type is IPv6 link-local addresses.

B. Case B: Network With an IoT proxy

In the case B, the IoT LAN has been enhanced with an IoT proxy, which is able to provide addresses and possibly connectivity services for the IoT nodes in the IoT LAN. The IoT proxy can have permanent or intermittent connectivity to the external network, or in some cases without connectivity. When the IoT proxy has uplink connectivity, it proxies communication between the local IoT nodes and nodes in the external network. In this scenario, where all communications go through a proxy, the IoT LAN does not need global addressing, but can manage with link-local or ULA addresses, depending on the type of proxy.

C. Case C: Connected IoT network

This is a typical setup for providing IoT nodes with Internet connectivity. An Internet connected IoT gateway provides Internet connectivity to the IoT LAN. The IoT gateway receives a globally routable IPv6 prefix from the Internet service provider, and uses that prefix to number the nodes in the IoT LAN. The different mechanisms for obtaining the IPv6 prefixes are further described in Section VII. This scenario allows the IoT nodes in the LAN to be provided with a globally routable address. In addition, the IoT nodes, which communicate only within their own link, may use link-local addresses, and the IoT gateway may also provide ULA addresses to the IoT nodes.

D. Case D: An IoT Network with Bridging Star Topology

In some scenarios a gateway is a center of a network utilizing star topology. In such a network the same IPv6 prefix can be shared by nodes connected via point-to-point links to a gateway, and the gateway may implement a bridge. This approach is used for ongoing IETF work on IPv6 transmission over Bluetooth Low-Energy[23]. Also IoT deployment scenarios exist that emulate a bridging star topology, but having only the gateway maintain IPv6 representational states for extremely simple IoT nodes which do not implement IPv6 at all. For example, sensors connect and communicate to an IPv6 gateway with 1-wire or a similar solution without being IPv6-aware. The gateway however allocates IPv6 addresses and internally maintains a 1:1 IPv6 to a proprietary ID mapping for each node.

E. Case E: A Point-to-Point Network with an Internet Gateway

The scenario E is very similar to scenario D, except that the IoT nodes are not on a same link. Instead, they are connected to the IoT Gateway via their own links. Hence communication using link-local addresses is not possible between IoT nodes.

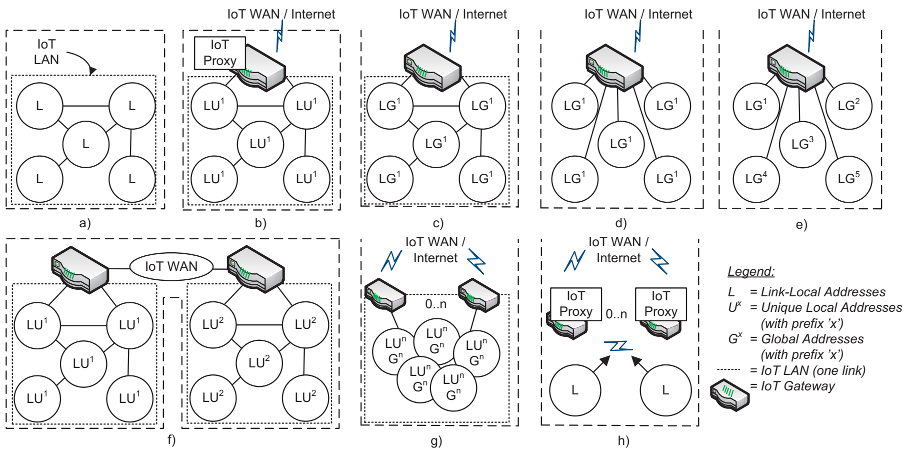


Figure 2: Set of IoT LAN Network Setups

F. *Case F: Interconnected IoT Networks*

When it comes to more advanced scenarios, two or more IoT LANs can be connected to each other via a shared link or through an IoT WAN. In this kind of case, the IoT nodes of different LANs are obviously not on a shared link with each other. Hence, the communication between them is not possible with link-local addresses. If the IoT nodes need to communicate with nodes on different network segments, used addresses have to be either ULA or GUA, depending on the network that is between the gateways. If the network is the Internet, globally routable IPv6 addresses have to be used within the network. Otherwise, the ULA will suffice. In addition, in cases where, for instance, the Internet connectivity is not always available, both address scopes can be used.

G. *Case G: Multiple Gateways*

The topologies described earlier represent the different basic topologies. There are, however, variations to these basic topologies that introduce additional addressing requirements. For instance, there can be multiple gateways connecting an IoT LAN to the Internet, or to the Internet and a private wide area network - such as a corporate network - therefore, making the IoT LAN multihomed. If the IoT nodes need to be reachable from both networks, the IoT nodes need to have addresses from the both networks. Consequently, the IoT nodes may need to have multiple addresses of global scope.

H. *Case H: Unidirectional IoT Nodes*

A significantly different approach for using IPv6 with IoT nodes is to run IPv6 over unidirectional links, or put otherwise: have send-only IoT nodes. The unidirectional approach is not fully compatible with existing IPv6 addressing solutions, as all of those assume bidirectional communications channel for Duplicate Address Detection, and for other signaling. However, in some cases it might be an attractive use case to utilize IPv6 even in such situations. For example, sensors could be reporting readings using IP multicast. One way to address nodes in such deployment is to trust link-local addresses to be unique, hard-code them, and use link-local multicast as destination address[22].

I. *Variations of the Basic Topologies*

In addition to the topology variations, the topologies can also be combined. For instance, in the same IoT LAN some nodes may be connected directly to Internet using global addresses, and some nodes may utilize ULA or link-local addresses for local communications or communicatios via a proxy. In these cases, even if the addressing requirements may vary between IoT nodes within the same network, it is important to remember that nodes which communicate between each other need to have an address of the same scope.

VI. Mobility Implications to Addressing

Movement in IP networks is related to the topological location of the network. Hence, virtually any event where the IPv6 prefix changes is considered movement. The IoT networks may either be stationary, or mobile. Thus, due to an event that causes renumbering of an IoT network, such as the loss and regain of Internet access with a prefix change, even a stationary network may move in the IP network topology. This generates challenges to the addressing of an IoT network. Figure 3 illustrates network movement and its implications to addresses. In the following, we will briefly describe mobility scenarios relevant to IoT networks.

1) The IoT network initially uses IPv6 global prefix 'A'. Due to movement or a renumbering event, the network is renumbered with global prefix 'B'. The IoT LAN is capable of renumbering, and hence resilient to movement.
2) As above, the IoT Gateway's WAN-side addressing changes. However, in the IoT LAN addressing is expected to stay stable. For instance, IoT nodes may be

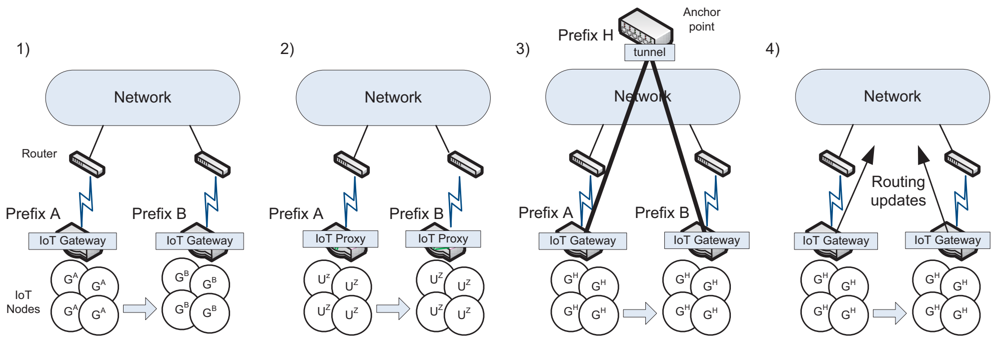


Figure 3: Mobility for IoT Networks

sleeping for extended periods and propagation of the new prefix would take long. Hence, the IoT network utilizes an addressing scheme independent from the addressing used to connect to the Internet - for instance, a ULA prefix ('U^Z' in Figure 3). The IoT gateway must isolate the IoT LAN's internal addressing from the global addressing with a mediation function such as a proxy.

3) In certain use cases it is expected that the IoT nodes' global addresses stay stable regardless of the address stability of the access network. An example is when the IoT nodes are expected to be reachable from the Internet, and their addresses are stored in some semi-permanent database such as the Domain Name System (DNS). In this case, IoT network must be numbered with a prefix topologically anchored to a different location ('H' in Figure 3) than where the gateway is actually attached. In this case, there is a tunnel to a remote anchor point. As the IoT gateway's uplink prefix changes from 'A' to 'B', no renumbering of IoT nodes takes place as the nodes' communications are using global prefix 'G^H' and always routed via the remote anchor.
4) In some deployments the gateway might have ability to make dynamic routing updates to the network, and hence the prefix used in the IoT LAN can remain functional even during movements. Essentially, the IoT LAN would move within the network topology. Applicability of this approach is limited to networks and IoT WANs, that allow such routing updates from IoT gateways, that do not leak updates to the Internet, that can handle the numbers of routing updates, and that contain IoT nodes tolerating delays caused by routing information propagation through the network.

As with the topology scenarios, the mobility scenarios can be combined in one network. An IoT LAN can be comprised of multiple different types of nodes that have different requirements for the connectivity. For instance, some of the nodes can be renumbered, some use either link-local or ULA for address stability, and there may be even nodes that require stable global addresses. This is dependent on the use case of specific nodes.

VII. Address Allocation Solutions

In the previous two sections, we discussed the different addressing requirements related to topology, mobility and resilience. In this section, we discuss solutions fulfilling the identified requirements. As the IoT gateway (possibly with proxy functionalities) is the element that provides the IoT LAN with Internet connectivity, it also has a central role in the IoT LAN address management. Thus, many of the following solutions place functional requirements on the IoT Gateway.

A. Allocating Addresses for IoT Nodes

The same mechanisms are used for address allocation of IoT nodes as other nodes in the Internet - Stateless Address Autoconfiguration (SLAAC), and Dynamic Host Configuration Protocol for IPv6 (DHCPv6). These mechanisms were described in Section IV-B in detail. Additionally, the nodes can be statically configured in different ways, an address can be configured over a management interface, or a node may use IPv6 address based on a hardcoded hardware identifier. A static IID may be helpful in scenarios where node identification based on IPv6 address is a requirement.

Considering the requirements of Section V, the most relevant mechanism for IoT node addressing is SLAAC with ULA or GUA addresses depending on the deployment scenario. A hardware-based or dynamically selected IID can be used for creating link-local address, ULA or GUA. However, link-local addresses based on hardware identifiers are practically the only choice of addressing in setups where IoT nodes do not have receive capabilities, and hence cannot perform Duplicate Address Detection procedure.

Static configuration through a management interface could be used, but the high operational maintenance cost, due to reconfiguration effort in the case of network address renumbering, makes it an impractical approach for IoT.

Hardcoding addresses, during manufacturing or by reflashing of IoT node firmware, of any other scope than link-local is not advisable. The network topology, and environment cannot be sufficiently known during manufacturing, and the network topology can significantly outlive the lifetime of an IoT node making this approach too restrictive. An IoT node with a topologically incorrect address would be unreachable, and transmissions of packets with improper addresses would likely fail due to routers' source address validation filters.

B. IoT Gateway/Proxy

IoT Gateway and IoT Proxy play central nodes in allocating addresses to the IoT nodes, because they are the routers in the IoT LAN and responsible for both the Internet connectivity and possibly connectivity between IoT LANs. In the scenarios C, D, E, and G, illustrated in Figure 2, the gateway advertises globally routable prefixes, which are used by IoT nodes to configure addresses with SLAAC. The IoT Gateway must obtain prefixes from the upstream network with mechanisms such as DHCPv6 Prefix Delegation. In addition to globally routable prefixes, and regardless of the presence of Internet connectivity, an IoT Gateway or IoT Proxy may also generate an ULA prefix, as illustrated in scenarios B, F, and G, and advertise it to the IoT LAN. As ULA addresses can be generated and maintained independently from global addresses or Internet connectivity, ULA is a good choice for IoT LAN addressing in use cases where internal address stability is important or connectivity to other IoT LANs is required.

The simplest of IoT nodes might only deliver data to the closest IoT Proxy, as illustrated in scenario H in Figure 2, which could furthermore aggregate data from multiple nodes. In these cases, having ULA or GUA in use might be either computationally exhaustive, or simply unnecessary. The simplest nodes could manage with link-local addresses, and utilize link-local multicast address as the destination address of their packets. The IoT Proxy would gather packets sent to the link-local multicast group, and proxy them forward.

When ULAs are used for providing address stability, but IoT Nodes need to communicate to the Internet, the IoT Proxy may need to implement application layer proxy, or possibly the IoT Gateway may need to support the experimental NPTv6. An application layer proxy could also perform other processing as well, in addition to just passing data between the Internet and the IoT Nodes. For example, the proxy could enrich the information sent by IoT Nodes. Use of NPTv6 cannot be recommended at this point of time, as the technology is experimental and more research is needed generally, and in particular in case of IoT, to assess its usability.

Until the Internet has fully transitioned to IPv6, the IoT gateway may only have an IPv4-only uplink. Despite significant efforts by the community to design a perfect IPv6 transition solution, no universal solution has been found despite the availability of a toolbox of various tricks. Due to the challenges in transition, forthcoming IoT deployments should be designed to have native IPv6 connectivity. Otherwise, suboptimal alternatives have to be chosen belonging to three categories: protocol translation from IPv6 to IPv4 (e.g. NAT64), tunneling IPv6 packets over the IPv4 (e.g. 6to4), and data relaying (e.g. application layer proxies).

C. Global Address Stability

As stated above, ULAs can provide address stability within the IoT LAN but with a cost. However, in certain scenarios the IoT network, be it physically mobile or not, may require globally routable addresses. IoT nodes or gateways have to implement ways to inform upstream network about IoT LAN, or node, movement. The solutions include tunneling based approaches discussed herein, or use of routing protocols discussed in Section VII-E.

Host-based mobility, such as DS-MIPv6, can be supported by computationally capable IoT nodes, but is unlikely to be supported by constrained nodes of any form. If the number of IoT nodes in an IoT LAN is large, the signalling load to the DS-MIPv6 tunneling end-point (Home Agent) - would become an issue. In addition to host mobility, DS-MIPv6 supports network mobility. Therefore, a more suitable place to terminate DS-MIPv6 is the IoT Gateway.

As explained earlier, other mobility management technologies in addition to DS-MIPv6 do exist for IPv6 including MIPv6, NEMO and PMIPv6. However, these technologies have limitations that make them inferior in the IoT context. MIPv6, and NEMO are constrained to work only on IPv6 which is a serious restriction in the beginning of IoT deployments, as currently, many networks are still IPv4 only capable. DS-MIPv6 supports tunneling to the Home Agent (HA) even over IPv4, and it works also in IPv6 only environments. Thus, it is both a good solution now when IPv6 is not ubiquitously available, and it is future proof. PMIPv6, on the other hand, is made for network based mobility management. It means that the access network has to provide the mobility management. Therefore, it is only available as a solution as an operator provided service.

D. Upstream Bridging

In some cases, address delegation mechanisms such as DHCPv6 Prefix Delegation may not be available. The IoT Gateway may be provided a single /64 prefix, from which it obviously cannot delegate prefixes to the IoT LAN. This scenario is present today, for instance, in 3GPP networks where a mobile device is allocated a single /64 prefix. In this case, the IoT Gateway has to somehow "bridge" between the upstream provider and the IoT LAN. This problem is topical in the IETF, as all proposals for this problem, this far, have all had some issues.

E. Routing

In cases where the IoT LAN is considerably big and independently operated, dynamic address allocation via DHCPv6 may not provide enough address stability and be sufficiently scalable. As with any relatively big networks, the interface between the IoT LAN, and its upstream network or networks can be done with routing. When the IoT LAN is served by one operator, and the IoT LAN has its address space from that operator, interior routing protocols such as OSPF and IS-IS may be feasible. In topologies where there are multiple segments and routers within the IoT WAN, IoT LANs may also use routing protocols for location updates. The IETF is working on such scenarios in the Homenet Working Group, which has been concentrating on the use of OSPF protocol.

F. Multihoming

Both networks, and nodes can be multihomed. We described use cases where multihoming may be needed in Section II. As small networks, such as home IoT LANs, cannot afford to have their own address space, a multihomed IoT LAN would have different prefixes from different Internet connections. Different approaches to this problem exist. The two main approaches are described in the following.

Proxy based approach: Either one proxy manages multiple upstream connections with their relative IP addresses, or there are separate IoT Proxies per upstream connection. In addition to the different proxies, an IoT Gateway providing a prefix from an upstream link can be provided.

Separate IoT Gateways: Multiple IoT Gateways provide their own address space to the IoT LAN, and the IoT Nodes are multihomed.

The first scenario above is relatively straightforward as the nodes would not be aware of multihoming. However, there may be cases where that approach is too restrictive. The communication between the node, and the upstream network is isolated by the proxy, which may not be desirable.

The challenge of multihoming is the multihomed node has to know, which source address to use in communication depending on the destination, and the IoT LAN originated packets have to be routed over the correct IoT Gateway. For upstream IoT Gateway selection, routing protocols can be used between the Gateways. However, the IPv6 source address selection rules may provide suboptimal results if the destination address and source address are not derived from the same prefix. Otherwise, the multihomed node has to use policy information to select the correct source address. However, currently no defined solution exists to convey such information. The IETF has worked on multihoming solutions, such as in the Multiple Interfaces (MIF) Working Group, but currently there is no consensus on the solution.

VIII. Conclusion

Device heterogeneity, IP networks interoperating with non-IP networks, and the amount of connected, unique entities are defining features of the IoT. The configuration schemes used to number IoT nodes are largely similar to current practices for numbering standard nodes. The bigger question is more often what address scopes to use and when, as we've demonstrated herein. However, the more constrained the IoT device, the more challenges emerge and areas for improvement appear. For constrained IoT nodes, mobility, multihoming, and generally renumbering events are resource consuming due to an increased need to monitor movement and refresh addresses. Mobility with help of remote anchor points requires more infrastructure, routing-based mobility poses scalability issues, and if these mobility events are hidden with the use of ULA, IoT gateways are required to implement non-transparent mediation functions. Similar requirements also arise from IoT nodes that operate in unidirectional mode, and are unable to configure global scoped addresses for themselves. These mediation functions can, for example, take the shape of application layer proxies or perhaps even perform translation via NPTv6. It is also entirely possible we may encounter situations where even the most constrained nodes can use IPv6, but are not addressable with, or may completely disregard obtaining IPv6 addresses. This may even be desirable in certain scenarios. If not, however, it might be wise to define lightweight, IPv6-friendly, and generic mediation functions. We anticipate further research by both the academy and the industry for discovering how, and if even the most constrained nodes could somehow be efficiently addressed with global scoped IPv6 addresses.

IX. Acknowledgements

This work was supported by authors' employers and in part by TIVIT's Internet of Things research program, Finland.

References

[1] International Data Corporation (IDC), "Nearly 1 Billion Smart Connected Devices Shipped in 2011 with Shipments Expected to Double by 2016", IDC Press Release, http://www.idc.com/getdoc.jsp?containerId=prUS23398412, 2012.
[2] Cisco Systems, "The Internet of Things How the Next Evolution of the Internet Is Changing Everything", White Paper, http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, 2011.
[3] ITU Broadband Commission, "The State of Broadband 2012: Achieving Digital Inclusion for All", ITU Broadband Commission Report, http://www.broadbandcommission.org/Documents/bb-annualreport2012.pdf, 2012.
[4] Ericsson, "More than 50 Billion Connected Devices", White Paper, http://www.ericsson.com/res/docs/whitepapers/wp50billions.pdf, 2011.
[5] ITU, "The Internet of Things", International Telecommunication Union Internet Reports, 2005.
[6] P. Neves and J. Rodrigues, "Internet Protocol over Wireless Sensor Networks, from Myth to Reality", Journal of Communications, Vol 5, No 3 (2010), 189-196, 2010.
[7] C. Cheng, C. Chuang and R. Chang, "Three-Dimensional Location-Based IPv6 Addressing for Wireless Sensor Networks in Smart Grid," Proc. IEEE 26th International Conference on Advanced Information Networking and Applications (AINA), pp. 824-831 2012.
[8] S. Hyojeong, E. Talipov and C. Hojung, "IPv6 lightweight stateless address autoconfiguration for 6LoWPAN using color coordinators," Proc. IEEE International Conference on Pervasive Computing and Communications(PerCom) 2009, pp. 1-9 2009.
[9] X. Wang and Y. Mu, "A secure IPv6 address configuration scheme for a MANET", Security and Communcations Networks, 2012 :Wiley
[10] K. Zhang, D. Han and H. Feng, "Research on the complexity in Internet of Things," Proc. 2010 International Conference on Advanced Intelligence and Awareness Internet (AIAI 2010), pp.395-398, 2010.
[11] I.F. Akyildiz and J.M Jornet, "The Internet of nano-things," IEEE Wireless Communications, vol.17, no.6, pp.58-63, 2010.
[12] G. Jayavardhana, R. Buyya, S. Marusic and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions." arXiv preprint arXiv:1207.0203 2012.
[13] IEEE-TG15.4, "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," IEEE standard for Information Technology, 2003.
[14] C. Bormann, M. Ersue and A. Keranen "Terminology for Constrained Node Networks", IETF Internet Draft draft-ietf-lwig-terminology-03 (work in progress), 2013.
[15] M. Lamppi, "Ambient Energy Harvesting", Proc. Aalto University T-106.5840 Seminar on embedded systems, https://wiki.aalto.fi/display/esgsem/2011S-iot, 2011.
[16] O. Troan and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", IETF RFC 3633, 2003.
[17] V. Deverapalli, R. Wikikawa, A. Petresku and P. Thubert, "Network Mobility (NEMO) Basic Support Protocol", IETF RFC3963, 2005.
[18] R. Hinden and B. Haberman, "Unique Local IPv6 Unicast Addresses", IETF, RFC 4193, 2005.
[19] S. Gundavelli (Ed.), "Proxy Mobile IPv6", IETF RFC 5213, 2008.

[20] H. Soliman H (Ed.), "Mobile IPv6 Support for Dual Stack Hosts and Routers", IETF RFC 5555, 2009.
[21] M. Wasserman and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", IETF RFC 6296, 2011.
[22] J. Arkko, H. Rissanen, S. Loreto, Z. Turanyi and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work-in-progress), 2011.
[23] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby and C. Gomez, "Transmission of IPv6 Packets over BLUETOOTH Low Energy", draft-ietf-6lowpan-btle-12 (work-in-progress), 2013.

BIOGRAPHIES

`Teemu Savolainen` received his M.Sc of Information Technology in 2011 from Tampere University of Technology, Finland. He is Principal Researcher and inventor at Nokia and has worked on wireless networking with emphasis on IPv6 enabled mobile handsets since 1999. Teemu has been actively contributing to the IPv6 related standards at IETF, 3GPP, and lately at Bluetooth SIG for standardization of IPv6 over Bluetooth Low-Energy. He is co-authoring a book with Jonne Soininen about "Deploying IPv6 in 3GPP Networks".

`Jonne Soininen`, M.Sc, is Head of Standardization Strategy at Renesas Mobile and IETF Liaison to ICANN Board. He has been working on IPv6 support for IETF and 3GPP standards and networks since 90s. Jonne has been active in 3GPP, 3G.IP, GSMA, ICANN, IETF, IGF, ISOC, ITU-T, OMA, and the WIMAX Forum contributing to the evolution of IPv6 and mobile cellular networks both from technical and regulatory angle. During this time Jonne Soininen has served as the chairman of the IETF V6OPS and NETLMM working groups, he has been a member and the chairman of the IETF Administrative Oversight Committee, the ETSI Liaison to the ICANN board, and the vice-chair of the ISOC Advisory Council.

`Bilhanan Silverajan` received his M.Sc in Engineering from Lappeenranta University of Technology, Finland in 1998 and his B.Sc in Computer Engineering from Nanyang Technological University, Singapore in 1993. He is currently pursuing his PhD in Tampere University of Technology. His research interests include pervasive networking, service discovery, communications middleware and constrained protocols for the Internet of Things.

Publication II

IPv6 Experiments in Deploying and Accessing Services from Home Networks

B. Silverajan, K. Huhtanen and. J, Harju

2006 IEEE Asia-Pacific Conference on Communications (APCC '06), pp. 1-5, 2006, IEEE.

DOI: 10.1109/APCC.2006.255794

IPv6 Experiments in Deploying and Accessing Services from Home Networks

Bilhanan Silverajan, Karri Huhtanen, and Jarmo Harju
Institute of Communications Engineering
Tampere University of Technology
P.O. Box 553, 33101 Tampere Finland
Firstname.Lastname@tut.fi

Abstract

New networking challenges are arising in home networks which will host advanced services in the future. In this paper, we look at home networks as they are managed today, and attempt to identify difficulties that will arise in future when advanced usage is desired. We discuss how IPv6 and standardised IPv6 transition technologies assist in deploying advanced services from home networks. Our practical experiences and experiments with bringing IPv6 to home networks using 6to4, building an experimental prototype home gateway device, as well as deploying image sharing, data and streaming audio services from these networks are detailed.

1. Introduction

In recent years, we have seen a dramatic increase in homes having broadband access to the Internet. Together with cheaper, more powerful and highly portable computers available today, computing at home is evolving from using a PC as a single shared commodity for the entire family, to each individual having his or her own laptop or PDA. Many of the machines today have integrated networking abilities, which has given rise to the desire to network them at home for shared and simultaneous access to the Internet.

With the advent of inexpensive 802.11-based routers, wired networking has given way to wireless environments at home. Consequently, setting up a home network in many homes just involves connecting personal devices and computers to a wireless router integrated with an ADSL or broadband cable modem. Upon startup, this device automatically contacts an ISP upstream to obtain an IPv4 address which is usually dynamically assigned.At the same time, the device serves as an Application Layer Gateway (ALG), a firewall/packet filter as well as a Network Address Translator (NAT). The use of Network Address Translation lets users conserve IPv4 addresses, and the use of firewalls prevents malicious access to home devices. The entire process on the whole is quite transparent to the user.

However, as home networks gain popularity, we envisage networked devices going beyond traditional uses of the Internet at home. Services arising out of social needs as well as digital media convergence will proliferate and eventually dominate. There will be an increased desire to host various kinds of services from the home network itself, making them accessible either to other devices in the home network, to private social communities, or to the Internet at large. At the same time, such networked devices will begin interacting wirelessly with other home consumer entertainment devices, set-top boxes and home security and surveillance systems.

The premise of this paper resides in addressing this fundamental shift in the way home networks of the future will function. In addition to consuming services coming into the home from the public Internet, these networks will just as easily need to export various services out into the Internet too. We therefore look at home networks as they are managed today, and attempt to identify challenges that will arise in future when such a change happens. We also discuss how next generation network technologies like IPv6 and standardised IPv6 transition technologies assist in alleviating shortcomings in IPv4 and deploying advanced services from home networks. Benefits and drawbacks of using this approach would be shown regarding ease of deployments and configuration, transparency in accommodating current and future services as well as minimising any disruption to existing IPv4 networks and services.

Section 2 looks at home networks in their current form, and some of the current limitations in IPv4 with them. Sections 3 and 4 discuss utilising IPv6 and IPv6 transition technologies in conjunction with home networks. Section 5 elaborates on our studies in bringing IPv6 to the home: our experimental home gateway device and our experiments with deploying some example IPv6 services related to image sharing, data transfer as well as audio streaming. In Section 6 we draw observations based on our work while Section 7 contains some concluding remarks.

2. Home networks: current issues

The kinds of devices found in a residence are usually desktops, laptops, PDAs, smartphones, gaming consoles, printers, cameras, and media centers. With these, the services and service types often desired or present include:

1. Interactions like audio/video chat, LAN Gaming
2. Calendar, task synchronisation with family members
3. Content storing, retrieval for movies, songs, photos
4. Accessing blogs, webcams and other types of multimedia content over the web
5. Remote access to the home network from external networks to documents, various files, printers.

Although service types can be identified into one of the above-mentioned broad categories, it is almost impossible to anticipate every specific application type, its behaviour and network or service usage. Since many services involve traversal through the ALG, firewall and NAT, this threatens to pose significant scalability and configuration trouble for a typical home network owner. This consequently impedes the deployment and adoption time of some of these services. Sometimes, it is even impossible for multimedia and interactive services to work behind a NAT. Consequently, some gaming and P2P applications had begun incorporating kludges in their protocols to circumvent ALG configuration issues, the most common of which is to tunnel the protocol through well-known ports such as port 80 or 25.

Another commonly used technique involves the use of Simple Traversal of UDP through NAT (STUN) [1] to discover and successfully penetrate different kinds of NATS. However, applications using STUN need to be redesigned and recompiled to become STUN-aware, as STUN usage is not transparent to the application.

To counter the fact that services such as those relying on P2P protocols, LAN-based gaming, video conferencing and messaging might break behind a NAT-based system, ALGs contain embedded intelligence and control interfaces. This allows a home network owner to configure the gateway at the application layer to allow some protocols to traverse the NAT. Again, this becomes a cumbersome approach, compounded with the fact that ALGs cannot transparently cope with multiple instances of the same service being hosted from the home network. For example, if two or three web servers reside in a home network, only one would be reachable at the default port, with the others needing to be mapped to non-standard ports on the NAT.

Therefore using NATs often continue to pose a hurdle in deploying many of the services mentioned. To overcome this, some homeowners have resorted to obtaining and using multiple public IP addresses for use in home networks from their ISPs. This goes a little way in lifting the “NAT tax”. However, unless these address are allocated from the same subnet block, there is no guarantee the addresses allocated by the ISP will reside in the same subnet; an ISP may own several non-contiguous network blocks from which addresses are dynamically allocated. This would result in inefficient bandwidth usage for the home network, as packet routing between 2 hosts in a home would begin to involve the access router of the ISP.

For example, assume a home network setup has an image server machine as well as a desktop machine and obtains 2 different IP addresses for the 2 nodes from its ISP, allocated from different subnets. If the desktop were to try and reach the image server, instead of the traffic being localised within the home, packets would be sent first to the router of the ISP before being sent back into the home. Although this does not present a serious problem in this scenario, the quality of service will significantly deteriorate if variable bit rate traffic such as video streaming becomes involved. Not only would the uplink and downlink be in danger of congestion, it would introduce unnecessary packet processing loads for the ISP access router.

3. Bringing IPv6 to home networks

Introducing IPv6 into home networks brings about the following advantages:

1. In addition to DHCPv6 configuration, IPv6 also introduces stateless auto-configuration. Home devices can automatically construct their own IPv6 addresses with their MAC addresses and incoming router advertisements. This is extremely useful in home networks where owners are novices in network administration.

2. Application developers need not take complex steps to implement solutions to overcome NAT issues such as tunnelling and STUN. This allows developers to focus efforts better in application behaviour and protocol design without having to worry about NAT-traversal and other low-level issues.
3. The IPv6 protocol itself has been built with end-to-end security and mobility in mind. This is a significant advantage considering the number of anticipated mobile devices that have been targeted for use in wireless and 3G networks.
4. Routing in IPv6 is more efficient, owing to its hierarchical address architecture.
5. All major operating systems, which include Windows XP, Windows Mobile, Mac OS X, UNIX, Linux and Symbian, support both IPv4 and IPv6 out of the box.

While the benefits of migrating to IPv6 remain clear, moving from IPv4 to an all-IPv6 Internet is not expected to happen anytime in the near future, owing to the need for significant upgrades to all the intermediate network elements involved. Instead, the IETF has outlined and standardised technologies which aid the IPv4 to IPv6 transition. One of the clear advantages the standards aim to promote is to allow a smooth migration for the future, while ensuring that existing IPv4 traffic and current IPv4 services are not affected adversely. The most prevalent idea for aiding the IPv4 to IPv6 transition is based on tunnelling IPv6 traffic in IPv4 packets.

4. IPv6 tunnelling using 6to4

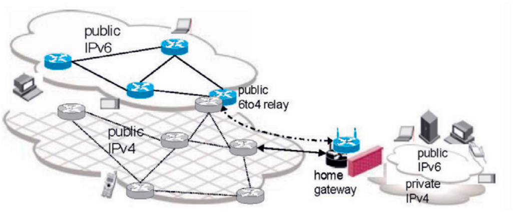


Fig 1. IPv6 home networks with 6to4

One of the most widely-used approaches to IPv6 tunnelling is 6to4 [2]. The 6to4 technique is a simple and effective way to set up a transitionary mechanism to bring IPv6 into an existing IPv4 network. Gateway devices at home can be set up as 6to4 border routers, which communicate in IPv6 by perceiving the IPv4 Internet as a unicast link layer. The tunnel setup is usually automatic, and requires minimal configuration. Such 6to4 automatic tunnels are created by locating the nearest 6to4 relay router using the well-known anycast address 192.88.99.1 [3]. Figure 1 shows how a typical setup looks like, with the home gateway having NAT capabilities in IPv4, but behaving as a 6to4 border router in IPv6.

The home gateway will also possess an entire subnet of the form 2002:ipv4address::/48, where ipv4address is the dynamic IPv4 address of the home gateway. Together with a 16-bit subnet, the home gateway can now behave as an IPv6 router in the home network, and can begin router advertisements containing this 64-bit prefix to the devices at home. Hosts in the home network use these router advertisements to configure their own IPv6 addresses, just like in a native IPv6 network. In this way, although IPv4 traffic and reachability remains unaffected with hosts in the home network having private IPv4 addresses, IPv6 addresses obtained will be global. A comprehensive description of 6to4 is given in [2].

5. IPv6 home network experiments

In order to conduct our tests, we decided to build an experimental wireless home gateway and a test network. This experimental home gateway was built using version 3.0 of the popular Linksys WRT54GS wireless router. OpenWrt [4], a Linux-based distribution containing many of the commonly found utilities for routing and packet filtering, was used as the router firmware. OpenWrt provides only a minimal firmware distribution with support for optional add-on packages, allowing a highly flexible customisation of the home gateway functionality.

IPv4 packages providing DHCP as well as NAT functionalities were installed to allow the router to act as a DHCP server and a NAT device. This allowed other devices in the wireless network to negotiate and obtain private IPv4 addresses.

At the same time, we installed OpenWrt packages that enabled IPv6 and 6to4 as described in the previous section. Once the gateway device received an IPv4 address, it then created a 6to4 tunnel device with the address 2002:ipv4address::1, to communicate with the nearest 6to4 relay router. It then began sending IPv6 router advertisement messages into the home network for home devices to automatically configure their IPv6 addresses. The /48 subnet from the router was subpartitioned to create a /64 subnet for the home network, the subnet arbitrarily chosen to be 1234. Therefore, the router advertisements were announced in the home network containing the prefix

2002:ipv4address:1234::/64. This is described informally in the sequence diagram of Figure 2.

With this home gateway setup, different services were deployed to provide access to clients sitting outside the home network. Research on content sharing at home reveals that right after general Internet usage such as www browsing and email access, the most widely shared content types in a home network are documents, photos and music [5]. Therefore, the following content sharing experiments reflected these three types of services and extrapolated sharing them out of the home networks.

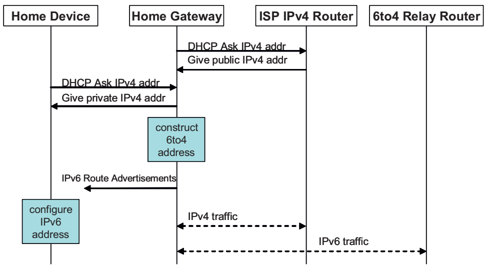


Fig 2. IPv4 and IPv6 connectivity for homes

5.1. Hosting a home web server image gallery

In the first test, we experimented with accessing a web server residing in the home network hosting photo albums over IPv6. An Apache web server was used together with Gallery version 1.5 [6], an open-source PHP software capable of organizing and sharing multimedia content such as digital images and videos over HTTP. Gallery's configuration was modified to allow users to register and log in via its web interface and the entire server was protected with a simple .htaccess password protection scheme. In addition, Gallery's web interface was also capable of allowing the administrator to configure various settings.

Clients used IPv6 from outside the home network to access the web server using common web browsers, as well as add, remove and post comments. Both native IPv6 access, with clients residing in true IPv6 space, and tunnelled access, with clients using IPv6 tunnelling using 6to4, were successfully tested.

In addition, IPv4 clients without IPv6 or 6to4 capabilities were also successfully able to connect and browse through the photo albums. This was achieved using a publicly available IPv6-to-IPv4 translation gateway [7] which allows clients from IPv4-only networks to view content from IPv6 websites.

This is especially important in some 3G networks which place access restrictions for IPv6-in-IPv4 traffic, resulting in IPv6-capable smartphones and/or 6to4 capable laptops being prevented from accessing IPv6 websites.

5.2. Hosting a file server for remote authoring

In this experiment, an Apache Tomcat web server in the home network was configured to allow access to local files through the use of Web-based Distributed Authoring and Versioning (WebDAV) [8]. WebDAV is a highly flexible protocol in providing writing as well as reading rights to files. It is also highly prevalent, with many operating systems contain built-in functionality for WebDAV, permitting application to perceive a WebDAV server as just another network drive or folder on the desktop.

Using an IPv6 enabled WebDAV server this way provided an extremely effective technique for file sharing, storage and access from outside a home network; any client with built-in or 3rd-party WebDAV support can be used to retrieve, edit and upload files directly on the server without any special editors, protocols or applications. Such files also included images or blog entries. In our experiments, we used client laptops running Windows XP as well as Mac OS X, to successfully mount WebDAV shares over IPv6.

5.3. Hosting a home audio streaming server

In the final test, we successfully experimented with accessing mp3 music streams originating from a home network. An IPv6-capable Icecast2 [9] streaming server was installed in a home machine running Ubuntu Linux. Icecast2 servers need source clients to originate the streams. We used a source client called IceGenerator [10].

IceGenerator resided in the same machine as the server. The source files for IceGenerator, however, did not reside locally on the machine. Instead, the Linux host also mounted shares containing mp3 files from 2 other machines in the home network, using SMB/CIFS over IPv4. These 2 other machines ran Windows XP and Mac OS X respectively. IceGenerator then retrieved mp3 files from these local mountpoints, and recursively fed them to the Icecast2 server. External clients listened over IPv6 to the audio streams using popular mp3 players such as iTunes, Windows Media Player, xmms and VLC.

Therefore, the test setup successfully involved IPv4 communication between the 3 nodes in the home network for file sharing, and IPv6 communication between the streaming server and the rest of the world.

6. Observations

In performing these experiments, we observed that the one of the primary requirements that need to be met before deploying IPv6 services from the home network, is the support for IPv6 transition mechanisms from ISPs, network operators and vendors. At the very minimum, the use of 6to4 requires at least 1 public IPv4 address from the ISP. In addition, ISPs should not prohibit using IP packet encapsulation, especially protocol type 41 (IPv6 in IPv4). For optimal performance, the public 6to4 relay router should naturally be as close as possible to the ISP. Our experiments used a publicly available 6to4 relay operated by the Finnish University Network, FUNET.

A client wishing to use the nearest 6to4 relay router would tunnel its packets to the IPv4 anycast address 192.88.99.1.This would raise doubts as to the trust relationship between the home network's 6to4 router and the closest 6to4 relay router which responds to the anycasting. However, only those ASes running 6to4 relay routers and are willing to provide access to the v6 network announce a path to the 6to4 anycast prefix [3]. Existing peering and transit agreements ensure control over 6to4 service provision and access.

Also, ISPs can encourage early adopters by providing their own 6to4 relays, IPv6 deployment instructions as well as experimental 6to4 firmware, as demonstrated by [11]. Other means of obtaining IPv6 tunnels to the home also exist. Teredo [12] can be used for end hosts if ISPs serve only private IPv4 addresses, and several Points of Presences (PoP) exist in many countries which provide free IPv6 tunnels [7].

Although the basic packages provided with OpenWRT performed adequately for our purposes, it is necessary to deal with changes in IPv4 addresses occurring whenever connections to the ISP from the home are dropped and need to be renegotiated. Instead of manually configuring our home gateway each time this occurs, we used some scripts in our home gateway to check for changes in the IPv4 address. If such a change occurs, the 6to4 tunnel device is recreated with the new IPv6 address, and router advertisements containing the new prefixes are also sent into the home network.

One important requirement for properly deploying IPv6 is the need for a proper DNS service. Home users typically are not expert administrators, and with IPv6 autoconfiguration, remembering numerical IPv6 addresses is highly difficult. Running a naming service inside the home network is equally unlikely. In all our experiments, servers in the home network used an external IPv6 Dynamic DNS provider [13], updating the DNS server whenever address changes occur.

7. Conclusions

Many network-level reports about IPv6 and transition technologies have been published, but very few studies, experiments and trials have been detailed to test the kinds of services commonly found in home networks.

A thorough performance analysis on the various services is needed. However our empirical experiments and observations show that the use of next generation networks and services in home networks holds real promise, and will strongly provide added value and market potential to broadband subscribers as well as network operators. With consumer broadband subscriptions representing a fast growing market segment, supporting such activities is very essential.

8. Acknowledgments

The authors wish to thank Antti Vekkeli and Jari Vanhala for their effort in installing OpenWrt and developing scripts for our prototype gateway device.

9. References

[1] J.Rosenberg, et. al, "STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, March 2003

[2] B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, February 2001

[3] C. Huitema, "An Anycast Prefix for 6to4 Relay Routers", RFC 3068, June 2001

[4] Linux OpenWrt project, http://www.openwrt.org

[5] Kurt Scherf, "Putting the Home Network to Work", Parks Associates White Paper, June 2005

[6] Gallery project, http://gallery.menalto.com

[7] SixXS IPv6 deployment effort, http://www.sixxs.net

[8] Y. Goland, et. al. "HTTP Extensions for Distributed Authoring – WEBDAV", RFC 2518, February 1999

[9] The Icecast project, http://www.icecast.org

[10] The IceGenerator project, http://sourceforge.net/projects/icegenerator

[11] Earthlink Research and Development, "Earthlink IPv6 in the Home", Last modified July 2005, URL: http://www.research.earthlink.net/ipv6/

[12] C. Huitema, "Teredo: Tunneling IPv6 over UDP through NATs", RFC 4380, February 2006

[13] IPv6 Dynamic DNS Service, http://www.dns6.org

Publication III

Developing Network Software and Communications Protocols Towards the Internet of Things

B. Silverajan and J. Harju

Fourth International ICST Conference on COMmunication System softWAre and middlewaRE (COMSWARE'09), pp. 9:1-9:8, 2009, ACM.

DOI: 10.1145/1621890.1621902

Developing Network Software and Communications Protocols Towards the Internet of Things

Bilhanan Silverajan and Jarmo Harju
Department of Communications Engineering
Tampere University of Technology
P.O. Box 553, FIN-33101 Tampere
+358 (0)3 3115 3906

Firstname.Lastname@tut.fi

ABSTRACT

One of the most profound changes today is the increase in mobility of portable yet powerful wireless devices capable of communicating via several different kinds of wireless radio networks of varying link-level characteristics. This paper addresses how the design and implementation of future applications and protocols can be facilitated by network programming frameworks. For the Internet of Things, upholding a clean layered network design while explicitly taking into account host capabilities, interfaces, device resources, device associations, multihoming, user policies and context management would impose many challenges for implementation support. The paper then presents a highly interoperable and lightweight event-based framework that serves many of these needs while also modelling, monitoring and handling events intrinsically present in communication architectures.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network communications, Wireless communication; C.2.2 [**Network Protocols**]: Applications, Protocol Architecture

General Terms

Performance, Design, Reliability, Experimentation.

Keywords

Protocol Engineering, Network Application Frameworks.

1. INTRODUCTION

Rapid developments in hardware and networking technology have today resulted in a diverse range of portable, yet powerful computing devices that are geared for connectivity and increased mobility over different types of networks. In 2005, the International Telecommunications Union (ITU) released a report outlining their vision of how networking, especially the Internet, will evolve in the face of increasing numbers of interconnected users and devices, entitled The Internet of Things [1].

The report outlines the enabling technologies, market potential, societal and standardization challenges, sustainable development and lifestyle implications. The report suggests that the number of users (both human and non-human) connected to the Internet would be counted in the billions. Devices are expected to greatly outnumber humans to become the biggest generators and receivers of traffic. Also, many of the nodes connected to the Internet would be extremely heteregenous in nature in terms of their computing resources, networking abilities and energy consumption. In addition to smartphones, laptops, PDAs and their various combinations, tiny RFID tags and pervasive RFID [2] have been touted as one of the enabling technologies for the Internet of Things, as are low power sensors and sensor networks as well as robotics and nanotechnology.

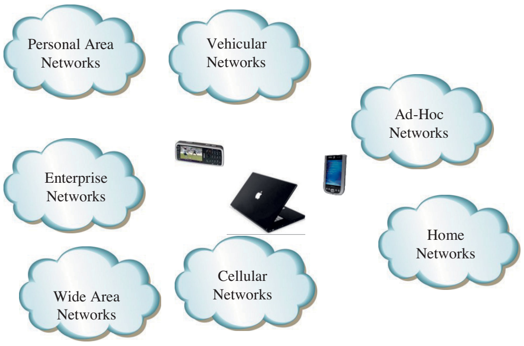


Figure 1. Communications protocols and networks incorporate a variety of contrasting fixed and wireless connectivity as well as device characteristics

In order to become truly ubiquitous though, the Internet of Things has to evolve to provide connectivity anytime, any place, for any Thing. The ubiquity of the wireless Internet is expected to pervade over multiple kinds of ad-hoc networks. As Figure 1 illustrates, these could range from personal area networks to ad-hoc and mesh networks to vehicular, community and metropolitan area networks. Such wireless technologies today include 802.11-based Wi-Fi LANs, WCDMA and EDGE-based cellular networks, and Bluetooth or IrDA based near-field communications. Wide-area metropolitan networks, such as 802.16 WiMAX, higher speed cellular networks based on HSDPA/HSUPA/HSPA+ and low-power energy efficient radio networks such as ZigBee are all widely expected to play dominant roles in the medium to long-term future for wireless mobility.

In addition to this range of potential wireless and radio technologies, wireless mobility at the network level, particularly for the Internet Protocol, must also be considered. Today, such mobility has been restricted to wireless terminals obtaining their IP connectivity via DHCP, either through a public wireless hotspot or via an ad-hoc network, with private IPv4 addresses used prevalently. However, research done in future IP-level efforts, particularly based on IPv6, have begun to produce research and industrial prototypes. For example, after the recent standardization of Mobile IPv6 [3], much research come to the forefront, focusing towards Network Mobility [4] and vehicular networking as well as bringing IPv6 to IEEE 802.15.4-based low power sensor, mesh and mobile ad-hoc networks [5]. Advances in delay-tolerant networking call for inordinate periods of disruption, in which end-hosts may not even be reachable end-to-end at any period in time [6].

This technical paper first gives an outline of the roles played by network middleware and frameworks in engineering network software and communication protocols, especially towards fulfilling the technical vision of The Internet of Things. The paper then presents the design and implementation of a highly portable and lightweight event-based network framework capable of modelling, monitoring and handling events intrinsically present in today's communication architectures. These range from minute, low-level interface information such as movement detection and network attachments and detachments, to messages in communication protocols, to high-level object-based communication models.

2. ROLES OF FRAMEWORKS AND MIDDLEWARE

The prevalence of the Internet as a communication medium and the availability of cost effective tools has opened the floodgates for the development of a multitude of network applications and protocols today. Application interaction with transport-layer protocols is performed via system calls and APIs to the underlying operating system, whereas application interaction with higher-level protocols normally tend to occur either as API calls to a library implementation or a language-level module which in most cases becomes part of the application itself. Consequently, developers tie in the implementation of network software and applications very closely with the design and implementations of the utilized communication protocols themselves. It is of little wonder then that these protocol implementations tend to impact the size, performance and robustness of the resulting applications. Hence it is important to understand how exactly these communication protocols are developed.

In terms of applications, new breeds that are location and context sensitive are rapidly being developed. On the micro level such as in sensors, the boundaries that define applications, protocols and operating systems are distinctly blurred. At the macro end of the scale large applications consisting of many components may need to be constantly aware of their environments at all times. Wide-scale mobility also implies significant changes in the way physical, link and network layer characteristics need to be exposed to higher layer protocols. The strict modularity proposed by the OSI model needs to be relaxed for low-level access to network and link level characteristics of the host they execute on. Often, such communication may not even occur asynchronously using packets and service access points. Instead, low-level libraries provide blocking function calls that applications can explicitly use to bypass the stack entirely.

Today's mobile user possesses multiple wireless devices that may have overlapping functionalities. User policies determine both device and application behaviour, ranging from issues related to pricing, controlling hardware functionality, multipath/multiradio communication and interaction with the immediate environment. Although simultaneous communication among endpoints using two or more radio interfaces is not yet easily achievable today, it is highly conceivable that multipath device communication would become a reality, forming and dissolving associations and ad-hoc connection paths as necessary, but belonging to the same context of usage. Personal and body area networks tomorrow may consist of a multitude of communicating applications working concurrently on behalf of the user to participate in the Internet of Things. Consequently, to cater to these demands, traditional networking frameworks need to evolve and become capable of explicitly accounting for these factors.

With the increase in mobility, the way service discovery and lookups are performed would also become significant, as devices would need to search for essential services upon movement. How broadcast and multicast capabilities can be utilized for active service discovery is mired in understanding both the link level and network level characteristics. For example, even though both a mesh wireless network and a Wi-Fi network may provide IP connectivity, broadcasting and multicasting capabilities in both networks would be vastly different. Armed with both link and network information however, discovery mechanisms may seek to either utilize broadcast and multicast capabilities present, or need to rely on point-to-point communication with a lookup service.

This wide variety of mobile environments, coupled with the distinct possibility of fluctuating bandwidth, frequent intermittent connectivity and performance/power constraints experienced by mobile devices, presents a daunting undertaking for the development of advanced applications and protocol implementations. When the range of activities in designing communications components is so widespread, frameworks and middleware begin to occupy a central position. Some roles they already fulfill are:

- Providing support for formal representations and graphical notations during requirements and specification.
- Providing generative support for transforming specifications, further formalizations and code generation.
- Assisting rapid implementation by providing useful libraries and building blocks.
- Transforming requirements, analysis and design models for simulation.

- Facilitating system verification by generating test models and test suites as well as by usage and behavioural testing.
- Aiding rapid deployment and performing activities related to release, installation, activation, update and removal of unneeded components during the entity's lifetime.

Network middleware and communication frameworks are set to play a major role in many aspects of the future Internet and especially the Internet of Things. The demands of executing protocols and network applications in dynamically changing network conditions would place a premium for such frameworks to actively deliver environmental data and timely semantic information to interested components, and not just provide a "middleware" layer between the application and the underlying platform that irons out differences in heterogeneity.

In fact providing platform-specific support to leverage cross-layer features, context management, link monitoring, power consumption, notification of connectivity and interface changes would become crucial elements to increase the effectiveness of an application or protocol relying heavily on the supply of proper context information for its execution environment.

If the framework or middleware environment induces runtime dependencies from resulting application, the actual size of the libraries and executables are also of great importance when we take portable devices into consideration.

We postulate that lightweight message- and event-based frameworks and middleware represent a good balance and tradeoff in developing many different kinds of communications components, easily being able to model and map discrete events occurring at many levels of granularity, ranging from deep within the detailed implementations of network components to the way those network components intelligently interact with each other.

3. THE DOORS FRAMEWORK

DOORS is a C++ network programming framework we developed for implementing protocols and network applications which is aimed at alleviating some of the communication concerns outlined earlier. The framework can also be used for developing network services, ranging from simple socket-based systems, to distributed applications, object-based gateways as well as general event-based client-server applications. This highly generic framework is founded on the event-driven programming paradigm, in which callback functions are written by the developer and invoked by event handlers. All network, application and user events are mapped into messages by DOORS that are then handled using a callback scheduling and I/O Handling model.

In terms of portability, DOORS is both portable and lightweight. The developer is not hindered from exercising any of the advanced C++ language-level features such as templates and exception handling, nor is prevented from using other frameworks and third party libraries as part of the developed application. This is a single-threaded framework that can be compiled and used across virtually all UNIX and Linux variants. Recently, support for DOORS was also extended to encompass embedded systems based on the Maemo platform used by Nokia Internet Tablets and OpenWRT, a Linux-based firmware distribution for wireless access points.

3.1 Framework Architecture

The DOORS framework conceptually resides between the operating system and the application, and provides a uniform development environment shielding the developer from operating system specific issues.

DOORS reduces any event-driven application to be implemented, into a set of interacting stateful Tasks, Ports and Messages. Tasks communicate with each other through connected Ports by passing Messages. Each Message represents an event, such as a user input, incoming protocol packet, timeouts, errors and so on.

Therefore, the resultant design and implementation of an application, as well as its interaction with DOORS, is largely asynchronous and message-based. This also allows the framework to address requirements for decoupled, asynchronous interactions in large-scale distributed systems. Figure 2 depicts how the various modules, subsystems and tools of the framework, together with the machine's operating system and external libraries, can be architecturally represented. Some of these features are described in this section.

The Protocols Store contains several existing protocol implementations that a developer can directly use. Implementations currently available include the Real-Time Transport Protocol (RTP) and RTP Control Protocol (RTCP) [7] for audio streaming, an IPv6-enabled Service Location Protocol (SLP) [8] for service discovery, Session Announcement Protocol (SAP) [9] for session announcements as well as UDP and TCP helper classes for both IPv4 and IPv6.

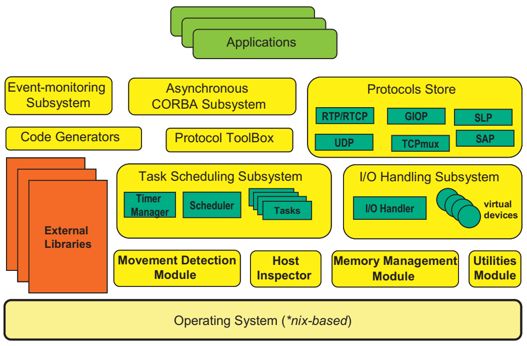


Figure 2. DOORS Architecture and logical subsystems

The Protocol Toolbox contains specialized Protocol Tasks for aiding protocol development, finite state machine implementations, Service Access Points, and encoding/decoding of Protocol Data Units, bi-directional Ports used by Tasks for message passing and multiplexers that allow M:N communications between Tasks.

The Asynchronous CORBA Subsystem comprises a lightweight, library-based CORBA solution for asynchronous communications which consists of an Object Request Broker (ORB), a Portable Object Adapter (POA) and dynamic interfaces for client-side and server side in the forms of the Dynamic Invocation Interface (DII) and the Dynamic Skeletal Interface (DSI), which run atop IIOP.

All the protocols in the Protocols Store as well as the components of the Asynchronous CORBA subsystem, are implemented as interacting stateful Task objects in DOORS. The Task Scheduling Subsystem consists of a scheduler handling requests from Tasks, and providing execution turns to these Tasks using a priority-based callback mechanism. The Subsystem also contains timers that can be used for providing soft real-time guarantees together with a timer manager.

The Event-monitoring Subsystem consists of tools used at runtime to monitor various parts of a running system such as the state of tasks, internal variables, the number of messages processed, their types, timers as well as the scheduler load. The Subsystem contains a local event monitor capable of inspecting intra-process events in a locally running application, a distributed event monitor capable of inspecting events simultaneously in multiple DOORS-based applications running in a network, tools for logging events into files as well as a hierarchically organized symbol handling interface through which the event monitors communicate with the application.

In addition, the Utilities and Memory Management Modules provide basic support for implementation and runtime memory management. The Utilities Module offers basic datatypes, structures and class templates. Frame and cell classes provide flexible ways in storing and manipulating large octet arrays. The Memory Management Module offers advanced memory management featuring basic, statistical and block memory allocations.

The I/O Handling Subsystem, Host Inspector, Code Generators as well as the Movement Detection Modules are described in subsequent subsections.

3.2 Host Capabilities and Device Interfaces

For any programming framework to provide effective support for implementing protocols and other communications components, it is essential to provide access to a device's interfaces and hardware configurations as well as have a flexible yet efficient input/output handling system.

Facilitating the former is the responsibility of the Host Inspector while the latter is handled by the I/O Handling Subsystem.

The Host Inspector is a single instance that can be queried by all running DOORS application for information pertaining to the device itself and its associated interfaces. Such information includes processor type and endian architecture, host operating system and version, list of all fixed, virtual and radio interfaces present as well as their currently assigned addresses.

The I/O Handling Subsystem in the framework is directly responsible for allowing event-driven tasks and applications a consistent interface to communicate with external entities by introducing the concept of a virtual device. Virtual devices model aspects of communications activities such as network endpoints or operating system inter-process communication (IPC) APIs as objects within DOORS. Such virtual devices currently include IPv4 and IPv6 network devices that perform both unicast and multicast communication, stream I/O facilities such as files and pipes as well as operating system services such as terminals. Virtual devices export a uniform interface to tasks, via functions such as *open()*, *close()*, *read()* and *write()*. Actual data transfer is however, asynchronous. Hence, writing to a device would return immediately; the data will be stored in a buffer until a later point in time when an execution turn is obtained from the scheduling subsystem, at which point a callback function is then invoked to perform the actual bytewise transmission.

The subsystem relies on a single functioning I/O Handler to manage the devices required for communication. The I/O Handler is responsible for obtaining execution turns from the scheduler to ensure that device read and write callbacks can be performed. In essence, each virtual device possesses an identifier represented by the operating system as a file descriptor. The virtual devices are registered with the I/O Handler using their identifiers. The Handler monitors the status of each virtual device's read and write file descriptors to detect activities and pending events on each device. When an event of interest is detected, a callback routine in the virtual device is invoked by the I/O Handler to handle the event.

Consequently, as long as the host operating system or an application's API supports I/O communication using file descriptors, DOORS is capable of being extended to support the technology in question using virtual devices.

This methodology provides DOORS the flexibility to natively support new interactions with various kinds of network and host interfaces whenever such a need arises. As an example, this mechanism enabled us to quickly prototype and implement virtual devices capable of Bluetooth communication with Radio Frequency Communication (RFCOMM) [10] and Logical Link Control and Adaptation Protocol (L2CAP) [11]. The prototype has been developed in Linux-based devices that use the BlueZ official Linux Bluetooth stack. Bluetooth APIs are not standardized across different operating systems. However, the framework's Bluetooth virtual devices can be easily extended towards other Bluetooth stacks and operating systems that DOORS can run atop of. Because the virtual devices export a consistent interface to applications and tasks in DOORS, it provides a very portable means to users of the framework to develop Bluetooth-based communications.

It is also conceivable for a developer to extend the framework's virtual devices to encompass other kinds of communication technologies and serial communications, such as ZigBee or even IrDA.

3.3 High-level Representation of Services and Messages

Information flow and service models in DOORS are consistently expressed using terminology from the OSI layered model. For example, messages are usually indicated as Service Data Units (SDUs) and are represented entering and leaving Service Layers and objects via Service Access Points (SAPs). Peer abstraction as well as invoking encoding and decoding rules upon Protocol Data Units (PDUs) are instilled in a separate Peer class.

Owing to heavy influence from the CCITT (later renamed the ITU-T) and ISO standards bodies, service and protocol specifications in classical protocol engineering have been performed with formal specification languages, especially SDL. This has changed today, with the majority of applications and protocols now being designed for the Internet. Such protocols are described informally as text-based standards documents by the Internet Engineering Task Force (IETF). With the circle of involvement for protocol engineering expanding greatly beyond the telecommunications domain, specifications are being issued in other formal or semi-formal ways, such as the use of XML by the World-Wide Web Consortium (W3C) or with IDL by the Object Management Consortium (OMG).

For this reason, the initial specification of implemented messages and task behaviour in DOORS that eventually are deployed as protocols, applications or network services, are written in XML by the developer. These XML specifications are then parsed by generators in DOORS at compile time to produce corresponding framework specific C++ classes such as Messages, Service Access Points, Service Primitives as well as extended finite state machines.

Figure 3 provides a rough idea of how a few of the Protocol Data Units (PDUs) of the Service Location Protocol are specified in DOORS using XML, while Figure 4 (which is explained in the next section) provides an example of how movement detection events garnered from a third party library are also specified in XML.

```
<Peer Name="SLPPeer">
 <Message Name="slpHeader">
    <Field>Uint8 version</Field>
    <Field>Uint8 functionID</Field>
    <Field>Uint32 packetLength</Field>
    <Field>Uint16 flags</Field>
    <Field>Uint32 extOffset</Field>
    <Field>Uint16 XID</Field>
    <Field>std::string langTag</Field>
 </Message>
 <Message Name="SRVRQST">
     <Parent>SLPHeader</Parent>
     <Field>AddrVector PRList</Field>
     <Field>std::string servicetype</Field>
     <Field>StringVector scopeList</Field>
     <Field>std::string predicate</Field>
     <Field>std::string spi</Field>
 </Message>
 <Message Name="SRVRPLY">
      <Parent>SLPHeader</Parent>
      <Field>Uint16 errorCode </Field>
      <Field>URLVector urlEntries</Field>
 </Message>
 <Message Name="SRVREG">
      <Parent>SLPHeader</Parent>
      <Field>ServiceURL url </Field>
      <Field>std::string  serviceType </Field>
      <Field>Uint16 errorCode </Field>
      <Field>StringVector  scopeList</Field>
      <Field>StringVector  attributeList </Field>
      <Field>AuthBlockVector  AttrAuthBlocks</Field>
 </Message>
….. more messages….
 </Message>
</Peer>
```

Figure 3. XML specification describing some SLP messages to be sent and received in an IPv6 network

Although the specifications are written in XML, the target is to use the DOORS Code Generators for the resulting datatypes to be native C++ objects. However using XML for message and protocol specification serves as a natural springboard for developers when designing communication architectures to extend its usage from simply being an "easy to read and write" specification document towards more powerful XML paradigms to encompass document templates, transformations and scripting with other XML technologies for the future.

This allows for a portrayal of the resulting software architecture which renders itself easy for future extensions as well as interoperability with other potential external services and applications. Any relevant kind of network or even simulation framework capable of producing events or traces as an XML output can be transformed for usage into DOORS. This also provides a natural way of expressing users, user policies, preferences and events to be fed as messages into a running DOORS system and vice versa.

In addition, considering the various ways in which protocols and network components are specified, this form of representation of services provides a neutral approach towards normalizing the usage of DOORS in various vertical networking domains.

3.4 Service Discovery for Mobile Devices

Mobile devices need rapid service discovery mechanisms that are neither predicated on the presence of lookup mechanisms nor on specific network topologies; the ability for mobile applications to detect movement, dynamically discover services and potentially even offer services in network spaces ranging from ad-hoc to infrastructure oriented environments presents an interesting research challenge.

Movement detection and service discovery in DOORS are handled by the Movement Detection Module as well as an IPv6 implementation of the Service Location Protocol (SLP) [8].

The Movement Detection Module handles interaction with low-level libraries and mobility protocol APIs, obtaining movement information such as a change in the network address or default route, and exporting it in a consistent manner for use by the other subsystems in the framework. It works in IPv6 and Mobile IPv6 networks, by monitoring changes in the host IPv6 address on behalf of all the various applications and components using or comprising the DOORS framework. Upon an address change, this is propagated to all other interested components by supplying the information in DOORS-specific messages.

Currently no POSIX-compliant APIs or methods yet exist in an operating system that allow any application to directly retrieve this information. In fact, kernel-space hooks are currently necessary to obtain the information directly from the TCP/IP protocol stack implementation. In the case of the Movement Detection Module, such low-level information is obtained by interfacing with a low-level library called libmobinfo [12], a small C-based library, whose sole purpose is to provide registered applications with changes occurring to the IP address as soon as the mobile node's IP layer itself becomes aware of it. Libmobinfo exports an API that can be used both synchronously and asynchronously.

Tasks send messages to the Mobility Module to register or unregister their interest in receiving movement notification. Upon movement, the Mobility Module converts the movement information received from libmobinfo into a message for processing by other DOORS Tasks. At the moment, this message contains 2 fields: The mobile node's home address, and the current care-of address. When a mobile node ventures into a foreign network, these addresses will obviously differ. However, when the mobile node returns back into its home network, both fields would contain identical values for the IPv6 address. These messages are described in XML in Figure 4.

```
<SAP Name="MobilityDetectionModule">
 <User>
   <Message Name="Register">
     <Field>String TaskName</Field>
     <Field>Uint32 Taskid</Field>
   </Message>
   <Message Name="Unregister">
     <Field>String TaskName</Field>
     <Field>Uint32 Taskid</Field>
   </Message>
 </User>
 <Provider>
   <Message Name="Movement">
     <Field>InetAddr6 coa</Field>
     <Field>InetAddr6 ha</Field>
   </Message>
 </Provider>
</SAP>
```

Figure 4. XML specification describing the Mobility Module's Service Access Point (SAP), containing the message types to be sent through its port, and categorised into a User section (comprising messages other Tasks send to Mobility Module) and a Provider section (comprising messages Mobility Module sends to the other Tasks).

Using the Mobility Module this way, it becomes easy to provide movement detection information to applications from the framework level: The applications themselves receive the information in a consistent manner of communication, and need not be concerned if changes occur to mobinfo. The Mobility Module on the other hand can easily be modified if such changes occur, without affecting the application logic of the other Tasks. Extra messages can also be defined in the future containing richer information (such as geographical co-ordinates, context information, router prefixes, etc).

The SLP implementation in DOORS comprises part of the Protocols Store and is able to provide support for service discovery to other interested components and applications.

SLP is a very versatile protocol, and it supports a high flexibility in configuration. Services can be discovered and advertised via multicast dynamically using agents, or agents' behaviors can be configured statically, to avoid using multicast, and so on. However, SLP does not support the notion of automatic service rediscovery, nor changes in network topologies, and relies on external methods for providing these facilities. Consequently, its use has been confined to fairly fixed network topologies.

The DOORS implementation of SLP extends the standard SLP towards usage in Mobile IPv6 in 2 significant ways [13]. Firstly, it understands and handles movement of the mobile node. When information pertaining to movement and a change in the network topology is received, the protocol is able to take corresponding steps to automatically discover services in the new network. Secondly, the protocol now supports the notion of persistent and transient services. This allows a high level of freedom to applications offering services, in selecting whether the service will be always reachable regardless of where the mobile node is currently located, or whether the service will only be offered and reachable at the current point of network attachment.

3.5 Interworking and Interoperability Considerations

The heterogenous nature not only of networks, but also the existence of other systems, tools, middleware, frameworks and platform specific libraries must be taken into account when considering overall system development. Indeed, it could be argued that since many frameworks and tools are optimized for delivering specialised feature sets or facilitating specific activities, it is extremely vital for tomorrow's frameworks to co-operate and exhibit a high level of interworking and interoperability to preserve their viability for the long-term.

In order for this to successfully occur at least one of the 2 types of integration and interoperability approaches needs to be supported by network programming frameworks: Protocol-level interoperability or Service level interoperability.

Protocol-based interworking and interoperability is a low-level approach harmonising service-specific message-based interaction with different gateway and conversion technologies, packet translation and protocol bridging to adapt different signalling and transport protocols. For example, many of the actively participating nodes anticipated in the Internet of Things would be miniature. Examples include ZigBee sensors, RFID tags and regular to ultra-low power Bluetooth devices. Being able to implement a gateway to interconnect these devices and networks into the Internet would require such a bridging and translation mechanism which qualifies as protocol-based approach.

Service-level interworking and interoperability is a higher-level approach that allows different systems and services to act consistently and work in parallel. This is possible as long as these various systems speak a common protocol. In many enterprise and core telecommunications networks, service level interworking has usually been implemented with different intermediary device and middleware integration solutions. These systems integrate different network components with service provisioning, billing systems and operations support systems to implement consistent data flow across these systems.

DOORS supports both protocol and service level approaches natively. As an example, the framework can be used to bridge smart devices and home automation, providing applications the ability to simultaneously communicate over several wireless technologies such as WiFi and Bluetooth: A wireless device such as a smartphone or an Internet tablet can push content from the phone to a media server or DLNA-enabled television using UPnP over WiFi, while using a low power Bluetooth interface as a remote control. As mentioned in section 3.2, DOORS virtual devices can be easily extended to encompass other kinds of networks as well. Therefore DOORS Tasks can serve as sensor gateways, bridging and translating data to and from both ZigBee as well as 6LowPAN [5] wireless sensor networks.

At the service level, many examples exist. The Service Location Protocol Agents which have been implemented using DOORS are fully compatible with other IPv6 implementations. SLP can therefore be used for actively discovering various location-based services by other applications, frameworks and tools. These can range from simple services such as printing and file servers to more network centric services such as the IPv6 address of a Domain Name Server in a foreign network by a visiting Mobile IPv6 node, or the spatial co-ordinates of a specific access point within a building to which the device is attached to.

At the same time, support is not only limited to specific software components of protocols. In considering research on pervasive RFID and the Internet of Things, some open-source middleware architectural support, such as mentioned in [14] will allow DOORS Tasks to join as subscribers to an RFID Agent either point-to-point or over a multicast channel to receive RFID information as XML strings which can then be processed.

Indeed, the advent of open source programming and the vast number of high quality open source projects on the Internet has led to a fundamental shift in the paradigm of code sharing and usage. Higher productivity and efficiency result from the ability to interact with binary, platform-specific third party libraries as well as from the ability to integrate and reuse third party source code within DOORS.

In addition to these approaches, DOORS also allows a few other kinds of approaches rendering the framework easy to interwork or even integrate into other middleware and frameworks.

As an experiment in facilitating support for distributed object computing using asynchronous communication in next generation IP networks, DOORS provides support for object-based software development using asynchronous CORBA communication. In effect, this means explicit support for CORBA/IIOP over standard, as well as Mobile IPv6 networks. Thus, a higher level of architectural integration with object-level CORBA interoperability is also provided by DOORS.

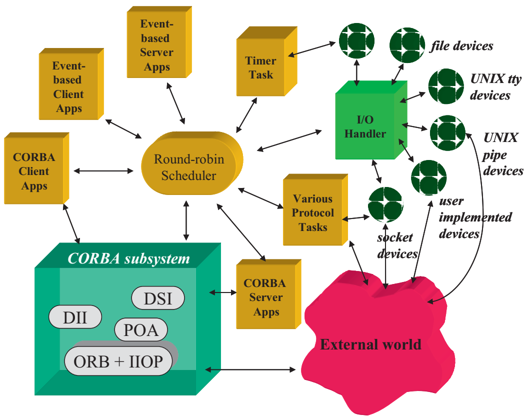


Figure 5. Multi-level communications and interoperability

Consequently, this makes it easy to develop systems using DOORS to communicate at multiple levels simultaneously, possessing the capacity of harnessing inter-process communication, virtual devices, protocols and the asynchronous CORBA system to interact with many types of external programs, and possessing the ability to translate messages arriving from one level to another. Figure 5 illustrates this, showing how several different kinds of applications can simultaneously be interacting with external applications.

3.6 Related Networking Frameworks

The Adaptive Communication Environment (ACE) [15] is widely regarded as one of the best known and prevalent framework for use in C++ network programming. ACE boasts an impressive development, implementation and deployment record, and has been ported to a wide multitude of platforms. Its feature list range from the ability to execute in resource constrained environments, mission critical and real time systems, high performance platforms, supporting concurrency and multiprocessing and multithreading. ACE makes extensive use of design patterns both for documentation and as a source of guiding programmers to become familiar with its various features to develop their protocols. CORBA support with ACE is provided via the The ACE ORB (TAO). Both ACE and TAO are commercially supported and enjoy wide industrial adoption. In contrast the DOORS framework relies on the developer's familiarity with common protocol engineering concepts and provides explicit support for Finite State Machines, Service Access Points, Peer abstraction and Protocol Data Units. In addition, DOORS provides a pluggable textual user interface capable of plugging in to a protocol or protocol stack under development for direct interaction by the developer, sending and receiving protocol messages, inspecting states and variables at run-time and performing interactive or automated conformance as well as white-box testing.

The Twisted [16] networking framework is also an open-source event-driven framework. Because it is written entirely in Python, Twisted is able to execute atop any platform which supports Python. Twisted has an active developer community and has subsequently amassed an extensive range of protocols that are ready to be imported as modules for immediate use in Python applications. Although Twisted does not contain direct support for object-based middleware communication, a quick browse through the framework's website would reveal that Twisted has been used for many kinds of network services which include web servers, ssh clients and instant messaging systems.

All 3 frameworks (DOORS, ACE and Twisted) adopt the same basic event-driven principle of inverting the flow of the program by using a single event loop. However, all three also implement similar event dispatching algorithms that correspond to the Reactor design pattern [17], whereby a single handler monitors resources and registered objects for arrival of events. Upon such arrivals, the handler subsequently demultiplex incoming requests and delegates individual event handlers to process associated events. In DOORS, the Reactor is implemented by the I/O Handling subsystem. Consequently DOORS can interwork with the foreign event loops of both the ACE and Twisted frameworks. However, in the case of Twisted, DOORS will need to rely on the Boost C++ library Boost.Python, to provide language level interoperability.

4. Conclusions

As networks of the future continue to evolve and converge, the need for simple visual models that can continue to serve as reference points which translate into working implementations of communications protocols and networking application remains dire. Network models must begin to take in to account users, devices and usage context to cope with the growing complexity of communication architectures.

In order to aid such development of communications protocols and software, we introduced the features of a generic C++ networking framework called DOORS. We explained the various ways in which DOORS can be used while remaining highly interoperable with many kinds of external libraries, frameworks and other middleware solutions developers may already be familiar with. By adopting familiar terminologies and concepts for protocol development in DOORS, we aim at eliminating the steep learning curve that appear to be inherent in learning to use network programming frameworks.

5. REFERENCES

[1] International Telecommunication Union 2005. The Internet of Things. ITU Internet Reports, November 2005.

[2] Michahelles, F., Thiesse, F., Schmidt, A., and Williams, J. R. 2007. Pervasive RFID and Near Field Communication Technology. IEEE Pervasive Computing 6, 3, (Jul-Sept 2007), 94-96.

[3] Johnson, D., Perkins, C., and Arkko, J. 2004. Mobility Support in IPv6. IETF RFC 3775, June 2004.

[4] Devarapalli, V., Wakikawa, R., Petrescu, A., and Thubert, P. 2005. Network Mobility (NEMO) Basic Support Protocol. IETF RFC 3963, January 2005.

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

[6] Cerf, V., and Burleigh, S. 2007. Delay-Tolerant Networking Architecture. IETF RFC 4838, April 2007.

[7] Schulzrinne, H., Casner, S., Frederick, R., and Jacobsen, V. 2003. RTP: A Transport Protocol for Real-Time Applications. IETF RFC 3550, July 2003.

[8] Guttman, E. 2001. Service Location Protocol Modifications for IPv6. IETF RFC 3111, May 2001.

[9] Handley, M., Perkins, C., and Whelan, R. 2000. Session Announcement Protocol. IETF RFC 2974, October 2000.

[10] Bluetooth Special Interest Group (SIG). 2003. RFCOMM with TS 07.10. Bluetooth Specification Version 1.1. June 2003.

[11] Bluetooth Special Interest Group (SIG). 2007. Core Specification v2.1 + EDR. July 2007.

[12] Kalliomäki, J., Silverajan, B., and Harju, J. 2007. Providing Movement Information to Applications in Wireless and Mobile Terminals. In Proceedings of 13th Open European Summer School and IFIP TC6.6 Workshop (Enschede, Netherlands, July 2007). EUNICE 2007.

[13] Silverajan, B., and Harju, J. 2007. Factoring IPv6 Device Mobility and Ad-hoc Interactions into the Service Location Protocol. In Proceedings of IEEE Conference on Local Computer Networks (Dublin, Ireland, October 2007). LCN 2007.

[14] Hoag, J. E., and Thompson, C. W. Architecting RFID Middleware. IEEE Internet Computing 10, 5 (September - October 2006), 88-92.

[15] Schmidt, D. C. 1993. The ADAPTIVE Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Software. In proceedings of 11th Sun User Group Conference (Brookline, MA, USA, December 1993).

[16] Fettig, A. 2006. Twisted Network Programming Essentials. O' Reilly Media Inc. ISBN 0-596-10032-9.

[17] Schmidt, D. C. 1995. Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching. Pattern Languages of Program Design (Software Patterns Series), Chapter 29. Addison Wesley 1995 ISBN 0-201-60734-4.

Publication IV

Factoring IPv6 Device Mobility and Ad-hoc Interactions into the Service Location Protocol

B. Silverajan and J. Harju

32nd IEEE Conference on Local Computer Networks (LCN 2007), pp. 387-394, 2007, IEEE.

DOI: 10.1109/LCN.2007.108

Factoring IPv6 Device Mobility and Ad-hoc Interactions into the Service Location Protocol

Bilhanan Silverajan and Jarmo Harju
Institute of Communications Engineering
Tampere University of Technology
Tampere, Finland
Firstname.Lastname@tut.fi

Abstract— **The rise in device mobility, interactions for service discovery as well as IPv6 networking, necessitate a means by which a clear and consistent approach towards supporting rapid, dynamic service discovery and service provision must be undertaken. This paper describes ways to provide movement detection, automatic and dynamic discovery of network services as well as network characteristics at the point of attachment for mobile devices moving in IPv6 network spaces. It employs a service discovery mechanism based on using the Service Location Protocol (SLP), one of the most extensively researched and standardised discovery protocols in existence today. We show why traditional means of service discovery have shortcomings when used in advanced mobile networking environments, and describe new extensions to overcome these issues without losing compatibility to existing work in SLP. Prototypes developed and tested from the work described provided empirical verification over several of our production and research IPv6 networks.**

Keywords-Service Discovery; IPv6; Mobility

I. INTRODUCTION

Research on service discovery mechanisms has increasingly gained prominence and importance owing to tremendous advances in consumer mobile computing. More than ever, the use of portable devices and the ease with which they roam into various kinds of network environments is fuelling a need for a simple yet scalable model through which users, devices and mobile applications can be automatically discovered for interaction. More importantly, new breeds of applications are emerging in which traditional paradigms of networks and services are being tested to their limits. Mobile devices need rapid service discovery mechanisms that are neither predicated on the presence of lookup mechanisms nor on specific network topologies; the ability for mobile applications to scale, interact and dynamically discover services from small, ad-hoc associations among a set of bandwidth and resource constrained terminals, to infrastructure oriented communication within large enterprise-level bandwidth-rich environments, presents an interesting research challenge.

Additionally, portable devices have increasingly begun carrying their content and services with them. Therefore, not only will these devices attempt to locate and access resources in their immediate environments, they will also need mechanisms with which they can advertise and offer their services into network spaces they roam into. To date, very few service discovery mechanisms exist which are both lightweight and dynamic, yet remain compatible with existing standards that could meet these requirements.

This paper describes a powerful, robust and very adaptable technique which employs a service discovery mechanism that can meet the needs of such a scenario. This approach is based on using the Service Location Protocol (SLP). Among the various IP-based service discovery protocols in existence, the IETF standardised SLP remains one of the most extensively tested and utilised to date. It is also one of the most flexible, scalable and lightweight discovery protocols to be conceived. Applying SLP for use in IPv6-based mobile environments however, revealed ambiguities and inadequacies in the standards. The work described in this paper attempts to overcome those limitations, and extends the usage of SLP in a clear and consistent manner towards supporting rapid, dynamic service discovery and service provision by mobile devices in IPv6. Our experiments with a prototype have provided empirical verification using the prototype over several of our production as well as research networks.

Section II provides an introduction to the Service Location Protocol, while Section III discusses some related work done on SLP itself within the IETF, extending it in various ways towards remote service discovery, IPv6 and notifications. Section IV contains a thorough description of the motivations of our service discovery mechanisms, while sections V to VII describe the enhancements necessary to properly realise the service discovery for mobile environments in IPv6 using SLP.

II. SERVICE LOCATION PROTOCOL: BACKGROUND

The Service Location Protocol (SLP) was standardized by the IETF SRVLOC Working Group in 1997 [1]. It was further revised in a second version in 1999 [2]. Subsequently, unless otherwise explicitly stated, all work done on and with SLP from that time on has referred to version 2 of the protocol.

Being such an established protocol, SLP has formed the basis of application and device service discovery in enterprise networks and products from several commercial vendors such as Apple Computers, Hewlett-Packard, IBM, Lexmark, Novell and Sun Microsystems. A standalone open-source version in both C and Java, called OpenSLP is also available [3]. Apart from these, SLP is applied widely in different ways: Recently, the IETF also ratified the usage of SLP for use in iSCSI as a basis for device discovery [4], while ANSI recommends SLP usage in its Architecture for Control Networks specification [5] as a discovery protocol for use in lighting systems, media servers and theatrical entertainment Equipment. Usage of SLP can also be found as modules in configuration, monitoring and management tools and frameworks [6], [7].

SLP's entire operation and logic is contained within entities known as agents, and a protocol used in interactions among them. Applications, middleware and other protocols can then use these agents to perform service discovery, advertise and offer existing services or perform updates whenever the services or their parameters change. The agents defined in SLP are the User Agent, the Service Agent and the Directory Agent (DA). Of the three, the UA and the SA are mandatory, while the DA is an optional entity.

The UA aids client applications in discovering the locations of services. Server applications register their service with SAs with a Service Type string URL. When there are only UAs and SAs present on a network, all service requests are sent to the administratively scoped multicast address 239.255.255.253. For example, a print server might register itself to an SA with "service:printer:ipp://printserver.mycompany.com/printer1". The SA would then respond to any UA multicasting a request for a printing service, such as service:printer" or "service:printer:ipp". SAs also multicast SA Advertisement messages on the network if solicited by a UA. In the absence of multicast, broadcast may also be used.

The DA provides a centralized service for all service announcements in a very large network, so that a point of single contact exists for a UA trying to discover various services. DAs also advertise their presence periodically and SAs are required to register their services with DAs they discover from DA Advertisements. UAs also interact directly with the DA instead of SAs, if one is present in the network. In this mode, UAs essentially use the DA as a lookup service.

There can be multiple UAs, SAs and DAs residing in the same network. For performance reasons, the UA normally exists in the same host as client applications. UAs can be a standalone process, but are also commonly found as a component or components, for example in middleware, that multiple client applications can use. In this sense, instead of being a standalone process, they can be invoked as a library function. SAs tends to reside on the same hosts as server applications, but they could also reside elsewhere in the network too. Depending on how they are configured, SAs tend to handle either a single service URL or a closely related set of services.

The number of DAs in the network are generally not as prevalent as UAs or SAs. Although SLP advocates the usage of a single DA in a network, the upper limit is not defined and it is not uncommon to find large enterprise networks that have tens of DAs running. The stipulated mandatory rule is that every SA must register its services with all DAs that are present in the network.

The discovery of a DA in a network by the UA can be accomplished in 3 ways: The UA can be statically configured with the location of a DA, it can dynamically discover the presence of a DA through DA Advertisements, or it may actively solicit a DA Advertisement by requesting a directory agent service type at any point in time.

Figure 1 depicts interactions and a possible event sequence among the three agents after the DA multicasts its presence on a network and responds to SA Service Registrations and UA Service Requests.

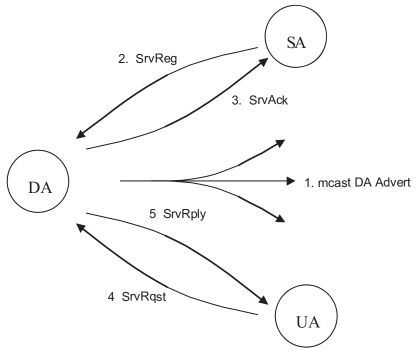


Figure 1. SLP Service Discovery

Apart from service requests, SLP also provides for the solicitations and replies for attributes using Attribute Requests and Replies. The operation works similar to requesting services on the network. Attribute querying is an optional feature.

III. Other Related IETF Work in SLP

Apart from the basic definitions of SLP, work was also done in addressing DA scalability using a mesh approach called mesh-SLP [8], where in a network with multiple DAs, SAs need only register with any and only a single DA discovered, with the DAs exchanging and synchronising all service registrations in the network amongst themselves. Additionally, an extension was defined for allowing notifications and subscriptions in SLP in which UAs can be explicitly notified whenever changes in service availability occurs [9]. Below, we discuss two of the most significant efforts in the IETF which concern our work. The first relates to SLP usage in IPv6, while the second describes SLP usage for remote service discovery.

A. Using SLP in IPv6

When SLP was extended for use over IPv6 [10], significant thought went into rectifying certain aspects of the protocol as used in IPv4. Usage in IPv6 was optimised to take advantage of a greatly expanded address space and address scoping as well as a slightly different multicast model from IPv4. These include:

1) Broadcast addresses and scoping rules

In IPv4, agents are able to broadcast service requests and solicit responses from the current subnet. In IPv6, such broadcast-only support is removed, as IPv6 itself uses link-local multicast in places of broadcast.

In IPv4, multicast service requests and advertisements can be bounded by administrative scoping rules, as well as the time-to-live (TTL) value of the datagram packets. In IPv6, the address prefix FF distinguishes a multicast packet from that of a unicast packet [11]. The scope of a service request or advertisement is determined by the next byte of the multicast address itself. Therefore the prefix FF0X is used in SLP for multicast and scoping, where the X is between 1 and 5 (1 for node-local, 2 for link-local and 5 for site-local). Because addresses in IPv6 can be represented both by the global address of the interface or its link-local address, services can be advertised with SLP using the link local address of the interface also. Consequently, careful usage is advocated in avoiding situations where a service with a link-local address location is advertised and obtained via multicast solicitations at the site-local scope.

SLP in IPv6 also introduces the concept of zones for multi-homed hosts, but as zones are out of scope of this paper, they are not discussed further.

2) *Multicast addresses for Service Request and Replies*

In IPv4, only one multicast address is used for all service requests and advertisements. In IPv6 however, a range of multicast addresses have been reserved for use with SLP. Using the scoping rules mentioned above, Service Type Request and Attribute Request messages are sent to FF0X::116. DA solicitations and advertisements are sent to FF0X::123. All other regular service request and advertisement messages are sent to an address which can range from FF0X::1:1000 to FF0X::1:13FF. The specific address is calculated by the UA and SA at runtime, by obtaining the result after applying a hash algorithm to the Service Type string to obtain a value between 0 and 1023. Beginning from the first address of FF0X::1:1000, this then establishes the offset to obtain the final multicast address. Compared to their IPv4 counterparts, SLP agents in IPv6 need to endure significantly less processing overhead owing to the fact that they join only the multicast groups pertinent to the service concerned.

B. *Remote Service Discovery in SLP*

A recent IETF standard describes how SLP can be used to perform service discovery in remote domains by conducting operations using an enhanced DNS infrastructure [12]. The focus is on how UAs can interact with the local DNS service to query for information about DAs in remote networks using SRV [13] resource records. This simple procedure is outlined below:

1. Let's assume there are two networks, A and B. A device from network A wishes to roam to network B.
2. Before services can be successfully discovered, network B must have at least one DA, which is registered with B's DNS server as an SRV record. For example, the DA could be registered as _slpda._udp.b.com for the target da.b.com.
3. Before the device moves from network A to network B, the UA on the device does a DNS lookup for the QNAME _slpda._udp.b.com and will obtain a DNS reply providing the location of the DA in network B as da.b.com
4. Subsequently, when the device moves into network B, the UA can interact with network B's DA as its location is now known, to perform service requests.

IV. Motivational Basis For Our SLP Extensions

Extending SLP for IPv6 as well as remote service discovery as explained in the previous section, provide an important basis for service discovery work in next generation IP networks. However, the primary use of SLP was still envisioned as service discovery for fairly fixed, enterprise network topologies, and the extensions were conceived at a time when IETF work with Mobile IPv6 was still ongoing. This mobility factor, together with the current general trend of portability and content sharing devices, imply that feature interactions among the various technologies have not been thoroughly investigated. This renders it difficult to use SLP in its current specification(s) for service discovery in advanced mobile networking.

In this section, these interactions are described in the following subsections below, leading to the motivation of the undertaken work as described in this paper.

A. *Moving from a client-server to a P2P paradigm*

One of the most radical paradigm shifts that appears set to occur in the mobile computing world is the move away from the traditional client-server model of communication, where a mobile device moves only as a client into new network spaces, needing only connectivity and communication with local servers.

From tangible products today, such as mobile devices carrying images, videos and documents with them, to research on location-aware peer-to-peer systems, we can observe that soon, it will become just as important for mobile devices to offer services in network vicinities the device roams into. SLP does not yet provide a mechanism for this; the remote service discovery described in Section IIIB has only been defined for DA discovery for incoming UAs trying to find services, not incoming SAs trying to register services. Clearly, a solution is needed to overcome this.

B. *Automatic and dynamic discovery*

In general, the mechanism of locating a specific resource, object or service can be performed in two ways [14], "Lookup" and "Discovery". "Lookup" refers to a passive process of locating a specific object, resource or service based on some matching criteria. It is initiated by a seeker, and requires the existence of some agent to answer the request. "Discovery" on the other hand, is used to refer to a more spontaneous process, in which many entities become aware of other entities on the network, and present themselves to other entities.

Based on the above definition, we can classify the remote service discovery mechanism in SLP as a lookup mechanism, owing to its reliance on a DNS server to perform DA lookups. Looking at the procedure outlined in section IIIB, it is clear to see that steps outlined can only occur in the simplest of scenarios, as it implies the device, while in Network A, already knows its next point of network attachment to be Network B before it roams there; what would be most likely is for movement from Network A to B to occur first, before the device queries Network B's DNS server for a DA.

In IPv4, addresses for mobile devices are generally configured via DHCP, where the DNS address is also supplied implicitly. Thus, the mechanism described above would be feasible. In IPv6 however, the preferred method of address configuration, for both fixed and mobile hosts, is stateless autoconfiguration [15]. This relies solely on router prefix advertisements in the device's point of network attachment. Consequently, DNS addresses are not supplied to the mobile device, implying that even if the foreign network does support this SLP extension, the mobile device would be unable to exploit it immediately. In the best case, the device, upon movement to network B, might try to query network A's DNS server, but network and firewall policies might prevent any meaningful exchange from taking place.

In order to successfully address this problem, a mechanism needs to be in place which actual discovery takes place in the new network, relying on the visited network's ability to provide broadcast or multicast mechanisms. This mechanism should also not impose any restrictions that may be placed on the mobile device or reliance on communication with its home network, such as having to relay on network A's DNS server to discover services in network B.

C. Rediscovery and re-registration of services

One of the most important factors affecting service discovery and device mobility is the rapid rediscovery of critical or everyday services in new networks upon movement. Currently UAs may cache service URLs obtained from SAs for a period of time corresponding to the lifetime defined by the SA. However, SLP has no provisions for service rediscovery, for UAs to automatically perform discovery of selected services whenever movement occurs. The same is also true for service registrations; SAs periodically advertise their services to the network (in the absence of a DA) or register their services to the DA before the service lifetime expires. Re-registrations upon movement to new network spaces are not provided.

We intend to augment this basic behaviour in SLP with a mechanism, whereby applications can notify their agents about services that need to be automatically rediscovered or reregistered upon movement.

D. Multiple addresses and service reacability

Extending SLP for use in Mobile IPv6 networks needs to address the challenge of dealing multiple source addresses and source address selection. When roaming, every mobile node (MN) will have two addresses: its permanent home address (HoA) and its temporary Care-of Address (CoA).

In cases where MNs have roamed into foreign networks and attempt to initiate unicast or multicast communication using UDP, source address selection procedures stipulate the HoA be used by default [16]. Accordingly, since the HoA is used as the source address, all packets are first tunnelled to the Home Agent (HA) in the home network before being subsequently forwarded on by the HA to their final intended destination. This would have an adverse effect in the case of a UA trying to perform active service discovery when the device it resides in roams into a foreign network. If the UA chooses to perform a Service Request at the site-local scope, the rules state that the HoA must be used. Consequently, the multicast Service Request packet will instead be sent to, and multicast only in its home network; agents in the foreign network would never see the Service Request.

To an extent, this problem is now being studied and addressed within the IETF, and procedures outlining how the source address selection algorithm can be influenced at the application level using a new IPv6 socket API for address selection [17] is being defined. The basic idea then could be that the SLP agents can choose to state that the usage of the CoA is always preferred over the HoA to avoid such situations. However, though promising, this work is still in its draft stages, currently with little implementation support if any.

Clearly, the way forward in this problem is to provide a mechanism to the SLP agents in mobile nodes informing them of newly acquired addresses, and allow them to use the CoA instead.

E. Multiple addresses and service advertisements

In addition to service requests, SLP service advertisements too can be adversely affected by IPv6 mobility. Consider the following scenario (fully qualified host names are used instead of numerical IPv6 addresses for clarity):

1. Mobile Node 1 has a permanent IPv6 address, HoA-of-mn1.NetworkA.com in its home network A, and it has an instant messaging client advertising its presence using an SA. So, the service URL advertised would be something like *service:im:jabber://HoA-of-mn1.NetworkA.com*, if we choose the Jabber chat protocol, just for the sake of an example.
2. MN1 then moves into a foreign network B, acquiring a temporary address, CoA-of-mn1.NetworkB.com. Since with Mobile IPv6, the IP layer shields movement, the SLP layer remains unaware of the new address.
3. Let's assume that Fixed Node 2 residing in network B, having the IPv6 address fn2.networkB.com is in the same subnet as MN1.
4. A chat application in FN2 wants to find other jabber chat clients in the same subnet for a P2P-like chat, and uses the UA in FN2 to issue a link-local Service Request for *service:im:jabber*.
5. Because MN1 is in the same subnet, it will directly respond to this request with a Service Reply, giving *service:im:jabber://HoA-of-mn1.NetworkA.com* as the service URL.
6. The messaging application in FN2 will attempt to connect to MN1's chat application using MN1's home address. MN1's Home Agent will capture this packet and attempt to tunnel it to MN1 in the foreign network.
7. At this point, one of two things will happen: Either the communication attempt will be successful, or Network B's

firewall policies will not allow incoming connections into the network and drop the packets. Thus, even though MN1 and FN2 reside in the same subnet and SLP communication succeeds, service reachability will be compromised, owing to the service advertisement using the HoA.

As proposed earlier in subsection D, providing SLP agents with information regarding the MN's CoA would alleviate the above issue to an extent.

However, the presence of multiple addresses, one permanent and the other temporary, also raises an interesting question of service persistence which SLP has not yet addressed: Does the application really want its service to be reachable permanently, regardless of its current point of attachment, or should the service be advertised so that it is reachable only in its current point of attachment, or both? These decisions reflect the desired persistence, availability and reachability of the advertised service and its address.

Therefore, any extension for SLP taking into account mobility should address the interaction between the application and the SLP agents, to accurately reflect the desired visibility of the service.

F. Movement Detection, Network Resource Awareness

When considering mobility and accessing local services, one of the foremost factors for consideration in application development is the provision of some basic method that allows for detecting device movement into different network spaces.

The type of information provided to the client application when the device moves may range from a very simple boolean trigger, to an encapsulated message containing very rich network information. The key idea in providing movement detection information is to allow it to be processed by any movement-sensitive applications resident in the mobile device reactively.

Therefore, closely related to all the issues in the previous subsections is the necessity of enhancing SLP functionality by providing movement detection to SLP agents which reside in mobile devices, regardless of whether Mobile IPv6 is used or not. Upon obtaining movement information, agents can choose to rediscover or readvertise services.

In conjunction with movement detection, we also wish to extend SLP to provide a notion of the underlying network's capabilities to support service discovery. With the current specifications, agents are provided with an idea of the network's capabilities only at startup, and cannot adapt to it at runtime.

When considering mobility, the link characteristics as well as the capabilities of the various networks to be visited can be vastly different. For SLP, support for multicast is of prime importance.

Hence, an agent must be able to cope roaming from a network supporting full native multicast, to one in which site-local multicast and above is unsupported, and still be able to successfully perform service discovery.

G. Compatibility and interactions with SLP extensions

Although many extensions to SLP have now been standardised by the IETF, all of them work invasively with the basic protocol; Usage of SRV resource records necessitates re-engineering UAs to understand DNS query-reply mechanism, SLP agents in IPv6 cannot interwork with SLP agents in IPv4 owing to a difference in the way multicast is used, mesh-SLP as well as service notification mechanisms all require UAs, SAs and DAs to again be re-engineered differently.

As previously mentioned, using these extensions in fairly fixed network topologies is rather straightforward, but could prove to become quickly unmanageable in mobile configurations. This is largely owing to the fact that there is no notification in SLP to an incoming mobile device to ascertain which of the abovementioned extensions are in place in the visited network. Hence, an agent optimised to work with one type of extension would fail to function properly when moving into a network supporting a different SLP extension.

Thus, the final target of this work is to ensure that our proposed new extensions remain as transparent as possible. Since the work to be done uses IPv6, we aspire to ensure communication with other IPv6 SLP agents is carried out without creating any additional incompatibilities. The new agents introduced should work non-invasively, but learn the SLP capabilities of the visited network. Other SLP agents (in IPv6) should continue to interact with the new extensions and agents developed without needing any changes or enhancements to their behaviour, protocol mechanisms nor message structures. From the perspective of the resident agents in any visited network, the incoming agents that use and advertise services would be treated without difference to any others already present.

V. New IPv6 Agents for SLP

In order to fulfil the objectives outlined in the previous section, we introduce two optional agents to SLP in IPv6. The first is a Visiting Agent, (VA) which resides in the mobile device itself. The second is an Access Agent (AA), which resides in the fixed networks, especially in network segments that form the initial point of entry for mobile devices visiting the network.

We first conceived the idea of using an AA and a VA in [18], which described a service discovery mechanism based on using SLP wirelessly over mobile IPv6. The work described provided a simple notion of movement detection and was designed to overcome the problem of using multicasting in foreign networks. However, as Mobile IPv6 protocol stack implementations matured and other standards became ratified, inadequacies in meeting the desired objectives were observed. Extensive refinements and testing was done, resulting in the work described in this paper.

The two agents are described in detail in the following subsections, beginning with the AA.

A. SLP Access Agent

The AA resides in the networks which the mobile node visits, periodically advertising its presence with link-local multicast AA Advertisements. AA Advertisements contain information that incoming mobile nodes with VAs can ascertain about the foreign network.

The general packet structure of each advertisement is described in Figure 2. Thin dotted lines indicate variable length fields, whereas fieldnames denoted in italics denote optional fields.

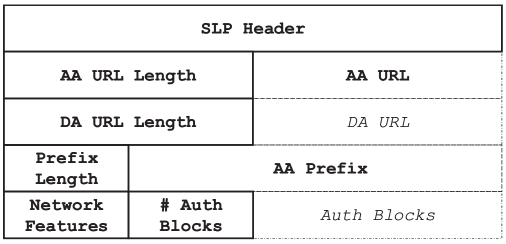


Figure 2. AA Advertisement Message

Each advertisement begins with a standard 4-byte SLP Header field. This is immediately followed by a 2-byte field describing the AA URL length as well as a variable length Service URL field of the AA. This URL would be of the form *service:access-agent://<address>*, where *<address>* is the globally unique IPv6 address of the Access Agent.

The advertisements are periodically sent to the link-local multicast address ff02::1:1259, the address being calculated using the standard hash algorithm for service announcements described in [10]. Also, AAs can respond to active solicitations for advertisements by VAs (or other UAs) in the network for the service type *service:access-agent*. Advertisements sent in response to solicitations are multicast into the subnet, instead of directly being sent as a unicast reply to the VA. This allows other VAs that might be present on the same subnet to receive AA Advertisements quicker in order to do movement detection. However, a minimum timing interval must be given to the AA, in order to prevent a flood of multicast packets from the AA into the network resulting from replies to all active solicitations in addition to periodic unsolicited advertisements. Should the AA receive an active solicitation before this timing threshold is exceeded, it must refrain from replying.

Access agents also listen for DA multicast advertisements, joining the well-known SRVLOC-DA group [10], ff05::123 at site-local scope and below. If at least one DA announces its presence, the location of the DA is included in the AA Advertisement. AAs also provide information about the IPv6 network prefix for the subnet they reside in. A 1-byte field then follows describing the capabilities and features in the network, with individual bits being set to reflect whether the network supports site-level multicasting, all other SLP extensions [8], [9], [12] that might be enabled in the present network, and if the AA is willing to serve as a multicast proxy for the VA. Both the network prefix as well as the subsequent 1-byte network capabilities field are configured from information provided to the AA by the site's administrator at startup. Should the network support IPv6 stateless autoconfiguration, the value of this prefix would be the default link prefix announced by the site's router.

SLP also defines its own optional authentication information that can be used to verify the authenticity of the message, and if used, these authentication blocks will form the last part of the structure of the AA Advertisement.

B. SLP Visiting Agent

Visiting Agents are designed to facilitate the process of service discovery as well as service provision for mobile client and server applications. Both service discovery and registration are performed using standard SLP messages: SrvRqst and SrvRply are used for discovery while SrvReg and SrvAck are used for registration. Visiting agents listen for AA Advertisements in the network segments they visit. By observing changes in both the source address of the advertisements and the announced IPv6 prefix contained within each AA Advertisement, VAs can discern whether they have moved into new network spaces.

However, it is also entirely possible some VAs can detect movement "out-of-band" in several other ways. These include the use of low-level packet capture or interface monitoring tools, information supplied from advanced applications capable of monitoring Mobile IPv6 routing headers [19], and application-driven prompting from a human user. In such cases, to reduce service latencies, the VA can actively solicit AA Advertisements to understand the capabilities of the new network from the received advertisements.

When AA Advertisements provide the location of a site-specific DA, it is obligatory for the VA to perform service discovery and service registrations using the DA. At the same time, the VA can also learn about the SLP characteristics of the new network: If mesh-SLP is used, the VA need only register any of its services with just one DA. Naturally, whenever movement occurs into a new network, the VA is also responsible for ensuring previous service registrations in the old network must be deregistered prior to re-registrations into the new network.

Alternatively, in the absence of a DA in the network, the AA Advertisement can also indicate to the incoming VA if the AA is willing to serve as a multicast proxy. If so, the VA sends its service requests and registrations to the AA via unicast. The AA subsequently multicasts the original service requests from the VA into the site-local network, relaying all replies it receives back to the VA. The AA also acknowledges any service registrations made by the VA, and listens for multicast service requests from any site-local agent requesting for the specific service. In case a service type matches, the AA sends a reply to the agent originating the request, containing the service URL that was originally registered by the VA.

Should the AA Advertisement not indicate the presence of a DA, nor the willingness of the AA to serve as a multicast proxy, the VA must check from the received advertisement if the visited network supports site-local multicast. If it does, in the case of a Mobile IPv6 node, the VA can use the CoA to issue and respond to site-local multicast SLP packets. The VA can choose to form its CoA, using its own link-local address and the network prefix provided by the AA Advertisement. Should the VA not choose to use the CoA, or if the network does not support site-local multicast, all SLP communication is limited to agents residing in the local subnet only.

Once the VA ascertains the necessary information, service rediscovery and re-registrations can be performed for mobile applications as described in the next two sections.

VI. Performing Automatic Service Rediscovery

In SLP, standard APIs in C and Java are defined for interaction between applications and SLP agents [20]. The function calls are designed to aid applications requesting services from UAs or registering services with SAs. For service discovery, the C function call that client applications invoke upon a standard UA is *SLPFindSrvs( )*. Among others, this function call is invoked with arguments that include a Service Type string, a callback function through which results are returned as well as a client-specific cookie value, which is passed back to the client's callback function upon returning results.

In addition to this function call, the VA also provides for automatic service rediscovery for client applications whenever the mobile terminal moves into a new network. Upon movement, service queries are reissued into the new network, either using the new DA, the AA or direct multicast. Clients are then notified of any new results. Table I provides the full binding for both C and Java. Both the original standard method calls as well as the additional automatic versions supported by the VA are shown.

TABLE I. Language Bindings for Service Discovery

	Standard Discovery	Auto-Discovery
C Binding	SLPError SLPFindSrvs(SLPHandle hSLP, const char *pcServiceType, const char *pcScopeList, const char *pcSearchFilter, SLPSrvURLCallback callback, void *pvCookie)	SLPError SLPAutoFindSrvs(SLPHandle hSLP, const char *pcServiceType, const char *pcScopeList, const char *pcSearchFilter, SLPSrvURLCallback callback, void *pvCookie, SLPBoolean findmode)
Java Binding For Interface Locator	public abstract ServiceLocationEnumeration findServices(ServiceType type, Vector scopes, String searchFilter) Throws ServiceLocationException	public abstract void autoFindServices(ServiceLocationEnumeration enumeration, ServiceType type, Vector scopes, String searchFilter, int findmode) Throws ServiceLocationException

The function call provided in C is *SLPAutoFindSrvs( )*. The signature for this function call is the same as for *SLPFindSrvs*, with the addition of a boolean field specifying a *findmode* value. If the function is invoked with *findmode* set to 1, auto-queries are initiated, with the client callback being invoked by the VA each time movement occurs and results need to be returned following an auto-query. If *findmode* is set to 0, the auto-discovery feature for this service type would be disabled, and subsequent movement into new networks would not trigger rediscovery.

VII. Service Re-registrations and Persistence

Similar to service auto-discovery, the VA also has provisions for automatic service re-registrations upon movement into new networks, for mobile applications wishing to be located by other network applications using SLP. In this case, the C function that mobile applications invoke upon the VA is *SLPAutoReg()*. Among others, this function call accepts a string containing a partially or fully formed URL (such as either the hostname, or a full service URL). It also accepts a second string specifying the service type, which could be combined with a partially specified URL to obtain a fully formed service URL. A callback function and a cookie are also supplied by the application and subsequently invoked by the VA upon operation completion to report status. The boolean value of 1 for *regmode* indicates to the VA that service re-registrations upon movement are desired in new visited networks (to either the local AA or a DA), whereas a value of 0 indicates that the application wishes an explicit deregistration of the service, with no new re-registrations when entering new networks. Table II below provides the full bindings in C and Java for service registrations that the VA supports.

TABLE II. Language Bindings for Service Registration

	Standard Registration	Re-Registration
C Binding	SLPError SLPReg(SLPHandle hSLP, const char *pcSrvURL, unsigned short usLifetime, const char *pcSrvType, const char *pcAttrs, SLPBoolean fresh, SLPRegReport callback, void *pvCookie)	SLPError SLPAutoReg(SLPHandle hSLP, const char *pcSrvURL, unsigned short usLifetime, const char *pcSrvType, const char *pcAttrs, SLPBoolean fresh, SLPRegReport callback, void *pvCookie, SLPBoolean regmode)
Java Binding For Interface Advertiser	public abstract void register(ServiceURL URL, Vector attributes) throws ServiceLocationException	public abstract void autoregister(ServiceURL URL, Vector attributes, int regmode) throws ServiceLocationException

When using Mobile IPv6, the notion of persistent and transient services are introduced for the VA to advertise in the various networks during movement. Persistent services are standard SLP services, registered by applications with the home address of the mobile node. To achieve this, the application provides the HoA as a partially formed service URL, and supplies the service type when using either *SLPReg* or *SLPAutoReg*. These registrations guarantee that the service remains available throughout the lifetime of the VA, transparently to any interested application regardless of the location of the mobile node, remaining true to the spirit of IP mobility.

Transient services, however, are services offered in the immediate localised surroundings, using the CoA of the mobile node. These services are only available for the duration of stay in the foreign network by the mobile node. However, applications may not have any idea what their current CoA is, nor indeed what future CoAs would be, whenever re-registrations are required. To overcome this, applications register the service by providing the string literal "*[coa]*" as the partially formed service URL instead of a hostname. The service type is supplied just as with persistent services. The VA then replaces "*[coa]*" with the current care-of address and forms the full service URL with the given service type. This URL is subsequently registered into the local network. Naturally, whenever the VA moves to a new network, it will need to unregister transient services from any DA or AA of the previous network and re-register them in the new network.

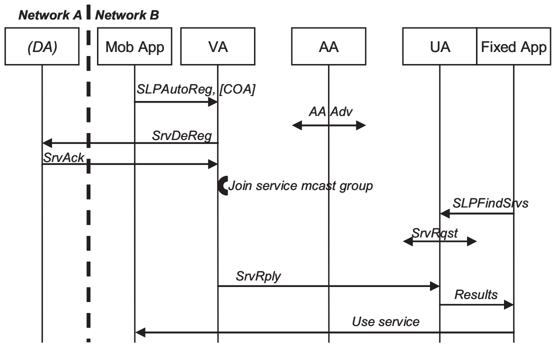


Figure 3. Transient service re-registration procedure when a mobile node roams from Foreign Network A back to its Home Network B.

This sequence of events is depicted in Figure 3, which shows a mobile application together with its VA moving from Network A to Network B. Let's assume that the mobile service to be autoregistered is a transient service, and that the VA has already previously registered it to a mesh-SLP DA, discovered via Network A's AA while the mobile node was in Network A. Upon movement into Network B, the VA ascertains via the new AA advertisement that no DA is present in Network B, and that full multicast is supported. Furthermore, by observing the AA prefix, it discovers the mobile node has returned back to its home network. Consequently it deregisters the service from the old DA, joins the service specific site-local IPv6 multicast group.and begins listening for multicast Service Requests. Once a service match is discovered, the VA replies, supplying the application's service URL with the CoA (which in this case will also be its HoA).

VIII. Conclusion

In this paper, two new SLP agents were described for service discovery with advanced mobile networking. Prototypes of VAs and AAs in C++ and Java on Linux and Windows laptops as well as Linux, Solaris and Mac OS X workstations were implemented, tested, and verified practically. Automatic service discovery and registrations were successfully accomplished and non-intrusive interworking with existing SLP agents (standard DA, UA and SA) was also achieved. IPv6 roaming with a portable Linux terminal running mobile applications and a VA was performed using four of our native IPv6 production and research networks. Experiments were successfully performed both when the portable terminal was Mobile IPv6 enabled, and without Mobile IPv6 support, with the terminal either acquiring a global IPv6 address through stateless auto-configuration, or simply using link-local communication with other IPv6 devices in an ad-hoc manner.

The work described in this paper, and subsequent design and implementation, fulfilled all the aims listed in section IV. Utilising SLP in this manner remains lightweight, running directly atop UDP and using the multicast characteristics of the network. It is unlikely that a single service discovery standard for advanced mobile networking would soon prevail, based on technical reasons or otherwise. However, the approach to overcome the challenges outlined in this paper can also be applied to various other popular service discovery mechanisms.

References

[1] J. Veizades, E. Guttman, C. Perkins and S.Kaplan, "Service Location Protocol," RFC 2165, June 1997.

[2] E. Guttman, C. Perkins, J. Veizades and M. Day, "Service Location Protocol, Version 2," RFC 2608, June 1999.

[3] The OpenSLP project, http://www.openslp.org

[4] M. Bakke, J. Hafner, J. Hufferd, K. Voruganti and M. Krueger, "Internet Small Computer Systems Interface (iSCSI) Naming and Discovery," RFC 3721, April 2004.

[5] EPI-19, "ACN Discovery on IP Networks," ANSI BSR E1.17, Entertainment Technology – Architecture for Control Networks, 14 March 2006.

[6] P. Goldsack et. al., "SmartFrog: Configuration and Automatic Ignition of Distributed Applications," Published in HP OVUA 2003, 29 May 2003.

[7] The LiveTribe project, http://www.livetribe.org/

[8] W. Zhao, H. Schulzrinne and E. Guttman, "Mesh-enhanced Service Location Protocol (mSLP)," RFC 3528, April 2003.

[9] J. Kempf and J. Goldschmidt, "Notification and Subscription for SLP," RFC 3082, March 2001.

[10] E. Guttman, "Service Location Protocol Modifications for IPv6," RFC 3111, May 2001.

[11] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 2373, July 1998.

[12] W. Zhao, et. al., "Remote Service Discovery in the Service Location Protocol (SLP) via DNS SRV," RFC 3832, July 2004.

[13] A. Gulbrandsen, P. Vixie and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)," RFC 2782, February 2000.

[14] R. E. McGrath "Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing," Presented at Center for Excellence in Space Data and Information Science, NASA Goddard Space Flight Center, April 5, 2000

[15] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC 2462, December 1998.

[16] Draves, R "Default Address Selection for Internet Protocol version 6," RFC 3584, Feb 2003.

[17] E. Nordmark, S. Chakrabarti and J. Laganier, "IPv6 Socket API for Address Selection," IETF work-in-progress, 5 March 2007.

[18] B. Silverajan, J. Kalliosalo and J. Harju, "A Service Discovery Model for Wireless and Mobile Terminals in IPv6," Proceedings of IFIP-TC6 8th International Conference on Personal Wireless Communications PWC 2003, Venice, Italy September 23 - 25, 2003.

[19] S. Chakrabarti and E. Nordmark, "Extension to Sockets API for Mobile IPv6," RFC 4584, July 2006.

[20] J. Kempf and E. Guttman, "An API for Service Location Protocol," RFC 2614, June 1999.

Publication V

Collaborative Cloud-based Management of Home Networks

B. Silverajan, J.P. Luoma, M. Vajaranta and R. Itäpuro

2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 786-789, 2015, IEEE.

DOI: 10.1109/INM.2015.7140376

Collaborative Cloud-based Management of Home Networks

Bilhanan Silverajan, Juha-Pekka Luoma, Markku Vajaranta, Riku Itäpuro
Tampere University of Technology, Finland
Email: firstname.lastname@tut.fi

***Abstract*—Future home networks are expected to become extremely sophisticated, yet only the most technically adept persons are equipped with skills to manage them. In this paper, we provide a novel solution as to how complex smart home networks can be collaboratively managed with the assistance of operators and third party experts. Our solution rests in separating the management and control functionalities of the home access points and routers, away from the actual connectivity, traffic forwarding and routing operations within the home network. By so doing, we present a novel REST-based architecture in which the management of the home network can be hosted in an entirely separate, external cloud-based infrastructure, which models the network within the home as a resource graph.**

***Index Terms*—Homenet, Network Management, Cloud, IoT.**

I. Introduction

Many residential home networks today consist of little more than a broadband-enabled WiFi access point and Network Address Translator (NAT), offering wireless and wired connectivity to any and all authenticated end-devices. If wireless coverage is spotty, wireless repeaters are used to cover blind spots. If application network performance is deemed to be laggy, more bandwidth is acquired from an Internet Service Provider (ISP). Granting connectivity to new devices or providing visitor access is typically accomplished by sharing a well-known password for wireless access. Therefore, apart from occasional activity, present practices require minimal management of the residential wireless home network by the home owner.

In recent years however, the home has rapidly become a natural convergence point for emerging technological developments and innovations. The smart home of tomorrow is expected to be an integral industrial and commercial testbed for the Internet of Things (IoT), Smart Cities and Grids, and even 5G connectivity. Connected homes are rapidly becoming a fertile ground for commercial vendors introducing their own ecosystems for home automation, health care and remote monitoring. This is widely anticipated to produce a rapid proliferation of computing, sensing and actuation systems in the home. Many of these systems also are integrated with home owners' personal mobile devices and into cloud-based platforms. At the same time, traditional Internet usage and connectivity usage scenarios would continue to play an important role.

Needless to say, these developments present severe strains onto any residential home network. To allow large numbers of sensors and smart devices to be connected and reachable via the Internet, network addresses need to be efficiently and properly allocated. Various members of a family may have different needs and connectivity priorities for their connected devices, while device vendors, utility and electricity providers may equip a smart home with their own products which may or may not be connected to the provider's network via the home network. A home network owner may also wish to segregate visitor network traffic away from critical portions of the network itself, giving rise to the need to perform network segmentation, traffic shaping and routing. Management of such a residential home network poses a few problems. Firstly, it is highly challenging for an average home owner to administer complex networks. Secondly, any technical assistance sought currently for management of access points and servers in the home relies upon the ability to remotely or physically access the customer premise equipment (CPE) which may not always be possible, owing to geographical issues or the presence of dynamic IP addresses, NATs as well as firewalls if other network routers or access points are positioned arbitrarily within the home network topology.

In this short paper, we address this challenge, as to how operators and third party experts can assist home owners in managing complex networks of the connected home in the future. Our focus is specifically on residential home networks that comply with the IETF Homenet standards and architecture. A brief description of Homenet is provided in Section II. We provide a novel solution of managing the home network in an external cloud using a browser based GUI or a REST-based interface, that integrates well with other REST-based and IoT services for added value to the home owner. The design and implementation of our solution are elaborated upon in Sections III and IV. With proper access control and authorisation methods in place, this provides a far more flexible and convenient solution for co-managed networks to be viewed by external parties, without having to manage individual home routers using traditional methods requiring remote access and local login credentials for each router. This is discussed in Section V. We then conclude this short paper in Section VI.

II. IETF Homenet

The Home Networking (Homenet) Working Group was chartered in 2011 by the Internet Engineering Task Force (IETF), in anticipation of the growing complexities of residential home networks, with the increase in number and demands by both connected computing devices and IoT-type constrained nodes. Homenet's intent is to research and standardise networking protocols and other mechanisms useful for residential home networks [1]. Although the properties and network topologies in a home network are not mandated, the home network is envisioned to grow large enough to require multiple network segments and subnets within the home, implying the existence of several routers which need to be orchestrated to perform actual routing using one more more well-known interior gateway routing protocols such as Babel [2], OSPF [3] or RIP [4]. Homenet supports both IPv6 and IPv4 address allocation mechanisms. Additionally Homenet advocates each node in the network to possess both a globally unique IPv6 address, as well as a local IPv6 address to avoid operational communications disruptions within the home, should an ISP uplink incur any downtime. Multiple ISP uplinks can be present in a Homenet-based network. Finally, a Homenet Control Protocol (HNCP) [5] is being specified, with which participating routers obtain information about the network capabilities, routing protocols and services present in the home network.

III. Design

Our approach has been inspired partly by Software Defined Networking (SDN) based principles, in which the network management and configuration functions of the home are reflected in the cloud and separated from the traffic flow and routing aspects of the home network. While common practices in SDN-based networks aim at real-time control of network elements, we choose to apply SDN concepts for network configuration management. Hence, direct control of packet flows and protocols such as OpenFlow are outside our scope. This is reflected in our architecture, as shown in Figure 1.

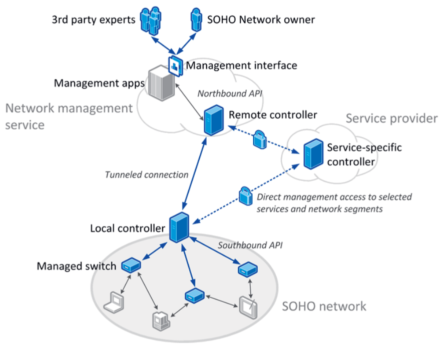


Figure 1: Proposed architecture

A cloud-based platform (henceforth referred to simply as a 'cloud') serves as a centralised network management and control platform to remotely manage home networks. This effectively behaves as a remote SDN controller, while having the network management service in the cloud provides a platform for the development and use for various kinds of operator tools, services and new network management apps, both native and web-based. The cloud interfaces with a trusted device that acts as a local controller, instead of directly communicating with the home network equipment. While the cloud provides a management interface and an effective means to deliver decisions to the home network, a local controller, possessed by the home owner, serves as a control point to execute decisions taken by the cloud-based controller, into the home network. To act as a local controller, a trusted device such as the owner's smartphone needs access to the Internet-based cloud service hosting the remote controller, and either a direct or tunnelled access to the home network. These connections need not be available at the same time if caching of configuration data is used by the smartphone.

If network configuration changes are made in the cloud, the smartphone can be made aware of any updates to the homenet configuration. Should the cloud provide a push-based notification service, the notification triggers management actions by the smartphone on the native management interfaces of homenet devices. If push notifications are not supported, REST-based polling by the smartphone can be used instead. Finally, the smartphone then connects to each element in the home network automatically to deliver the changed configuration. This is illustrated in Figure 2, and further elaborated upon in Section IV.

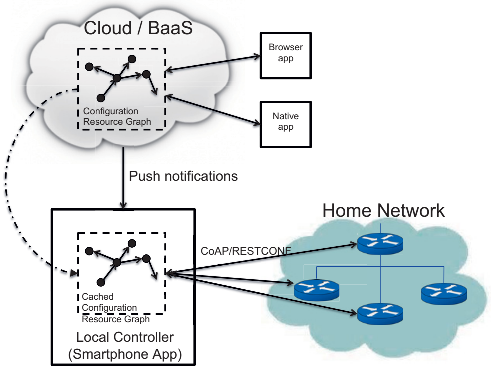
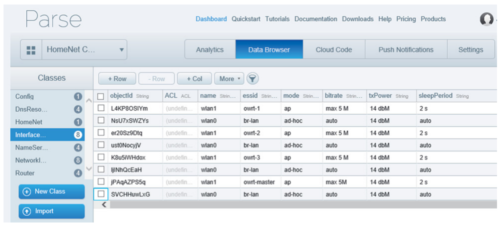

Cloud / BaaS
Configuration Resource Graph
Browser app
Native app
Push notifications
Home Network
CoAP/RESTCONF
Cached Configuration Resource Graph
Local Controller (Smartphone App)

Figure 2: Collaborative Management Design

IV. Implementation

Our test environment for prototyping our cloud-based collaborative network management solution consists of several portions. Firstly, we created an ISP capable of providing Internet connectivity to various home networks via IPv4 and IPv6. A DHCP server delivers a single IPv4 address to home border routers (as most ISPs do today), while IPv6 prefix delegation consisting of a /60 prefix is provided to supply the home network with global IPv6 addresses. Secondly, we then deployed a Homenet-compliant residential network with four wireless access points (consisting of TP-Link TL-WDR4300 and Buffalo WZR-HP-AG300H). The stock firmware was replaced with the latest OpenWRT snapshots from the trunk, based on Linux kernel version 3.10.49. The 2.4 GHz radio interfaces provided WiFi connectivity to client devices in the home, while all the 5 GHz radio interfaces were dedicated towards creating a wireless mesh network, in which the Babel routing protocol was utilised. This allowed for a resilient residential network where the network topology adjusted to favour routes with the strongest link characteristics, while remaining transparent towards the clients.



objectId String	ACL ACL	name Strin...	essid Strin...	mode Strin...	bitrate Strin...	txPower String	sleepPeriod String
L4KP8OSIYm	(undefin...	wlan1	owrt-1	ap	max 5 M	14 dbM	2 s
NsU7xSWZYs	(undefin...	wlan0	br-lan	ad-hoc	auto	14 dbM	auto
er20Sz9Dtq	(undefin...	wlan1	owrt-2	ap	max 5 M	14 dbM	2 s
ust0NocyjV	(undefin...	wlan0	br-lan	ad-hoc	auto	14 dbM	auto
K8u5iWHdox	(undefin...	wlan1	owrt-3	ap	max 5 M	14 dbM	2 s
ljINhQcEaH	(undefin...	wlan0	br-lan	ad-hoc	auto	14 dbM	auto
jPAqAZPS5q	(undefin...	wlan1	owrt-master	ap	max 5M	14 dbM	2 s
SVCHHuwLxG	(undefin...	wlan0	br-lan	ad-hoc	auto	14 dbM	auto

Figure 3: Parse Cloud-based platform

To manage the home network, we used Parse [6], a popular Backend-as-a-Service (BaaS) cloud platform, to host the remote controller. The configuration parameters of home network devices are represented as data objects on Parse as depicted in Figure 3. Parse provided several features useful for our prototype, such as a cloud-based data store, user management, user role based access control, push notifications, and support for cloud-hosted code. Our prototype uses Parse JavaScript SDK to provide a web GUI for the users of the cloud-based management service. This GUI allows users to add and remove managed devices, view and set configuration parameters of a network device, and change the set of configuration parameters currently used by all network devices.

Figure 4 shows our design for storing versioned configuration data in Parse. Several named configurations of the home network devices can be stored in the cloud, maintaining current as well as previous versions of configurations. Each configuration comprises the parameters of all home network devices being managed. When there are changes to a configuration and the changes have been committed, a new version of the configuration is created. The set of previous versions of each configuration allows reverting to a previous version of the network configuration if needed. New named configurations can be created as needed, either using a previously stored configuration as a template or by taking a snapshot of the current state of home network device configurations.

Parse also supports a REST based API that could be used for providing access to parts of the Homenet configuration by Internet-based automation services such as If This Then That (IFTTT) [7]. This provides the ability to build management apps that can configure and manage the home network via Parse, based on policy or context-based events triggered from IFTTT recipes (such as powering down non-essential radio interfaces based on time, user presence, or power savings profiles). An initial prototype of the local controller implemented in Node.js is currently running on a Linux PC and

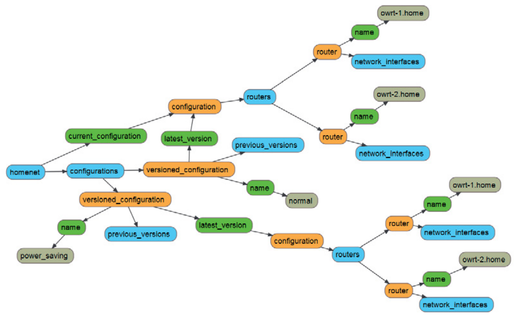


Figure 4: Versioned configuration data

uses the Parse JavaScript SDK to access the home network configuration data stored in the cloud. It then executes management commands on each router over SSH. Work is being undertaken to instead use an Android-based smartphone as the local controller that receives push notifications from Parse. The phone then uses REST-based method calls using CoAP [8], or RESTCONF [9]. CoAP is the basis for device management in the OMA Lightweight M2M [10] specifications, while RESTCONF is a RESTful approach towards using NETCONF [11] to manage the access points.

V. Security Issues

To allow collaborative management of the home network by the different stakeholders, it should be possible to define access rights to different subsets of network configuration parameters for groups of users according to their user role. The basic rule we wish to adhere to is to ensure that all modifications to the network are done via the cloud, and that executing these changes to the residential network via the local controller can only be rendered possible with explicit authorisation of the transaction by the home network owner. To accomplish this, we rely on a two-phase authentication and access control security model which are closely tied with each other.

In the first phase, authentication and access control to the data in the cloud is addressed. In this phase, the owner of the network would be free to assign roles and time-based access to various types of apps (or providers and users) accessing the cloud-based data. This allows trusted users to read and write the resource graph of data objects representing the configuration parameters of home network devices. Different views and access rights to the resource graph can be provided on a continuous or time-limited basis according to arbitrarily defined user roles. Certain roles, such as the primary Homenet user, could have full read-write access to all configuration parameters, while other users may be limited to read-only access or have no visibility to parts of the network configuration at all. An external service-specific controller such as an Internet-based automation service could also be provided access to selected parts of the network configuration via the cloud.

Once a valid management operation is performed in the cloud, the local controller is notified, and successful execution of the operation is achieved by the local controller onto the residential access points upon successfully completing the second phase of authentication and access control. For this phase, an AAA-based mechanism is employed, in which the RADIUS protocol [12] is heavily used. Each home contains a RADIUS server capable of authenticating local users against provided credentials such as passwords, for obtaining network connectivity from the home network. However, when the local controller receives a request to update network settings, it can escalate its privileges from obtaining connectivity, towards performing network administration, by supplying credentials provided to it from another federated RADIUS-based realm, such as a security service provider (which could be the network operator, or a third party service provider). Successfully authenticating against these credentials would allow the local controller to be recognised as a valid executor. In our implementation, our initial strategy for the local controller has been to obtain an assurance of its identity by the security service provider, which informs the home RADIUS server what the device being authenticated is allowed to do.

VI. Conclusions and future work

In this short paper, we presented our active work-in-progress architecture, design, and implementation experiences in providing a new solution to allow expert external assistance to co-manage residential home networks of the future, without the need to have end-to-end network or physical connectivity to each element that needs to be configured and controlled in a home. Our initial tests and results appear promising as valid solutions of using a cloud-based controller for the management of, among others, Homenet-compliant home networks, without breaking any compatibilities. The connected home is becoming the technological focal point for intelligent control and communications systems, as well as consumer and personal electronics and advanced sensoring and actuating components. However, it is commonly assumed that it is the sole responsibility of the home owner to manage his network, which is a daunting prospect. Our solution offers a clear separation of concerns for the multiple stakeholders interested in making the smart home a success: The owner, the members of his family, the network operator, the cloud service providers and various third party experts. In so doing, new business models and revenue streams are created, while allowing service providers access and management capabilities to govern any devices they own residing within the home.

Apart from obtaining expert help, there are several other advantages to our approach. For example, a home network operational state and properties of the routing and switching elements can be preserved easily in the cloud. As residential access points tend to be commodity, cost-effective equipment, this approach allows easy rectification and replacement of defective routers, by restoring existing network configurations into new equipment. Also, upgrading network equipment in the home to take advantage of new technologies can be performed without much consternation.

Communication between the cloud and the local controller is completely REST-based. Uplink disruptions, as well as disruptions between the local controller and the routers, do not affect the actual operation of the home network. Once the uplink resumes, the local controller synchronises any changes or new policies with the cloud and applies them to the home network. In future, we aim to deploy RESTCONF-based communication between the local controller (i.e. smartphone) and the residential network elements. This allows easy interworking and integration with the Web of Things, allowing the formulation of intelligent decisions and sophisticated policies by obtaining contextual and geophysically relevant data from external web-based data sources. This results in fine-tuned policies for network performance towards various energy profiles, bandwidth control, data aggregation and traffic routing.

The architecture also considers proper authentication and role-based as well as time-limited access control as an important facet. While we are using RADIUS-based authentication and access control using passwords, in the future we aim at investigating SIM-based authentication solutions [13] that could identify the roles a owner's smartphone can fulfill.

Finally, the solutions proposed and studied in our research are highly scalable, allowing not only management of home routers, but also other types of constrained IoT-like nodes such as sensors and smart consumer appliances that allow REST-based resource retrieval and manipulation.

VII. Acknowledgements

This work is funded by the Finnish Digile IoT Programme.

References

[1] T. Chown, Ed., J. Arkko, A. Brandt, O. Troan and J. Weil, "IPv6 Home Networking Architecture", IETF Internet draft, work in progress, July 4, 2014; http://tools.ietf.org/html/draft-ietf-homenet-arch-17.

[2] J. Chroboczek, The Babel Routing Protocol, IETF RFC 6126, Apr. 2011; http://tools.ietf.org/html/rfc6126.

[3] R. Coltun, D. Ferguson, J. Moy and A. Lindem, Ed., OSPF for IPv6, IETF RFC 5340, July 2008; http://tools.ietf.org/html/rfc5340.

[4] G. Malkin, RIP Version 2, IETF RFC 4822, Nov. 1998; http://tools.ietf.org/html/rfc2453.

[5] M. Stenberg and S. Barth, "Home Networking Control Protocol", IETF Internet draft, work in progress, June 25, 2014; http://tools.ietf.org/html/draft-ietf-homenet-hncp-01.

[6] "Parse - The complete mobile application plaform", Oct. 5, 2014; http://parse.com.

[7] "IFTTT: Put the Internet to work for you", Oct. 5, 2014; http://ifttt.com.

[8] Z. Shelby, K. Hartke and C. Bormann, The Constrained Application Protocol (CoAP), IETF RFC 7252, June 2014; http://tools.ietf.org/html/rfc7252.

[9] A. Bierman, M. Bjorklund, K. Watsen and R. Fernando, "RESTCONF Protocol", IETF Internet draft, work in progress, Feb. 13, 2014; http://www.ietf.org/archive/id/draft-bierman-netconf-restconf-04.txt.

[10] "Lightweight Machine to Machine Technical Specification", Candidate Version 1.0 - 10 Dec 2013, Open Mobile Alliance, OMA-TS-LightweightM2MV1_0-20131210-C; http://www.openmobilealliance.org.

[11] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., Network Configuration Protocol (NETCONF), IETF RFC 6241, June 2011; http://tools.ietf.org/html/rfc6241.

[12] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote authentication dial in user service (RADIUS), IETF RFC 2865, Jun. 2000; http://tools.ietf.org/html/rfc2865.

[13] H. Haverinen, Ed. and J. Salowey, Ed., Extensible Authentication

Publication VI

Enhancing Lightweight M2M Operations for Managing IoT Gateways

B. Silverajan, M. Ocak, J. Jiménez, and A. Kolehmainen

2016 IEEE International Conference on Internet of Things (iThings 2016), pp. 187-192, 2016, IEEE.

DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.55

Enhancing Lightweight M2M Operations for Managing IoT Gateways

Bilhanan Silverajan*, Mert Ocak†, Jaime Jiménez†, Antti Kolehmainen*
*Tampere University of Technology, Finland
{bilhanan.silverajan, antti.kolehmainen}@tut.fi
†Ericsson Research, Finland
{mert.ocak, jaime.jimenez}@ericsson.com

***Abstract*—Heterogeneity in constrained networks and gateways is perhaps one of the single greatest challenges facing end-to-end management of devices and networks in the IoT. Today, the Lightweight M2M (LWM2M) protocol, leveraged on open Internet standards, has become a strong contender for REST-based IoT management. However, significant challenges exist for remote gateway management, particularly for proxies and exposing proprietary or vendor-specific operations. This paper presents some effective solutions in LWM2M for overcoming these challenges, using object modelling and linking. The presented approaches are also validated with prototypes.**

***Index Terms*—LWM2M, IPSO, CoAP, Gateway Management**

I. Introduction

Several major Standards Development Organisations (SDOs) have undertaken considerable effort in establishing protocols, architectures and communication guidelines in the Internet of Things (IoT) domain. The recent standardisation of the Constrained Application Protocol (CoAP) [1] by the Internet Engineering Task Force (IETF) was an important milestone. It effectively extended the Representational State Transfer (REST) paradigm, upon which the web API model is predominantly based today, towards constrained IoT devices and smart objects. CoAP also is becoming the foundation for REST-based device management in IoT. The Open Mobile Alliance (OMA) developed the Lightweight M2M (LWM2M) architecture [2] around CoAP, specifying guidelines, interfaces and data models for managing devices, sensors and actuators. A similar strategy was adopted by the IPSO Alliance [3], which augmented LWM2M with semantically interoperable object models as building blocks which can be composed together to describe more complex smart objects.

Because the growth of the IoT is fairly organic, there would always be a need to integrate and interconnect a variety of existing and deployed smart objects, many of which use legacy and non-IP communication using different kinds of gateways. Even with natively IP-enabled smart objects in the future, connecting billions of devices in the IoT would encompass multiple Layer 2 and Layer 3 solutions. This would require IoT gateways to not only supply connectivity and routing, but also behave as protocol translators or application level proxies.

We characterise an IoT gateway according to its functionality. It can supply wireless IP connectivity to devices and sensors by bridging two kinds of Layer 3 networks with fixed or cellular backhaul links. It can proxy between non-IP networks and the Internet. It can facilitate endpoint reachability for management operations in the presence of firewalls and NATs using packet encapsulation or tunnelling. It can finally provide application-level architectural interoperability among existing management tools and systems.

Other ways of categorising IoT gateways in terms of the computational power, form factor or energy consumption patterns are also possible. Quite often, gateways in IoT can fulfill multiple roles. Many IoT gateways used in constrained environments also possess similar properties as IoT devices and smart objects in terms of hardware heterogeneity, computational ability, battery lifetimes or remote management. Consequently, as long as any potential limitations are addressed, managing IoT gateways using IoT device management protocols and architectures would, on the surface, appear to be a logical step for the configuration, control, and lifecycle management procedures. Even in situations where the gateways themselves are not constrained or are more computationally powerful, there are significant advantages in terms of semantic interoperability and evolution, when a management server can adopt the same methods in managing an IoT device as well as the gateways along the network path. This forms the basis of our work described here.

In this paper, we apply our experience gained with LWM2M-based management and standardisation activities in IPSO towards utilising LWM2M and IPSO data models for effectively managing IoT gateways. While in this section, we showed the importance of gateways and gateway management in the IoT, section II provides an overview of the LWM2M architecture and IPSO Objects. Section III offers a deeper insight into the major challenges of using LWM2M for gateway management particularly with exposing dynamic operations as well as proprietary and non-IP data models. Section IV offers some novel solutions for overcoming these limitations, by using new object models as well as web linking and the Information Reporting Interface of LWM2M. To add further clarification in this paper, we present a simple example of how this can be achieved in practice, by demonstrating an example energy management scenario in Section V. Findings that validate our work are discussed in Section VI before conclusions are presented in Section VII. Related work is discussed throughout the paper as concepts are presented.

II. LWM2M Architecture and IPSO Objects

LWM2M provides a light and compact secure communication interface along with an efficient data model, which together enables device management and service enablement for M2M devices [5]. The architecture defines a client-server communication model. A LWM2M Server is typically located in a private or public data centre and can be hosted by the M2M service provider to manage LWM2M clients. The LWM2M Client resides on the device and is typically integrated as a software library or a built-in function of a module or device. An optional Bootstrap Server can be used to manage the initial configuration parameters of LWM2M Clients during bootstrapping the device.

Management operations between the server and client are grouped logically into four interfaces: The Bootstrap Interface, the Device Discovery and Registration Interface, the Device Management and Service Enablement Interface, and the Information Reporting Interface. As mentioned previously, LWM2M uses CoAP for communication as the underlying transport protocol. CoAP itself has similar method calls and response codes as the Hypertext Transfer Protocol (HTTP). LWM2M's management operations map onto CoAP method calls and response codes. CoAP resources are also identified using Uniform Resource Identifiers (URIs). LWM2M introduces a data model where LWM2M resources are logically collected into LWM2M Objects. This is achieved by introducing a simple hierarchical object model, in which objects represent sensor or actuator types, with a list of resources representing the properties specific to that object type.

The OMA also maintains a naming authority called the Open Mobile Naming Authority (OMNA). The OMNA operates an Object and Resource Registry with which objects and resources can be standardised with a name, description, types and so on. An unambiguous identifier is also assigned. For on-the-wire efficiency, this ID is used instead of names, during CoAP operations. Multiple objects of the same type can be differentiated by their Instance ID. This allows objects and resources to be mapped in a standard way into CoAP resources and URI path components, in the form of `/<object ID>/<object instance ID>/<resource ID>`.

It is common that at startup, each LWM2M Client registers several distinct objects to a LWM2M Server. Upon registration, the Server assigns a unique endpoint ID to each Client. After registration, the Server is further able to differentiate various endpoints by prepending the endpoint ID to the object and resource identifiers. Subsequently, the Server is able to perform read, write and execute management operations on Client objects and resources using standard CoAP method calls and URIs. Such a CoAP URI might be `coap://lwm2m.example.org/123/3/0/2` which retrieves the serial number (Resource ID 2) of object instance 0 of the LWM2M Device object located at the LWM2M endpoint hosted on lwm2m.example.org having an endpoint ID of 123.

In addition to the OMA, the IPSO Alliance also defines smart objects and object models. IPSO Objects are defined in such a way that they do not depend on the use of CoAP, and any RESTful protocol is sufficient. Nevertheless, to develop a complete and interoperable solution the IPSO Object model is based on LWM2M specification and object model [6]. While LWM2M uses objects with fixed mandatory resources, IPSO Objects use a more reusable design. As devices increase in complexity, IPSO specifies how to compose sophisticated Objects by using simpler Objects as building blocks and linking them with specific resources called Object Links. Object Links point to other objects within the Client and are expressed as two concatenated integers separated by a colon, such as 3304:2, where the first integer represents the Object ID and the second the Object Instance. IPSO Objects are also registered with the OMNA.

III. Considerations of Gateway Management in IoT

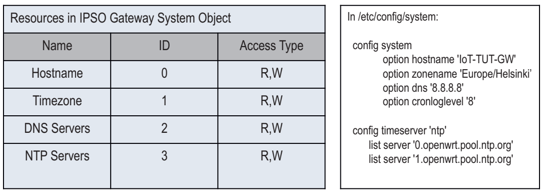
A study undertaken by the Internet Architecture Board (IAB) to understand the impact of IoT on existing Internet infrastructure led to the publication of an informational standard on architectural considerations for smart object networking [4]. One of the findings outlined is the device-to-gateway communication pattern, which is an architectural pattern connecting proprietary and non-IP nodes to the Internet using gateways. While the pattern considers reachability and end-to-end communication, such a communication pattern also impacts gateway management, operations and the management of non-IP endpoints. These are described in the subsections below.

A. *Integrating non-IP and proprietary data models*

The use of LWM2M and IPSO data models between the LWM2M server and managed endpoints requires both reachability and end-to-end IP connectivity. The specifications do not cater for the management of non-IP end-points which use short range wireless radios such as Bluetooth Low Energy (BLE), Z-Wave or Zigbee.

One approach towards managing non-IP nodes is to masquerade the nodes' properties in the gateway itself. Should a gateway be used to bridge these networks into the IoT, it can also translate between the kinds of data models expected by the LWM2M architecture and the proprietary data models used by the end-points. Integrating such proprietary models to the network requires the gateway to serve as an intermediary on behalf of these end-points in terms of registration and management operations. Often this can be an effective solution for exposing properties as a proxy to the LWM2M server in a RESTful way as standard LWM2M or IPSO Objects. This is highly effective for managing very resource-constrained endpoints, providing semantic interoperability for connected end devices. It can also be used to expose proprietary gateway models for configuration and control.

As an example, the Unified Configuration Interface (UCI) [7] of the well-known OpenWRT Linux platform for gateways organises configuration information as various files with well-defined names in */etc/config/*, such as *network*, *firewall*,



Resources in IPSO Gateway System Object		
Name	ID	Access Type
Hostname	0	R,W
Timezone	1	R,W
DNS Servers	2	R,W
NTP Servers	3	R,W

```
In /etc/config/system:

config system
        option hostname 'IoT-TUT-GW'
        option zonename 'Europe/Helsinki'
        option dns '8.8.8.8'
        option cronloglevel '8'

config timeserver 'ntp'
    list server '0.openwrt.pool.ntp.org'
    list server '1.openwrt.pool.ntp.org'
```

Fig. 1. Excerpts showing mapping between IPSO and UCI system data models

system and *wireless*. Reading these files would reveal that UCI internally models configuration-specific data into well-defined sections called *zones* with nested properties called *options*. Correlation between the nested UCI configuration data and LWM2M or IPSO hierarchical data models can therefore be performed. Figure 1 depicts this correlation between the system configuration data used by UCI and the IPSO Gateway System Object. The IPSO Gateway System Object can be used directly with LWM2M, although, at the time of this writing, its Object ID as well as URN are still subject to change and final assignment by IPSO.

A similar strategy can be employed to model the properties of non-IP devices and sensors. In BLE, data exchanged through the Generic Attribute Profile (GATT) is organised hierarchically into *Services* and *Characteristics*. A single service is a logically grouped collection of characteristics, while characteristics contain type, values and properties such as supported operations. GATT services and characteristics are therefore very similar to LWM2M Objects and Resources. Consequently, an intermediate gateway can be used to manage BLE endpoints in LWM2M using this approach. Integrating such proprietary models to the network requires the gateway to translate between the data models.

However, such translation is done using proprietary methods in most of the current gateway implementations and hence, creates silos between different gateway manufacturers. As an example, Bluetooth SIG publishes GAP and GATT REST API white papers [8] [9] to standardize the APIs defined on Bluetooth gateways but common data models to seamlessly integrate the Web and BLE data models are needed to provide interoperability between these two technologies. The BIPSO project, on the other hand, aims at harmonising BLE data models with those of IPSO, by providing a standard mapping between IPSO Objects to BLE Characteristics with well-defined Characteristic Values [10].

The LWM2M architecture currently does not also possess adequate semantics for describing such proxying, as while CoAP itself supports proxies via the *Proxy-URI* option, how this can be achieved and defined in LWM2M remains open. Consequently, while the gateway can integrate and host data models on behalf of proprietary or non-IP endpoints, it needs to masquerade the objects and resources of each endpoint on its LAN as its own. In other words, the management server remains unaware that such Objects refer to physically distinct endpoints external to the gateway. Ideally, discovering and using proxy functionality should be incorporated into future data models which unambiguously assert the location and type of managed endpoints residing behind such a gateway. This way, the runtime interaction would become expressive enough to cleanly separate end-to-end management from any underlying connectivity issues.

B. Exposing proprietary and gateway-specific operations

IoT device management rarely takes into account non-atomic, transaction-based or system-specific method calls which are inherently present in many operations related to gateway configuration. Often, these calls also differ from vendor to vendor. While the LWM2M data and operation models support read, write and execute operations on a resource (effectively a CoAP GET, PUT or POST operation), a gateway's proprietary or native configuration API often needs to be invoked upon a change in a resource's representation. This is effectively an implicit process not exposed to the management server as the object model does not attach any semantics to the kinds of native operations that could or should be invoked upon actions on Objects and resources. When applied to gateway management this can become an issue, as the object model does not accurately capture the kinds of operations and configuration changes a gateway may need to perform in the background, to properly reflect the resource state in the LWM2M object model.

As an example, consider a packet filtering mechanism on an access point, for which an example LWM2M or IPSO object model comprises of firewall objects. Each firewall object represents a single rule, while each resource represents an option, such as a name, source or destination address, incoming interface, connection state. In order to first properly construct and then trigger the firewall rule on the gateway as well as to subsequently monitor its status, each action on the object to be taken needs to be modeled as a resource. Moreover, an additional final commit resource is needed to trigger the firewall rule to make the changes effective. Without any additional semantics, similar kinds of object and resource models need to be developed for other policy objects, system configuration and the gateway's network interfaces. A different gateway implementation may also choose to perform its runtime firewall management in a different manner, and incorporate additional resources to reflect these operations in a static manner within a different object model. In the long run, this leads to untenable and fragmented Object models with poor interoperability. Clearly, a better solution would be to retain a core reusable Object model representing the properties of a gateway or an endpoint being proxied by the gateway, and conceive a mechanism which exposes system-specific operations dynamically at runtime.

In HTTP-based REST models, resources can be dynamically supplied from a server to a client with hypermedia by embedding a REST constraint called Hypermedia As The Engine of Application State (HATEOAS) [11]. Using HATEOAS, an origin server only exposes resources directly necessary for the current context, while returning possible operations on the object or resource as hyperlinks and relation types implementing standard REST verbs. While using HATEOAS increases the messages exchanged between a client and a server, it allows the responses from the server to the client to be more transparent, and allows run-time discovery of allowed operations on the server by a client.

If HATEOAS can be used, the firewall object can be simplified so that the gateway's native API invocations are exposed to a management server over a REST interface when necessary. A commit hyperlink could be exposed under a name resource if and only if the firewall object has yet to be deployed, for example. Hypermedia-based APIs, content types and HATEOAS, however are currently unsupported by LWM2M.

IV. Strategies for IoT Gateway Management

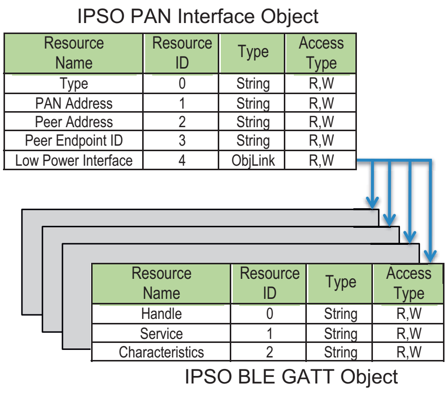
In order to overcome the challenges outlined in Section III, our solutions were inspired by a technique employed by IPSO. Recognising that attempting to define complex devices as discrete Objects can easily result in large and unsustainable models, IPSO deliberately defines collections of Objects as simple building blocks. These are then aggregated by developers to construct Composite Objects using Web Linking [12]. Such a Composite Object would have resource lists similar to simple Objects. However they would also contain Resources whose types are defined as LWM2M Object links. These would then contain reference URIs which are mapped to other Object Instances assembled to create the Composite Object Model, as outlined in section 4 of [6].

A. Semantics for Proxy Management

The issues described in Section III-A can be overcome using a similar approach as IPSO by using Web Linking. This requires the Object model of a managed non-IP endpoint to furnish enough semantics to the LWM2M Server which describes and identifies the gateway being used as a proxy. For example, we have drafted 2 IPSO Objects which aim at working with low-powered non-IP nodes, particularly for BLE. These are depicted in Fig 2 as the Personal Area Network (PAN) Interface Object, and the BLE Generic Attribute Profile (GATT) Object.

The PAN Interface Object is a generic Composite Object that can be used for modelling multiple PAN interfaces of either an endpoint or a gateway. For each interface, the *Type* Resource specifies the kind of PAN this interface represents, such as BLE, Zigbee, Classic Bluetooth and so on.

The address associated with the interface is also expressed, followed by 2 Resources representing the properties of its communicating peer. For an endpoint, these would represent the gateway's PAN address as well as the LWM2M Endpoint ID supplied by the Server, while for a gateway, multiple instances of the *Peer Address* Resource would represent managed end-points, and optionally, any Endpoint ID. In the case of a BLE-based endpoint, multiple instances of the *Low Power Interface* Resource would be linked to multiple instances of the


IPSO PAN Interface Object

Resource Name	Resource ID	Type	Access Type
Type	0	String	R,W
PAN Address	1	String	R,W
Peer Address	2	String	R,W
Peer Endpoint ID	3	String	R,W
Low Power Interface	4	ObjLink	R,W

Resource Name	Resource ID	Type	Access Type
Handle	0	String	R,W
Service	1	String	R,W
Characteristics	2	String	R,W

IPSO BLE GATT Object

Fig. 2. IPSO PAN and BLE GATT objects.

BLE GATT Object, with each BLE GATT Object representing a single BLE service.

Describing managed endpoints via a proxy in this manner supplies an LWM2M Server with enough structured information to correlate as well as understand the kinds of non-IP endpoints being managed and through which gateways communication is possible.

B. Interaction model with dynamic linking

Gateways often have system-specific operations that need to be performed during management. Section III-B outlined the current shortcomings that prevent LWM2M Servers from understanding and invoking them. However, just as with proprietary data models from the previous section, a simple but effective technique using Object links allows an LWM2M Server to be aware of contextual operation invocation on the gateway, based on the current representational state of the gateway's objects and resources. Additionally, the LWM2M Information Reporting Interface is relied upon as a notification mechanism at runtime to dynamically supply information of gateway-specific operations to the LWM2M Server. This notification mechanism is based on CoAP Observe [13], which allows an observer to register an interest on an observable CoAP resource, so that when the representational state of the resource changes, the observer is notified.

Classic LWM2M and IPSO data model depict Resources and Objects in a fairly static structure. To this, another class of gateway objects need to be introduced, which we term Interaction Objects. Interaction Objects contain a list of executable Resources, each of which represents a system-specific operation required by the gateway, in order to effect runtime changes when requested by the LWM2M Server. Such an instance of the Interaction Object could be linked to a specific Gateway Object Instance together with another Interaction Object having observable Resources, which computes the state of the Object Instance and updates its Resource values to supply the object link to the correct Interaction Object Instance

(A) IPSO Operations Object

Resource Name	Resource ID	Type	Access Type	Observ-able?
Op List	0	ObjLink	R	Yes
Active Op	1	Integer	R	Yes

(B) IPSO BLE Commands Object

Resource Name	Resource ID	Type	Access Type
HCI_TOOL	0	String	X
HCI_CONFIG	1	String	X
GATT_TOOL	2	String	X

(C) IPSO UCI Commands Object

Resource Name	Resource ID	Type	Access Type
UCI_GET	0	String	X
UCI_SET	1	String	X
UCI_COMMIT	2	String	X

Fig. 3. IPSO Interaction Objects.

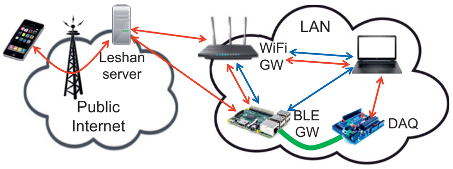


Fig. 4. Experimental setup. Red arrows indicate IP communication, while blue arrows depict BLE communication. Green line indicates direct physical connection to monitor power.

TABLE I
IPSO WIRELESS GATEWAY INTERFACE OBJECT

Resource Name	Resource ID	Type	Access Type
Ifname	0	String	R,W
Mode	1	String	R,W
Disabled	2	Boolean	R,W
SSID	3	String	R,W
MAC Address	4	String	R,W
IP Address List	5	String	R,W
Transmission Power	6	Integer	R,W
Network	7	String	R,W

and the Resource ID representing the system-specific operation to be invoked.

Fig 3 illustrates three draft IPSO Objects that, when composed together with an IPSO or LWM2M Gateway object, can accurately provide the management server with hypermedia-like operations on the gateway. For example, assume that the gateway registers an IPSO Wireless Interface Gateway Object, composited together with the IPSO Operations Interaction Object as well as the IPSO UCI Commands Interaction Object. The Wireless Interface Gateway Object is shown in Table I. At registration, the *OP List* Resource in the Operations Object would point to the UCI Commands Object Instance registered by the gateway, while the *Active OP* Resource value would reflect the currently active operation. In this particular case, the value would be 0, to reflect that the currently active operation would be the *UCI_GET* executable resource. If the LWM2M Server, at some future point in time, choose to disable the Wireless Interface Object by setting the value of the *Disabled* Resource to "1", the value of *Active OP* would be changed by the gateway to "1", reflecting that the next active operation on the Wireless Interface Gateway Object would be the *UCI_SET* executable resource. Subsequently, if the LWM2M Server has a previously established notification relationship with the Resources in the IPSO Operations Object, it would be automatically informed by the gateway of a pending system-specific operation.

V. Evaluation and Testing

The enhancements to LWM2M discussed in the previous section were evaluated by implementing the proposed Objects for IoT gateways in an experimental setting. Our main use case for the validation of our approach was a simple energy management use case. Our test environment consisted of several devices to mimic a heterogenous gateway management scenario using LWM2M. This is depicted in Figure 4. Two kinds of gateways were set up. The first was TP-Link AC-1200 Wi-Fi Access Point (AP) connected to the Internet over a fixed Ethernet connection offering IPv4/IPv6 access. The stock firmware was replaced with the OpenWRT Chaos Calmer distribution and additional hardware and software was installed so that the AP was capable of scanning and communicating over BLE as a GATT client. The second gateway was a Raspberry Pi 2 serving as a BLE gateway primarily for bridging and connecting to BLE sensors. The BLE gateway was connected to the AP as a Wi-Fi client. Both gateways had embedded C-based LWM2M client code from the Eclipse Wakaama project modified and implementing the IPSO Wireless and System Gateway Objects, proposed PAN Object, BLE GATT Object as well as the Interaction Objects. A custom GATT server was also implemented in the BLE gateway with a proprietary service, enabling authorised clients to connect and write to a specific service and characteristic to control its Wi-Fi interface. Additionally, the gateway periodically announced its presence by transmitting BLE advertisements. The precise power being supplied to the BLE gateway was non-invasively monitored in real-time using a data acquisition (DAQ) device which provided real-time, precise, energy consumption statistics of the BLE gateway. The DAQ device was built in-house, containing an energy consumption monitoring toolkit, a Hall-Effect current sensor connected to an Arduino board and controlled over a Raspberry Pi Model B. Finally an Ubuntu laptop was also added into the setup for monitoring purposes. The laptop was also connected to the WiFi network and was capable of BLE-based communication. All 3 devices (the AP, the BLE gateway and the laptop) used the Linux BlueZ Bluetooth stack.

Both gateways registered to, and were managed over the cloud, using a public Java-based LWM2M server from the Eclipse Leshan project. The LWM2M server was hosted by the Eclipse Foundation and was residing at *leshan.eclipse.org*. An HTTP interface to the Leshan server allowed control, configuration and management operations using a web browser. This was performed using an Android-based phone forming a connection to *https://leshan.eclipse.org* over LTE.

To perform our tests, we initially controlled the server to successfully retrieve and write resources on both gateways. This was performed over IP to both gateways. However, both the AP as well as the BLE gateway registered their PAN

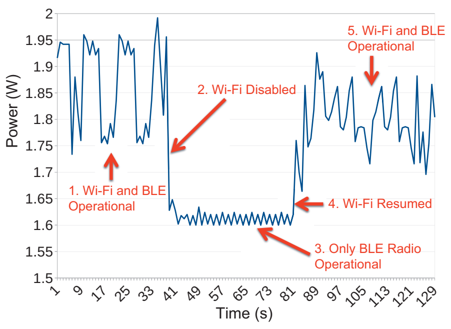


Fig. 5. Managing Energy level in the BLE gateway using LWM2M

Objects, with registration updates being performed every 300s. Both gateways also listed each other as peers in their respective BLE GATT Object Instances. Our intention was to see if using our approach with exposing system specific operations dynamically as well as proxying and working with non-IP endpoints, we can first successfully power down the Wi-Fi interface of the BLE gateway, thereby losing direct Internet connectivity between the LWM2M Server and the BLE Gateway without adversely affecting management. Subsequently, we wanted to demonstrate that by using the AP as a proxy, the LWM2M Server would be able to use the BLE Command Object of the AP to communicate with the BLE Gateway and power its Wi-Fi interface up once more. During these steps, the power consumption of the BLE Gateway should be observed to see if we were successful.

VI. Findings

Using the object modelling techniques described in the previous sections, the LWM2M server successfully executed several kinds of operations on the BLE gateway, depending on the communication context. The server was able to retrieve or change resource representations on the BLE gateway directly when IP connectivity was present, and the BLE gateway was able to provide the correct object links to allow the server to disable the WiFi interface. Additionally, once the WiFi interface was deemed to be disabled, the server was also able to retrieve the correct endpoint identifier of the AP and utilise it as a proxy to enable the WiFi interface via the APs BLE interface. The BLE gateway then resumed communication with the server over the higher powered WiFi interface. These events are graphically depicted in Fig 5, which shows the power consumption of the BLE gateway during this test scenario. From the graph, it can clearly be seen that periodic transmissions over Wi-Fi resulted in bursts of energy consumption, and when the Wi-Fi interface was powered down at around 40s, the power consumption of the BLE gateway decreased significantly. In order to allow the BLE gateway to update its LWM2M registrations, Wi-Fi was once again powered up and the subsequent activities such as association with the AP, obtaining IP addresses, resuming services and updating LWM2M registrations over Wi-Fi, are once again clearly visible. In addition to the power measurements, these activities were also verified by using the laptop to solicit ICMP responses from the BLE gateway to Ping packets, inspecting the LWM2M server for fresh registrations and subsequent updates from the BLE gateway at the Leshan server.

VII. Conclusion and Future Work

In this paper, we presented novel methods to manage IoT gateways effectively, using LWM2M. We focused on two specific areas, namely allowing vendor-specific and proprietary gateway operations to be effectively modelled as resources in an object model which the management server can interact with, and the ability to expose and manage endpoints by explicitly modelling proxy operations that gateways may possess. These were achieved by extending existing object models to contextually expose permitted operations on a gateway to a management server at runtime.

Work is ongoing for REST-based management of IoT devices and gateways, particularly in the IETF. RESTCONF [14] is one such example. RESTCONF is however, HTTP-based, and it faces limitations for constrained device management. Ongoing work on CoAP Management Interface (CoMI) [15] aims to adapt RESTCONF for use in constrained environments using CoAP. Additionally, the use of HATEOAS as well as Web Linking to define the relation types between object instances on different endpoints would heavily aid IoT gateway management. These are actively being pursued in other SDOs.

VIII. Ackowledgements

The authors thank Carsten Bormann for his review and useful comments. Funding was provided by the EIT Digital ACTIVE project.

References

[1] Z. Shelby, K. Hartke, and C. Bormann. ”Constrained Application Protocol (CoAP)”, IETF 7252, Jun. 2014.
[2] Open Mobile Alliance, ”Lightweight Machine-to-Machine Technical Specification v1.0, Candidate Enabler”, Aug. 2016.
[3] IPSO Alliance. IPSO Objects http://ipso-alliance.github.io/pub/
[4] H. Tschofenig, J. Arkko, D. Thaler, and D. McPherson, ”Architectural Considerations in Smart Object Networking”, IETF RFC 7452, March 2015.
[5] J. Prado,”OMA Lighweight M2M Resource Model”, IAB IoT Semantic Interoperability Workshop 2016, March 2016
[6] J. Jimenez, M. Koster and H. Tschofenig, “IPSO Smart Objects”, IAB IoT Semantic Interoperability Workshop 2016, March 2016
[7] OpenWrt UCI System, https://wiki.openwrt.org/doc/uci, Accessed September 2016.
[8] Bluetooth SIG, ”GAP REST API White Paper”, April 2014.
[9] Bluetooth SIG, ”GATT REST API White Paper”, April 2014.
[10] BIPSO, http://bluetoother.github.io/bipso/, Accessed September 2016.
[11] M. Kovatsch, Y. N. Hassan, and K. Hartke. ”Semantic Interoperability Requires Self-describing Interaction Models”, IAB IoT Semantic Interoperability Workshop 2016, March 2016
[12] M. Nottingham, ”Web Linking”, IETF RFC 5988, September 2016.
[13] K. Hartke, ”Observing Resources in the Constrained Application Protocol (CoAP)”, IETF RFC 7641, September 2015.
[14] A. Bierman, M.Bjorklund, and K. Watsen, ”RESTCONF Protocol”, ID-ietf-netconf-restconf, September 2016.
[15] P. van der Stok, and A. Bierman, ”CoAP Management Interface”, ID-vanderstok-core-comi, March 2016.

Publication VII

A Redundant Gateway Prototype for Wireless Avionic Sensor Networks

D. Scazzoli, A. Mola, B. Silverajan, M. Magarini, and G. Verticale

29th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE PIMRC 2018), pp. 1-7, IEEE.

DOI: 10.1109/PIMRC.2017.8292683

A Redundant Gateway Prototype for Wireless Avionic Sensor Networks

Davide Scazzoli*, Andrea Mola*, Bilhanan Silverajan†, Maurizio Magarini*, Giacomo Verticale*
*Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy
Email: {davide.scazzoli,maurizio.magarini,giacomo.verticale}@polimi.it, andrea1.mola@mail.polimi.it.
†Tampere University of Technology, Tampere, Finland
Email: bilhanan.silverajan@tut.fi

***Abstract*—Wireless Sensor Network (WSN) technologies provide advantages that allow them to replace traditional wired systems in an ever growing number of applications. This paper describes the design of a WSN for mission critical applications such as the case of avionics, in which data collected from the sensors can be delivered to a cloud application through multiple independent gateways, thereby increasing data availability in presence of failures. Since the same data might be distributed along multiple paths, system-wide synchronization must be provided in order to guarantee data consistency. A heartbeat protocol is introduced along each path in order to guarantee timely detection of any single failure. We present a solution that can be implemented using open source software and commercial off-the-shelf hardware, which makes this approach viable for networks with a large number of heterogeneous sensors. Results reported in this paper show some sample measurements as well as the performance evaluation for our heartbeat algorithm in terms of latency between a failure and a full recovery of the system.**

***Keywords*—Wireless Sensor Networks (WSNs), Reliability, Single Point of Failure (SPF), Mission Critical, Redundancy Management**

I. Introduction

Wireless sensor network (WSN) technology has been successfully employed in many non-critical application domains ranging from agriculture and habitat monitoring to smart buildings. Often they employ multi-hop communication and routing protocols over lossy channels with varying levels of reliability. However, a recent survey on reliability in WSNs indicates that by carefully selecting link metrics and power consumption levels, WSN-based systems can often provide a high level of reliability [1], which make them a feasible alternative to wired sensor networks for mission-critical applications. The International Telecommunications Union (ITU) arrived at a similar conclusion, and allocated the spectrum band between 4.2 and 4.4 GHz for the development of wireless avionics intra-communications (WAIC), with the intent of replacing traditionally cabled systems with wireless ones within aircraft [2].

In order to obtain real-time sensor data monitoring systems, a common technique used, particularly in the Internet of Things (IoT), is the device to gateway communication pattern for smart objects [3]. This pattern captures a common way in which sensors communicate with a local gateway over a short range wireless radio, which subsequently aggregates the data to be sent to a remote server or a cloud. This technique also allows the transmission of wireless aviation sensor data via a gateway situated in the aircraft, to ground-based command and control centers for real-time monitoring, either over a satellite or cellular connection. In such instances, the gateway itself becomes a critical component of the communication architecture.

However, while the reliability of WSN communication is being addressed by active research, and architectural patterns can be used to describe the role of the gateway for transmitting sensor data, there is little guidance given for the proper functioning and reliability of the local gateway, particularly for mission critical aviation systems. A gateway failure would mean complete blackout of all critical sensor data. Similar challenges exist in gateways of other mission-critical systems for autonomous land and maritime vessels.

In this paper, we look at addressing this aspect by describing a novel technique which employs heartbeats, algorithms and redundant gateways to improve failure detection of primary gateways and providing a fast recovery mechanism using a secondary gateway to continue transmitting sensor data to the cloud with no sensor data loss. Performance measurements using gateway prototypes built using low-cost commercial off-the-shelf (COTS) hardware demonstrate the feasibility of the approach. We remark that we focus on fail-over management for the redundant gateway assuming that the link between gateway instances is not a limiting factor, as we will see in Sec. IV. Henceforth, we rely on standard transport layer protocols to handle data delay, replication and link failures.

Realistic options exist for low-cost alternatives in WSN-based communications. An example of this is the emergence of XBee modules from Digi International [4] which provide different wireless connectivity solutions within the same form factor component for easy integration with other standard platforms such as Arduino. Coupled with open source software, general purpose hardware can address many of the major obstacles for sensor network technology to become a transformational force in mission-critical application domains, such as avionics. Using this approach it is possible to improve upon the weak aspects of current WSN applications such as lack of reliability, flexibility, interoperability, and in its extreme difficulties in long-term deployment, operation, and maintenance especially by non-skilled personnel [5].

The rest of the paper is organized as follows. Section II gives a survey of the redundancy management solutions available in the literature as well as typical WSN applications. The architecture of our WSN prototype is described in Sec. III, while Sec. IV illustrates our solution for handling redundant hardware. Experimental results are given in Sec. V and, finally, Sec. VI concludes the paper.

II. Related Work

A. Redundancy Management

Traditionally, fault detection, isolation and recovery (FDIR) is achieved by having several redundant systems interconnected with each other in some way. Each system monitors the output of other systems and detects faults by comparing results in a voting scheme [6]. Much work has been done within this method [7] and many systems adopt it for its robustness and reliability against silent data corruption (SDC) errors. While this method offers good reliability and performance it is resource intensive as multiple systems must perform the same task to provide the results for the voting algorithm, making the solution ill suited to the constrained environment of IoT and WSNs. More recently, an approach based on heartbeats has been introduced in the field of redundancy management [8]. This methodology was first adopted to monitor processes inside distributed software applications [9]. This method is based on the transmission of status packets called heartbeats in which each redundant system informs the other systems of its current status. Different models of this solution exist, the PUSH model, the PULL model and even hybrid solutions [10]. When using the PUSH model, the nodes being monitored periodically send heartbeats to the nodes monitoring them, while in the PULL model, the monitoring nodes request heartbeats from the nodes being monitored. Simulations of the performance of different implementations give interesting performance results [11], motivating further work in this field. This recent shift in methodology is further aided by the introduction of machine learning algorithms that are able to detect SDC failures without relying on any redundant data [12].

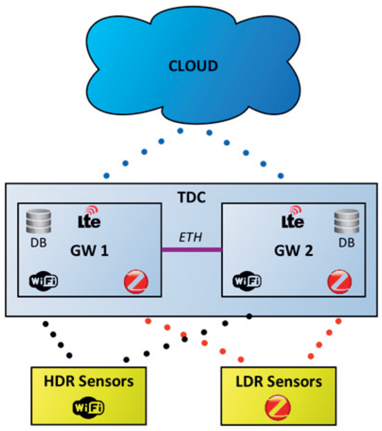


Fig. 1. System architecture. Multiple sensor nodes can connect with multiple gateways concurrently. All the gateways belong to a single entity defined as the Transmission Data Concentrator. Sensors are distinguished as High Data Rate (HDR) or Low Data Rate (LDR). Each gateway has an independent connection to the cloud.

B. Wireless Sensor Networks

The main objective of WSNs is the acquisition of data from sensor nodes without relying on any communication infrastructure. Many example applications of such technology exist in the literature, from wide area networks deployed to monitor an active volcano [13] to personal area networks used to monitor health parameters [14]. These solutions take advantage of the relatively close proximity of sensor nodes in order to gather data collected from multiple sensors into a single node, the sink, where it can then be stored and analyzed. By doing this, the nodes of a WSN are able to meet more stringent requirements than a general IoT node in terms of power consumption and hardware resources. In order to combine the advantages of IoT and WSNs the sink node often takes the role of an IoT gateway, delivering collected data from the sensor network towards the internet. Some example applications of this approach are emerging in the literature [15]. We aim at extending these results towards mission critical applications where reliability requirements often contrast with the limited resources paradigm.

III. System Architecture

We consider a WSN with different types of sensors, possibly characterized by different wireless technologies. The data sinks are two (or more) gateways, which are interconnected among them. The set of interconnected gateways is referred as Transmission Data Concentrator (TDC). The gateways are equipped with the wireless technologies used by the sensors and are also interconnected with each other by means of an independent communications medium, such as an Ethernet connection. We assume that the local link between them is highly reliable, that is, it is assumed free of failures. This assumption can be met without a large increase in costs by exploiting the close proximity of the gateways inside the TDC.

The wireless sensor nodes are broadly distinguished into high data rate and low data rate depending on whether they require more or less than 10 Kbps of capacity for transmission. For high data rate Wi-Fi was chosen for its low overhead and high capacity while for the low data rate we used ZigBee for the low energy consumption and high scalability. The respective benefits of these two protocols have allowed them to be used in many WSN applications, which, in turn, translated to a large amount of COTS devices made available on the market. The reliability of the communication from sensors to gateway is provided by standard transport layer protocols and is not a subject of this paper, while the redundancy on the sensor nodes is handled according to the type of sensor. For Wi-Fi we have implemented an ad-hoc network which allows the sensors to communicate with all the gateways at the same time, while for ZigBee we have addressed the problem of the network coordinator being a single point of failure by using the method described in [16].

Figure 1 shows the system architecture. The sensors acquire the relevant measurements and transmit them wirelessly. This has the advantage of eliminating the downtime needed for sensors to switch between gateways in the event of a gateway failure. Each TDC is thus responsible for data acquisition from the sensors and the configuration management of the sensor, which includes providing synchronization to the wallclock time. We achieve synchronization solutions by means of the Timing-sync Protocol for Sensor Networks (TPSN) [17], which can be seamlessly incorporated in our redundancy management algorithm described in Sec. IV. Internet connectivity to the Cloud is provided by the use of external components, *e.g.*, USB keys connected to the cellular network. A smart gateway, which is capable of processing, aggregating or even reducing the data volume transmitted towards the cloud, will greatly reduce resource consumption [18]. Another advantage of having smart field gateways capable of collecting and aggregating data in a meaningful form is a great simplification of maintenance procedures.

The functionality of the gateway is organized in five modules, as shown in Figure 2: sensor management, cloud management, redundancy management, controller and dashboard. These modules will be further described in Sec. IV-D. All these modules need to share information between each other so a database which is capable of handling concurrent access in a reliable manner is needed. In the prototype we use the Redis NoSQL, open

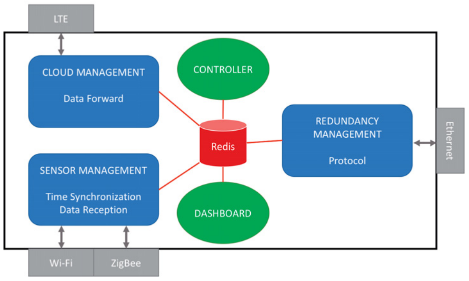


Fig. 2. Gateway software modules. The REDIS database handles concurrent data access from multiple independent services.

source, in-memory data structure store, which works as a lightweight database, cache, and message broker [19]. Further, Redis' keyspace notifications makes it possible to implement synchronous communications between the modules. This allows for easy management of multiple concurrent applications by exploiting a common data structure. Thanks to its replication function it is possible to quickly synchronize databases belonging to different devices when needed. We used this function in our prototype to fill gaps in the data among different gateways as further described in Sec. IV.

IV. Gateway Redundancy Management

To explain our approach for managing redundant hardware, we consider a TDC with two gateways, in which one acts as a primary gateway while the second acts as a secondary gateway. The primary gateway contains the master database and is responsible for sending the relevant data towards the cloud. The secondary gateway is kept in hot standby, in order to assume the role of the primary gateway and minimizing downtime in the event of a failure in the primary gateway. A protocol was designed and implemented in the gateways, which allows each gateway to participate in the election of primary and secondary roles. Subsequently this protocol was generalized to the case of N gateways joining the TDC at arbitrary time instants.

In this section we describe the general protocol structure, the assumptions, the definition of the failure points, the procedures for cold start, and how failure situations are managed.

A. *Design Assumptions and Goals*

We consider the following design assumptions:

- No power constraints in the TDC. The TDC is always on to collect data from sensors that can be freely transmitted once ready.
- There are no bandwidth constraints inside the TDC. Since the gateways inside the TDC will be in close proximity this assumption is easily verifiable.

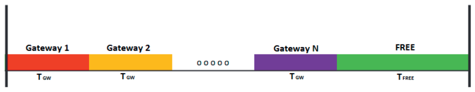


Fig. 3. Heartbeat Period

- A faulty gateway is silent. Each gateway either operates correctly or is silent. We do not consider the case of gateways forwarding incorrect messages or sending heartbeat messages but ignoring data messages.
- No residual transmission errors. The wireless MAC protocols employ error correction and error detection mechanism that result in erroneous message being discarded by the receiver.
- Non-catastrophic failure scenario. We consider that at least one gateway is working at any given time.

B. Periodic Heartbeat Procedure

The heartbeat period is dynamic, because it depends on the number of active and available gateways. Within each period, each gateway must send a heartbeat packet inside a predefined time window, as shown in Figure 3. After all active gateways have transmitted their heartbeats, a time window of duration T_{free} is allocated to allow other gateways to join as well as handling the rest of the traffic present on the channel such as data synchronization between the gateways. A gateway joins the network by successfully transmitting its heartbeat in the T_{free} slot, after all other currently active gateways have transmitted theirs. Once joined it will be allocated a new slot of time duration T_{gw} for transmitting its heartbeat packet by all the gateways which have received its heartbeat in the previous cycle. The application of this structure makes a timely identification of missing heartbeats easy to implement. Due to the close proximity of the gateways it is possible to take advantage of high speed communication technologies such as Ethernet or Wi-Fi. Since the heartbeat contains only essential information its impact on the communication resources is limited, thus T_{gw} is generally much smaller than T_{free}, as such it is possible for it to share the communication channel with other services. The heartbeat packet transmitted by the gateways contains the following information:

- **hostname:** The machine hostname. It must be unique in the network. As it does not impact our algorithm we do not delve into hostname or address assignment.
- **gateway_state:** There are three possible gateway states: *available*, *backup*, *outofservice*. These states will be described later in this section.
- **gateway_role:** Either *primary* or *secondary*.
- **total_number_of_gateways:** The number of gateways in *available* or *backup* state seen by the transmitting gateway in the previous heartbeat transmission interval.
- **list_of_interfaces:** The List of the gateway's monitored interfaces with respective working/not working status. In the prototype considered in this paper, the interfaces are ZigBee, Wi-Fi, and LTE. The Ethernet was not included as it is assumed free of faults.
- **reliable:** Reports the last data set acknowledged by the cloud. It is used in case of a primary gateway failure by the gateway which takes over. By resuming transmission from this point data continuity is guaranteed even after a takeover.
- **gateway_id:** This field shows the place of the heartbeat inside the dynamic frame structure. It is used by gateways which join the network to identify the place of the heartbeat received inside the dynamic frame structure.

Algorithm 1 Gateway Initialization and main loop

```
for all interfaces do
    check interface status
end for
if all interfaces online then
    gateway_state ← available
else if all interfaces offline then
    gateway_state ← out_of_service
    shutdown
else
    gateway_state ← backup
end if
Listen for Heartbeats for time $2(T_{GW} + T_{FREE})$
if Heartbeat detected then
    gateway_role ← secondary
else
    gateway_role ← primary
end if
loop
    Transmit Heartbeat in allocated timeslot $T_{GW}$
    Store Received Heartbeats
end loop
```

C. Employed Procedures

The gateways are divided in three possible groups: *available*, *backup* and *out-of-service*.

A gateway belongs to the *available* group only when all the interfaces are working. This means that the gateway can receive the data from all the sensor nodes, and it can forward them to the cloud. On the contrary, the out-of-service group is related to the gateways with no working interfaces. Otherwise, gateways that have partial failures will belong to the backup group. These gateways can assume a primary role only if there are no other available gateways. Even when there are only backup

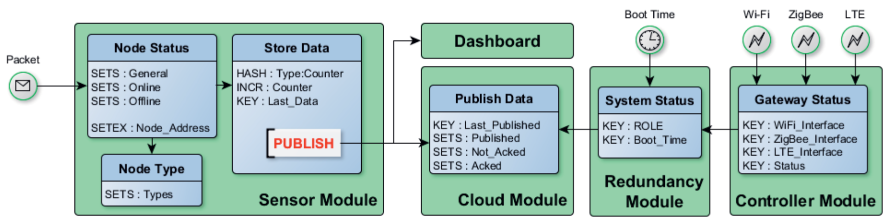


Fig. 4. Data structure of the Redis database shared by the services running on the gateway and their relationship.

gateways running if there is at least one working interface for each network section (LTE, ZigBee and WiFi) the system can still work by leveraging on the Redis Replicate function. By using this function the databases from the gateways can be shared via the redundancy channel enabling gateways with malfunctioning interfaces to still receive data by sharing another gateway's connectivity. This is possible thanks to the assumption of faultless local link which was stated earlier in this section.

There are three main procedures to handle role assignments which are detailed below.

a) Initialization: When a gateway is switched on, it verifies the status of all its interfaces and, according to the result, it decides which status to assume between the *available*, *backup* and *out-of-service*. After this initial check it looks for the other gateways by listening for other gateways' heartbeats for a minimum time of $2(T_{\text{gw}} + T_{\text{free}})$. If no heartbeat is received, the gateway self-elects as primary and begins broadcasting its own heartbeat. Otherwise, the gateway role depends on its status and those of the seen gateways. In the case of equal status the one that was already transmitting the heartbeat will retain the primary role while other gateways joining the network will assume secondary role. This procedure allows for a dynamic number of gateways to join the TDC arbitrarily, thus supporting scenarios where some gateways are in cold-standby or might attempt to fix internal issues by rebooting.

b) Main: After the gateway initialization, the periodic procedure consists in sending a heartbeat for the other gateways. In the case of the primary gateway, it sends its heartbeat with its log data and with the `gateway_id` field equal to 1. Otherwise, it replies back with its heartbeat and increments the `gateway_id` field by 1 for each heartbeat received. After each gateway has transmitted its heartbeat inside their T_{gw} slot they start listening for a period of time defined as T_{free}. During this interval the available and backup gateways listen for heartbeats of new gateways that want join the TDC.

c) Failure: An interface failure is detected by the gateway thanks to the self-verification test. When a failure is detected the gateway will switch to backup

Algorithm 2 Election of a new Primary Gateway

```
if Primary_State != available or Primary Heartbeat Timeout then
    if gateway_state == available then
        for Each available gateway do
            Check if gateway_id is greater
        end for
        if Own gateway_id is the lowest then
            gateway_role ← primary
        end if
    end if
end if
```

status and, depending on the status of the other gateways, it may also switch role to secondary. If, after the failure, there are no available gateways without failures it will maintain its primary role until a new gateway with available status joins the network. After the other gateways discover the failure, either through notification or a timeout, a new primary gateway is elected following the procedure depicted in Algorithm 2. For cases where multiple gateways have the same status then the primary is chosen by the order established in the frame, which is indicated in the `gateway_id` field of the heartbeat packet.

D. Modular Architecture

The gateway software modules interact with each other through the Redis data channels. Figure 4 shows the data structures used during normal operation by each module:

a) sensor module: stores various information about the nodes of the WSN, such as their status: online or offline, whether they have been connected before or not and the type of available sensors. To do that, four Redis sets and one key with the expiration time have been defined. Of these, three keep track of the sensor nodes which are currently online, offline and online at least once. The last one is used to track the type of sensor node which is identified from the node's MAC address. To update the sets, for example in case of a node failure, a key value pair with the expiration time is set. Once this key is expired, the system will publish a

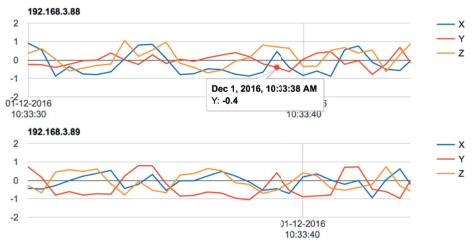


Fig. 5. The cloud dashboard used to visualize data which is sent from the primary gateway twice per second. In this instance the data from two sensor nodes equipped with accelerometers is displayed. A failure of the primary gateway is simulated at the time 10:33:38 indicated in the figure, however, data continuity is maintained.

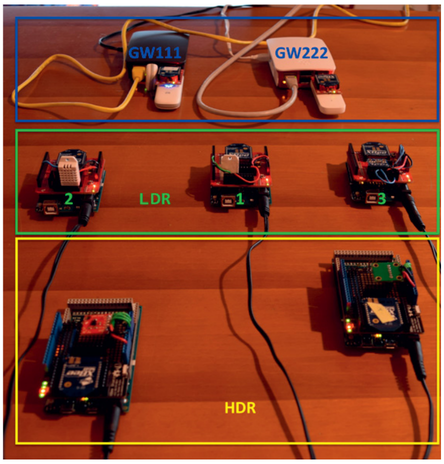


Fig. 6. The experimental prototype used for verification of the redundancy algorithm. In the figure are shown the high data rate Wi-Fi sensor nodes at the bottom, the low data rate ZigBee sensor nodes in the middle and the two gateways at the top. The hardware used consists in Arduino UNO with XBee S2 transceivers for the LDR sensors, Arduino DUE with XBee S6B transceivers for the HDR modules and, for the gateways, Raspberry Pi 3 Boards with XBee S2 transceivers and LTE keys.

related notification, and a software module will update the offline and online sets. A node can have multiple different sensors or one sensor can sample different physical quantities. Hence, for any node a set is created which contains the data measured and the timestamps associated.

b) controller module: keeps track of the status of the various interfaces by using four key values as shown in Fig. 4. Three are used for the gateway interfaces which were described in Sec. III while the last one is used for the status of the gateway as a whole which, as described earlier in this section, can assume three states: Available, Backup or Out of Service.

c) redundancy module: utilizes only two keys, one for the boot time and one for defining the gateway role, which can be primary or secondary, as described earlier in this section.

d) cloud module: keeps track of the status of the data sent towards the cloud by updating three sets, Published, Not Acked and Acked. The Published set is filled with the relative data coming from the sensor nodes, as they get forwarded toward the cloud the data is moved to Not Acked and lastly Acked when the Cloud acknowledges the received data. These sets also act as a backup of the data and are used by the secondary gateways for retransmission in case of a failure of the primary one. Data consistency is guaranteed by the use of application layer protocols such as MQTT or CoAP.

e) dashboard module: is tasked with the local display of the sensor data. It can be used for maintenance procedures or as a backup when cloud connectivity is unavailable.

V. Experimental Results

We have tested the effectiveness of the redundancy management algorithm in handling failures by implementing the protocol in a prototype. Then, we simulated various failures and measured the system reaction times. Figure 5 shows an example of the cloud dashboard showing the Wi-Fi sensor nodes equipped with 3-axis accelerometers. We simulated a crash of the primary gateway at time 10:33:38 AM and, as can be seen from the figure, the continuity of the data is maintained despite the failure. The performance was measured in the time taken for the system to fully recover after a failure which is given by the equation:

$$T_{\text{down\_time}} = T_{\text{detection}} + T_{\text{heartbeat}} + T_{\text{role\_switch}}, \quad (1)$$

where $T_{\text{detection}}$ indicates the time taken for a gateway to detect a failure of their interfaces, $T_{\text{heartbeat}}$ the time taken to transmit an updated heartbeat and $T_{\text{role\_switch}}$ the time taken by the secondary gateway to switch to primary role. This time was measured experimentally with our prototype which is depicted in Fig. 6. We used two gateways which exchanged heartbeats every 100 ms. The reduced dimension of the heartbeat packet limits the impact on the network resources. We performed 200 tests which consisted in simulating an interface failure on the primary gateway and measuring the interval of time elapsed until a heartbeat with switched role was received from the other gateway, all of this was recorded using a laptop connected as a control unit. The histogram of the measured intervals is reported in Figure 7.

From these results we can see that, in the majority of cases, the response time is below the heartbeat exchange interval that we used. Nevertheless, a non negligible amount of simulations gave higher downtimes. This is caused by the channel used for running the algorithm, as

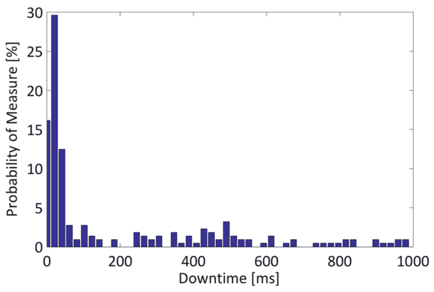


Fig. 7. Histogram of the observed downtimes for our proposed heartbeat algorithm measured with our prototype. The gateways were set to exchange heartbeats with an interval of $2T_{\mathrm{gw}} + T_{\mathrm{free}} = 100\,\mathrm{ms}$.

inside this channel there were two Secure Shell (SSH) sessions open for monitoring the gateways as well as gathering the experiment results. This result helps in dimensioning proper measures against the worst case scenarios that can happen in a field implementation where limited resources are contested by multiple services. From our experimental verification we have an average time to recover of approximately $170\,\mathrm{ms}$, this is comparable to results obtained with non COTS devices using the same Ethernet protocol we employed in the redundancy channel [11].

VI. Conclusion

This paper describes an architecture for the management of faults in gateway nodes in a wireless sensor network with the aim of meeting strict reliability requirements of mission critical applications while using off-the-shelf commercial hardware and open source software. The main contribution is the development of an algorithm for efficiently managing the redundant hardware in the constrained environment typical of wireless sensing applications. Data consistency across different devices' databases is guaranteed despite partial link failures thanks to transport layer protocols, which allows us to neglect issues like duplicated delayed packets and missed sensor data broadcasts. We have deployed this algorithm in a prototype within the context of Wireless Avionic Intra Communications. From experimental verification, we have observed that the algorithm provides a quick response to failures. Its simplicity allows easy implementation and limited taxing of the resources available. Many options are available as future work in this field. The redundancy management can be improved to work together with machine learning algorithms to deal with silent data corruption errors. The prototype architecture enables the further study of data aggregation solutions for machine type communication in upcoming Internet of Things scenarios.

Acknowledgment

This work has been partially funded by the EIT Digital HII ACTIVE Project and the DIMECC Design4Value Programme.

References

[1] M.A. Mahmood, W.K.G. Seah, I. Welch, ”Reliability in wireless sensor networks: A survey and challenges ahead”, Computer Networks, Volume 79, Pages 166-187, ISSN 1389-1286, 2015.
[2] International Telecommunication Union (ITU). Technical characteristics and operational objectives for wireless avionics intra-communications (WAIC). Report ITU-R M.2197, 11, Geneve 2010.
[3] H. Tschofenig, J. Arkko, D. Thaler and D. McPherson, ”Architectural Considerations in Smart Object Networking”, RFC 7452, DOI 10.17487/RFC7452, March 2015.
[4] Digi International Inc. http://www.digi.com/
[5] J. Williams, “Internet of Things: Science Fiction or Business Fact?” Harvard Business Review Analytic Services Report, December 2014.
[6] Osder S. ”Practical view of redundancy management application and theory” Journal of Guidance, Control, and Dynamics 22.1 (1999).
[7] Fayyaz M. and T. Vladimirova. ”Fault-tolerant distributed approach to satellite on-board computer design” Aerospace Conference, 2014 IEEE.
[8] F. T. Shane and D. B. Thomas. ”Heterogeneous Heartbeats: A framework for dynamic management of Autonomous SoCs” Field Programmable Logic and Applications, 24th International Conference IEEE, 2014.
[9] Hoffmann, Henry, Jonathan Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. ”Application heartbeats for software performance and health” ACM Sigplan Notices 45, no. 5 (2010): 347-348.
[10] Zou, J. X., Zhang, Z. Q. and Xu, H. B. (2010). ”Design of heartbeat invalidation detecting mechanism in triple modular redundancy multi-machine system” COMPEL-The international journal for computation and mathematics in electrical and electronic engineering, 29(2), 495-504
[11] Lei, Z., Liu, F., Yang, S., Heng, Z. ”Dynamic heartbeat algorithm of redundant mechanisms in WIA-PA” 3rd IEEE International Conference on In Computer Science and Information Technology (ICCSIT), 2010
[12] Hinojosa, Alfredo Parra, et al. ”Handling silent data corruption with the sparse grid combination technique” Software for Exascale Computing-SPPEXA 2013-2015. Springer International Publishing, 2016. 187-208.
[13] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, ”Deploying a wireless sensor network on an active volcano” IEEE Internet Computing, vol. 10, pp. 18-25 (2006).
[14] Lo, Benny PL, et al. ”Body sensor network - a wireless sensor platform for pervasive healthcare monitoring” (2005): 77-80.
[15] Ferdoush, Sheikh, and Xinrong Li. ”Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications” Procedia Computer Science 34 (2014): 103-110.
[16] D. Scazzoli, A. Kumar, N. Sharma, M. Magarini, and G. Verticale. ”Fault Recovery in Time-Synchronized Mission Critical ZigBee-Based Wireless Sensor Networks” International Journal of Wireless Information Networks, 1-10, 2017.
[17] Ganeriwal, Saurabh, Ram Kumar, and Mani B. Srivastava. ”Timing-sync protocol for sensor networks” Proceedings of the 1st international conference on Embedded networked sensor systems. ACM, 2003.
[18] Dawy Z, Saad W, Ghosh A, Andrews JG, Yaacoub E. ”Toward Massive Machine Type Cellular Communications” IEEE Wireless Communications. 2017 Feb.
[19] Redis. https://redis.io. Accessed: 2016-11-18.

Publication VIII

Measuring Energy Consumption for RESTful Interactions in 3GPP IoT Nodes

T. Savolainen, N. Javed and B. Silverajan

7th IFIP/IEEE Wireless and Mobile Networking Conference (WMNC 2014), pp. 1-8, IEEE.

DOI: 10.1109/WMNC.2014.6878863

Measuring Energy Consumption for RESTful Interactions in 3GPP IoT Nodes

Teemu Savolainen
Nokia, Finland
Email: teemu.savolainen@nokia.com

Nadir Javed
Nokia, Finland
Email: nadir.javed@tut.fi

Bilhanan Silverajan
Tampere University of Technology, Finland
Email: bilhanan.silverajan@tut.fi

***Abstract*—As the Internet of Things (IoT) evolves to encompass ever increasing quantities of smart devices, sensors and other smart objects, attention must be paid to considering what kinds of wireless networks should be employed, and the data transfer protocols to be used to allow communication among these entities. It is highly important that energy consumption for communication remains as minimal as possible. In this paper we present power consumption measurements in end devices, when REST-based resource retrieval is performed with HTTP and CoAP over 3GPP EDGE, HSPA and LTE networks. Our findings are based on actual measurements taken over the radio interface of a mobile handset in live cellular networks, and show that for a few transactions of small packet sizes, HTTP performs comparatively well in terms of power consumption. Also, power consumption is lowest in the handset when EDGE is used, but an LTE network with operator assisted power savings approaches power consumption levels seen with EDGE. Finally, for sessions consisting of large number of transactions, we show that using CoAP over the WebSocket protocol results in significantly less power consumption compared to HTTP.**

***Index Terms*—Energy Measurement, IoT, CoAP, HTTP, 3GPP.**

I. Introduction

Today, leading cellular technologies are those based on the Third Generation Partnership Project (3GPP) standards, including 2.5G Enhanced Data rates for Global Evolution (EDGE) [1], 3G High Speed Access (HSPA) [2], and 4G Long Term Evolution (LTE) [3]. They are ubiquitous, globally available and provide a wide coverage area. Because they support Internet Protocol (IP)-based data communications, they are very attractive for wireless smart objects which form the basis of the Internet of Things (IoT) [4]. Such smart objects comprise many kinds of networked devices, ranging from computationally powerful smartphones and tablets to highly energy constrained sensors and actuators. Interactions are envisioned to occur among things as well as human users arbitrarily regardless of time and location.

There is also an expectation that IoT communication is highly data driven, and will converge with the vision of the Web of Things [5], where seamless integration and interoperability with the World Wide Web (WWW) occurs for IoT nodes in exchanging, serving and retrieving resources. An important paradigm for Web-based resource retrieval is the Representational State Transfer (REST) [6] architectural approach. Two significant REST-based protocols are the Hypertext Transfer Protocol (HTTP) and the Constrained Application Protocol (CoAP) [7].

IoT-based communication is anticipated to be substantial, and heavily dominate cellular and Internet traffic. REST-based interaction forms a significant share of this activity. Using cellular networks requires greater power compared to wireless communication technologies such as IEEE 802.11ah, IEEE 802.15.4, and Bluetooth Low-Energy. As cellular networks themselves provide only basic IP-connectivity, IoT nodes have to select the appropriate transport and application layer protocols to utilize for power efficiency on their own.

To date, little or no work exists in literature giving an indication of the energy consumption of IoT nodes when REST-based interactions are performed in cellular networks. In this paper we address this by undertaking detailed, live measurements to analyse the power consumption of a cellular node interacting via 3GPP networks consisting of 2.5G EDGE, 3G HSPA, and 4G LTE technologies.

We performed extensive data acquisition on power consumption for REST-based communications on these cellular networks. Such communication considered not only HTTP, but also CoAP, CoAP over WebSockets [8] and CoAP over Secure WebSockets. Section II provides a background of the REST-based protocols for IoT and the justification for measuring them. Section III describes aspects of 3GPP cellular networks that impact a host's power consumption and presents related studies. Section IV describes the experimental setup used for measurements. The methodology for the measurements and preliminary results are described in Section V. The findings based on measurements are documented in Section VI where the impact on energy consumption resulting from the type of 3GPP network as well as REST protocol are presented. We then conclude the paper in Section VII.

Throughout this document 2G refers to EDGE, 3G to HSPA, and 4G to LTE. Furthermore, "CoAP" refers to CoAP over UDP, "CoAP+WS" to CoAP over WebSockets, "CoAP+WSS" to CoAP over WebSockets secured with Transport Layer Security (TLS), and "HTTP" for HTTP over TCP. "REST-based protocol" or "REST protocol" refers to any of these application-level REST-based resource retrieval mechanisms.

II. IoT Resource Retrieval

REST-based resource retrieval and manipulation is ideal for use in the IoT, as its communication primitives are simple and well understood. A Universal Resource Identifier (URI) providing sufficient information about each resource is advocated, while four types of methods (GET, PUT, POST and DELETE) manipulate the representation of resource states. In the IoT, many resource constrained nodes act as both clients and servers. Examples of such nodes are tiny sensors behaving as servers and actuators behaving as clients. Less constrained nodes such as smartphones or workstations can also communicate with IoT endpoints, but the REST architecture stipulates that such communication is undertaken in a simple request-response style purely for resource retrieval or manipulation.

HTTP-based APIs for many Internet services and cloud platforms are prevalently REST-based today [9]. Although HTTP communication between Internet servers and smart objects and sensors is popular [10], [11], it is verbose, text-based, and not suited for compact message exchanges. Instead, an extremely lightweight and bandwidth efficient alternative called CoAP [7] is being developed in the Internet Engineering Task Force (IETF). Like HTTP, CoAP also uses URIs and REST-based methods and similar response and error codes, that allow both constrained and non-constrained nodes to interact. It is, however, completely binary, and it uses UDP which is more suited than TCP for lossy and constrained networks. Hence, CoAP nodes, therefore, cannot communicate with HTTP nodes as is. One solution for this would be to use a CoAP-HTTP protocol translation gateway, in which an intermediate proxy node assists in translating CoAP method calls and URIs to HTTP and vice versa [12]. Proxies can also choose to translate TLS-based *https://* into *coaps://* URIs, in which CoAP uses Datagram Transport Layer Security (DTLS).

Using such a gateway would be useful in networks where Internet access is allowed only via an HTTP proxy. It is also useful in corporate or cellular networks which have restrictive administration policies on some non-TCP and non-HTTP traffic, such as CoAP over UDP, but from which clients are allowed to conduct end-to-end bidirectional HTTP sessions to well-known web servers. For the client however, HTTP-CoAP proxying would continue to incur the cost of using HTTP between itself and the translation gateway without gaining any advantage in CoAP communication.

A lower overhead to HTTP can be incurred instead by substituting UDP with the WebSocket [13] protocol to transport CoAP packets [8]. This avoids the need for HTTP proxying, but requires the client and server to first mimic an HTTP session by performing an HTTP handshake over TCP, before upgrading it to a WebSocket session. During the WebSocket handshake, the use of CoAP is negotiated between the endpoints. Subsequently, CoAP messages can be exchanged by conveying them within WebSocket frames. End-to-end security in this case, can be preserved, using secure WebSockets (WebSocket over TLS). The paper therefore focuses on the relative energy profiles of these REST-based protocols: HTTP, CoAP and the use of CoAP over (secure) WebSockets.

III. 3GPP Background and related work

Power management is one of the biggest challenges in today's cellular devices. Gadgets have to be charged often, which is to great extent caused by power consumed for wireless communications. The power consumption depends on many things, some of which are physical in nature like distance to base stations, and some of which are logical and caused by properties of used protocols. 3GPP cellular access' power consumption has been analyzed by several research papers, such as those from Henry Haverinen *et al.* [14], Jui-Hung Yeh *et al.* [15], [16], Niranjan Balasubramanian *et al.* [17], and GSMA [18]. Studies have also been performed for suitability of LTE as a wireless technology for IoT gateways. Costantino *et al.* used packet level simulations on ns-3 to investigate performance issues with LTE when conveying CoAP traffic from interconnected edge devices via a gateway [19].

Below we briefly describe the main aspects of 3GPP cellular network protocol features relevant to our measurements.

A. PDN connection

In 3GPP networks logical Packet Data Network (PDN) connections are established for transportation of IP packets [20]. The PDN connection in 2G and 3G is referred as Packet Data Protocol (PDP) context, and in 4G as Evolved Packet System (EPS) bearer. The PDN connection can be considered to be layer two in Open Systems Interconnection (OSI) model. For each PDN connection IPv4 and/or IPv6 addresses are allocated by the network. Throughout this paper an already established and always-on PDN connection is assumed. This means that for each measurement we took, the PDN connection had been already established, and continued to be active after the measurement ended. While setup and teardown of PDN connections do consume energy, an established PDN connection effectively does not.

B. 2G properties

Procedures used for Medium Access Control/Radio Link Control (MAC/RLC) at the GPRS radio interface are specified in 3GPP 44.060 [21]. A key feature is Temporary Block Flow (TBF), which is a unidirectional MAC layer connection between a mobile station and a network. For bidirectional transfers TBFs are set for both directions. The TBF is set up, and radio resources allocated for it, only temporarily for the duration of the data transfer. TBF is closed after configurable time of 0 to 1500 milliseconds after the data transmission ends. No TBF is active when a cellular device is idle.

Niranjan Balasubramanian *et al.* found 2G GSM to be more power efficient than 3G, especially for smallish (<500kB) data volumes [17]. Pauls Friedrich *et al.* studied the feasibility of 2G GPRS for machine-to-machine (M2M) [22] and the research focused on developing a connection model and using that to analyze whether nodes should stay always-on or completely turn off the cellular radio when not transmitting. The results from their research points to use always-on for scenarios where frequent communications are needed.

C. 3G properties

Features of 3G's Radio Resource Control (RRC) state machine [23] are significant for cellular devices' power consumption. RRC can be in idle mode or in connected mode with four possible substates: CELL_DCH (Dedicated Channel), CELL_FACH (Forward Access Channel), and URA_PCH (URA Paging Channel) or CELL_PCH (Cell Paging Channel). The URA_PCH is not currectly used [14]. The CELL_DCH state is designed for bulk data transfer, but is also the most power consuming. The CELL_FACH state, designed for signaling and transfers of few hundred bytes, consumes roughly 40% of the CELL_DCH [18]. The optional CELL_PCH state consumes about 1-2% of the CELL_DCH [14]. Entry into the CELL_DCH state is triggered by the network, for example, after an operator configurable amount of data is sent within a period of time. After T1 seconds of inactivity in CELL_DCH state, the RRC transitions to the CELL_FACH state, from where transition to the CELL_PCH state or to idle mode occurs after T2 seconds of inactivity. If the CELL_PCH state is used and entered, transition to idle mode happens after T3 seconds of idle in CELL_PCH. T1 and T2 timers are usually configured for two seconds or slightly more, while T3 can be several minutes [14].

In 3GPP Release-8 the CELL_FACH state is improved with enhanced dedicated channel (E-DCH), which allows larger amounts of data to be sent before switch to CELL_DCH occurs [24]. The 3GPP Release-8 also allows devices to actively enter to less power consuming states by sending a Signalling Connection Release Indication (SCRI) message, which is also referred as Fast Dormancy [23], [18].

D. 4G properties

4G LTE networks are able to provide fast connectivity for smartphones, but with the expense of complex and power consuming circuitry. LTE standards support a technique called discontinuous reception (DRX), which enables significant energy savings for applications that do not require constant data streams. When a mobile station is in the DRX mode, it listens for incoming data in (DRX) cycles. The DRX cycle is divided into short and long DRX cycles. When DRX mode is entered, first a short DRX cycle is used for a predefined time before switching to a long DRX cycle. The short DRX cycle can have values from two to 320 milliseconds, and the long DRX cycle can have values from 32 to 2560 milliseconds [25]. The longer the time in DRX cycle, the more power is saved, but the higher the latency as transmissions can start only at these cycle intervals. For applications requiring fast responsiveness shorter cycle times are needed, whereas delay tolerant applications can accept longer cycle times. Similar to 2G and 3G, the DRX system contains tunable parameters such as the DRX inactivity timer T1, which defines time without traffic until enabling of DRX. As of this writing, DRX is supported in some but not all commercial 4G LTE networks.

IV. Experimental setup

The experimental setup for our energy consumption investigation, described in Figure 1, consisted of a prototype Nokia Lumia Windows Phone 8 smartphone using Qualcomm Snapdragon™S4 dual-core processor and connected to 2G, 3G and 4G cellular networks. The smartphone periodically transmitted resource records with GPS data to Internet-connected PCs running Ubuntu and Windows, using REST protocols: HTTP, CoAP, CoAP+WS or CoAP+WSS.

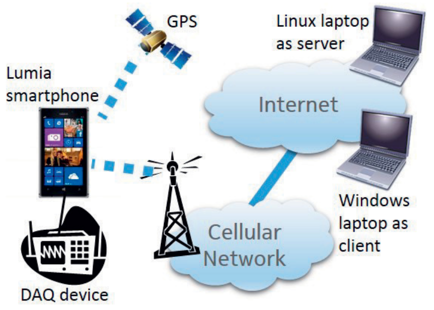


Figure 1: Setup for measuring energy consumption

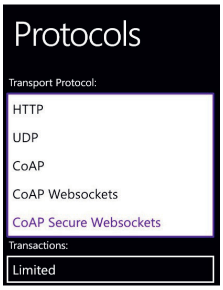
Instead of a built-in battery, power to the smartphone was supplied and instrumented using a proprietary precise data acquisition (DAQ) device based on a National Instruments chipset. The DAQ device was used as an ammeter which monitored and delivered the total power consumption in realtime via a USB interface to a PC.

The phone was equipped with a test app that enabled the smartphone to supply GPS data to remote endpoints, in effect mimicking a smart object or IoT node capable of periodically transmitting sensory data or location information. The app maintained the GPS receiver in a constant, powered-on state throughout its lifetime, and used a native Geolocation API to acquire location co-ordinates every few seconds. As depicted in Figure 2, selection for the required REST protocol could be performed at runtime by the application via a settings page, although for the purpose of this paper, experiments over raw UDP were not conducted. For our measurements, an option for sending the location information as a fixed number of transactions was also provided in the same page. The implementation and usage of the remaining REST protocols were as follows:

- For HTTP communication, the HTTP client library in .NET framework was used. HTTP POST requests were periodically transmitted to an HTTP server with the location information supplied as a URL-encoded payload.
- For CoAP communication, an in-house C# CoAP protocol library was used. The app behaved as a CoAP server listening for incoming requests from CoAP clients. The CoAP server supported the CoAP Observe option, which a CoAP client specified in its GET request message to

retrieve the location information resource. This subsequently set up a notification relationship between the CoAP server and the client, in which individual, periodic CoAP 2.05 Response messages were sent by the server containing the location information as a binary payload.

- For CoAP+WS communication, another in-house CoAP library was used, which was extended with the *WebSocket4Net* WebSocket library for .NET. In this case, the app initially behaved as an HTTP client, and upgraded the HTTP connection into a WebSocket session with the server. During the WebSocket handshake, usage of CoAP as a sub-protocol was negotiated. Subsequently, the app periodically transmitted CoAP POST requests with the location information supplied as a binary payload to the server, encapsulated as WebSocket frames.
- For CoAP+WSS, app behaviour and implementation was identical with plain WebSockets. TLS version 1.0 was used to secure the WebSocket connection. A 1024-bit RSA key encoded in an X.509 certificate was used for the TLS handshake, in which the use of Advanced Encryption Standard (AES) cipher with a 256 bit keysize was negotiated for payload encryption.


Protocols

Transport Protocol:

HTTP
UDP
CoAP
CoAP Websockets
CoAP Secure Websockets

Transactions:

Limited

Figure 2: App Settings: Ways of delivering GPS information

Two laptops on the public Internet were used to communicate with the smartphone. Both laptops were equipped with packet capture tools to allow on-the-wire packet inspection. A Windows-based laptop with an in-house C# implementation served as the CoAP client obtaining location updates after setting up a CoAP Observe relationship. An Ubuntu Linux laptop was used to communicate with the app for the other REST protocols: Apache, together with a small PHP based implementation was used as the HTTP server, while a Node.js-based implementation with the *Worlize* WebSocket-Node library was used by the server end-point for CoAP+WS and CoAP+WSS communication.

V. Methodology

This section presents our methodology and measurements. We start by describing the acquisition of our dataset and then follow with analysis of costs in terms of transactions as well as energy consumption.

A. Data Acquisition

Data acquisition was performed in four types of live cellular networks having the following types of connectivity: EDGE, HSPA, LTE without DRX enabled and finally, an 80ms cycle DRX-enabled LTE. In each of these networks, attention was paid towards determining the energy consumption of deterministic REST-based transactions, in which the transmission rate for requests or responses are kept constant.

Two types of data samples were taken in which the transmission intervals were either 1s or 10s. For the 1s intervals, 20 points of measurements were taken in which the number of REST transactions ranged from 1 to 20. For the 10s intervals, 10 points of measurements were taken in which the number of REST transactions ranged from 1 to 10. This acquisition process was performed four times in total, to compare energy patterns for HTTP, CoAP, CoAP+WS and CoAP+WSS.

From the data harvested from the smartphone, a cumulative total of 480 distinct energy consumption patterns were acquired over several weeks. The DAQ equipment provided means to detect and eliminate the phone's idle power consumption, which includes the display backlight, away from the obtained readings. While the power consumption figures in the measurements taken for cellular radio communication included location fetching, the actual power consumed by the GPS receiver remained consistent throughout the tests at 45mA, and hence did not interfere with the final measurements. Figure 3 provides a single graph containing the plots of all the energy consumption patterns of the REST protocols in all the cellular networks, for 1s location information transmission intervals. Certain lines are highlighted, while others are intentionally greyed, to emphasise the range of results and variations. The rest of the paper discusses these.

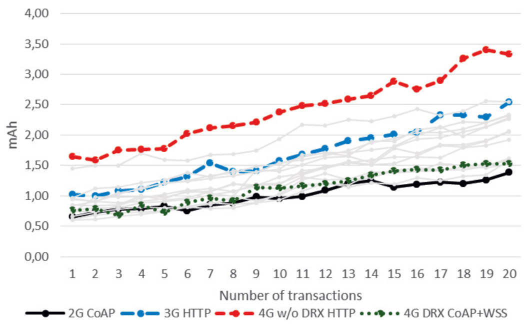


Figure 3: Spread of measurements for 1s intervals.

B. Transaction Costs

Major factors influencing power consumption used for resource updates hinge on the number of connectivity events, packets, and bytes transferred for each REST protocol and underlying transport. For a single transaction involving a resource update, Table I provides a breakdown of the number

Table I: Bytes and packets exchanged for connection setup, RESTful update, and closure. Measured for session containing only one transaction. Figures include all TCP signaling, where applicable.

Protocol	Setup bytes	Setup packets	RESTful bytes	RESTful packets	Closure bytes	Closure packets	Total bytes	Total packets
CoAP	102	3	94	2	0	0	196	5
CoAP+WS	656	6	124	2	252	6	1032	14
CoAP+WSS	aprx. 2252	16	aprx. 440	6	aprx. 577	aprx. 9	3193	28
HTTP	144	3	763	4	160	4	1067	11

Table II: Power costs for 10s (left) and 1s (right) transmission intervals in mAh for protocols tested in 2G, 3G, and 4G.

Protocol	2G one trans.	3G one trans.	4G w/o DRX one trans.	4G with DRX one trans.	2G additional trans.	3G additional trans.	4G w/o DRX additional trans.	4G with DRX additional trans.
CoAP	0,82 / 0,65	0,78 / 0,84	1,46 / 1,45	0,70 / 0,72	0,30 / 0,04	0,31 / 0,06	0,62 / 0,06	0,33 / 0,05
CoAP+WS	0,53 / 0,61	0,72 / 0,76	0,80 / 0,97	0,73 / 0,66	0,35 / 0,05	0,30 / 0,06	0,60 / 0,07	0,31 / 0,05
CoAP+WSS	0,71 / 0,62	0,76 / 0,85	1,15 / 0,94	0,76 / 0,76	0,29 / 0,05	0,34 / 0,06	0,66 / 0,07	0,37 / 0,04
HTTP	0,65 / 0,74	0,82 / 1,02	1,36 / 1,65	1,08 / 0,96	0,31 / 0,08	0,61 / 0,08	0,63 / 0,09	0,41 / 0,07

Table III: Number of theoretical exchanges and lifetime for 2000mAh battery for 10s (left) and 1s intervals (right).

Protocol	2G exchanges	2G lifetime	3G exchanges	3G lifetime	4G w/o DRX exchanges	4G w/o DRX lifetime	4G with DRX exchanges	4G with DRX lifetime
CoAP	6,6k / 52k	18h / 14h	6,6k / 31k	18h / 9h	3,2k / 35k	9h / 10h	6,0k / 44k	17h / 12h
CoAP+WS	5,8k / 39k	16h / 11h	6,7k / 33k	19h / 9h	3,3k / 28k	9h / 8h	6,4k / 44k	18h / 12h
CoAP+WSS	6,8k / 38k	19h / 11h	6,0k / 32k	17h /9h	3,0k / 29k	8h / 8h	5,3l / 50k	15h / 14h
HTTP	6,4k / 24k	18h / 7h	3,2k / 25k	9h / 7h	3,2k / 23k	9h / 6h	4,9k / 28k	13h / 8h

of IP-level packets and bytes exchanged for both requests and responses. This breakdown shows a general distribution of the number of packets needed for various stages, such as session setup (data exchanged prior to sending resource updates), resource update and its acknowledgement, as well as connection closure.

The table can be also used as a guide to understanding setup and teardown overheads when multiple transactions are sent using each REST protocol. Using CoAP as a protocol with our app incurs the overhead of three packets to set up the Observe relationship, and two packets for the update transaction. Because CoAP uses UDP, no overhead is present for session teardown when Observe cancellations are not used. HTTP needs four packets for each RESTful update: An HTTP POST packet, a TCP ACK, an HTTP OK packet and its corresponding TCP ACK packet.

With CoAP+WSS, while the number of setup and RESTful packets can be determined accurately, the actual number of bytes per packet exhibited slight variations across several runs of our measurements. This can be attributed to the usage of variable length padding by the TLS record layer, to frustrate attacks based on the analysis of the length of exchanged messages [26]. Closure figures are also an estimation due to TLS interleaving the final transaction and session closure messages.

C. *Energy Costs*

From the measurements taken, Table II contains the energy cost for one transaction, including setup and closure, and cost for additional transaction during the same session. In each row, results are displayed in the form *x*/*y* where *x* is the cost incurred for 10s transmission intervals, while *y* is that for 1s intervals. In the case of HTTP, CoAP+WS and CoAP+WSS, the follow-up exchanges occur over the established transport layer connection. In the case of CoAP, in this measurement, the initial transaction is the CoAP GET with Observe from CoAP client to server and one notification, and the follow-up exchanges are additional notifications.

Table III further illustrates how the energy consumption differences can affect battery lifetimes for an IoT node doing REST transactions. In this case, the measured values have been extrapolated for a 2000mAh battery with the assumption that all (or a very significant amount of) the power is available for message exchanges.

VI. Measurements and Findings

Based on our measurements, both Figure 3 as well as Tables II and III indicate that the best protocol and radio combination allowed approximately 60% more power savings compared to the worst. This highlights that the selection of which RESTful protocol to use, and particularly which radio access network to use, is an important decision for power constrained nodes. Measurements from the DAQ device are presented in Figures 4-8 and 11, in which horizontal axes represent time in seconds (s) while vertical axes represent current consumption in milliamperes (mA). As an example, Figure 4 graphically depicts an overview of what power consumption patterns look like for CoAP+WSS transactions, in this case highlighting the differences in 2G, 3G, and 4G (with and without DRX) networks.

A. *2G Networks*

In 2G measurements, the setup and teardown of the TBFs cause significant overhead per transaction, if transactions happen seldom enough. This can be seen when comparing the power consumption profiles we gathered, in Figures 5 and 6 and Tables II and III. In the case of ten second intervals, the TBFs are opened and closed for every transaction, which

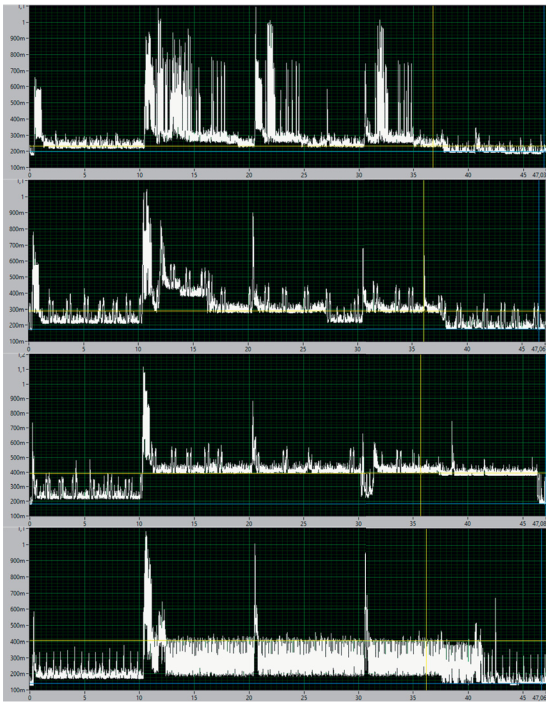


Figure 4: Energy consumption profiles of the same CoAP+WSS transactions over 2G, 3G, 4G without DRX and 4G with DRX (from top to bottom).

causes significant power consumption when compared to the RESTful interaction itself. This is why additional transaction costs for ten second intervals are roughly equal for all REST protocols in 2G. However, in one second interval scenario TBFs are continuously active and hence the amount of data to be transmitted for RESTful interactions becomes dominant. Thus, in the one second interval case, additional transactions with HTTP are significantly more expensive than with other REST protocols.

B. 3G Networks

As mentioned in Section III-C, the CELL_DCH state consumes substantially more energy than the CELL_FACH state. Figure 7 illustrates how HTTP pushes the radio to the CELL_DCH state in every transaction, while CoAP seldom does so. Compared to other REST protocols as shown in the Table II, HTTP fared significantly worse in ten second interval cases, the main reason being that HTTP always pushed the radio to CELL_DCH. In one second interval measurements however, HTTP fared better, as for such fast intervals all REST protocols caused the 3G radio to stay in the CELL_DCH state throughout.

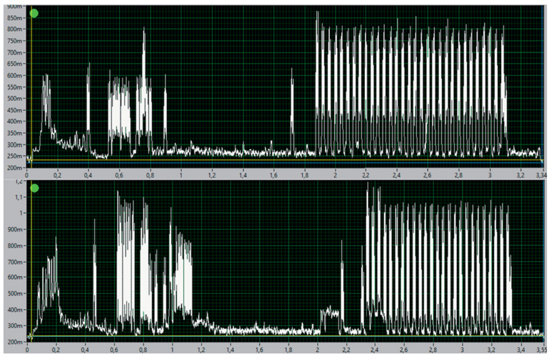


Figure 5: Transactions from the middle of long lived session using 10s intervals on 2G. CoAP above and HTTP below.

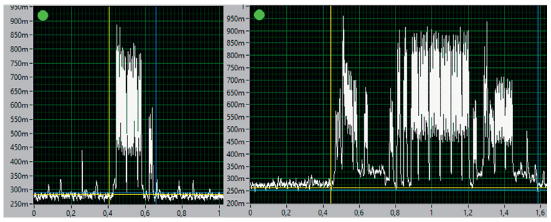


Figure 6: Transactions from the middle of long lived session using 1s intervals on 2G. CoAP left and HTTP right.

C. 4G networks

Some observations can be made by looking at the results shown in Table II. First of all, 4G with DRX is significantly less energy consuming than without DRX. The effect of DRX is further illustrated in Figure 8. The 80ms DRX cycle used in the access networks helps power consumption to drop significantly. A longer DRX cycle would have pushed power consumption down even further.

D. Findings

Setting up the CoAP Observe relationship with the first resource update occurring ten seconds after observation establishment, as well as the TCP session idle timeout closure for HTTP after ten seconds of inactivity, cause additional connectivity events for CoAP and HTTP scenarios, which are not present in CoAP+WS tests. This causes visible spikes in power consumption in some one-time transaction measurements, such as with 4G without DRX. This difference could be minimized by implementations providing the CoAP resource immediately, and closing the TCP socket right after the final HTTP transaction. The impact of this extra session diminishes as transactions increase within the same connection.

Assuming CoAP provides energy savings when compared to HTTP is easy, as significantly less bytes are transmitted. However we discovered this not to always be the case. CoAP does save energy whenever the cellular radio is operating on

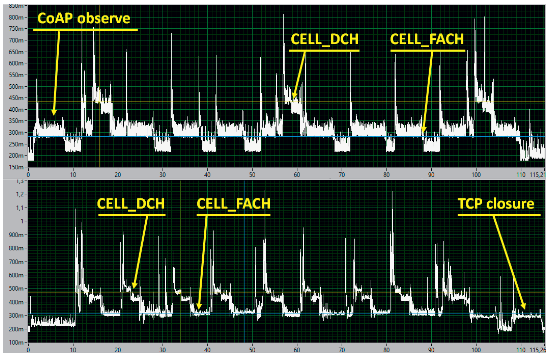


Figure 7: Nine transmissions over 3G with CoAP (above) and HTTP (below).

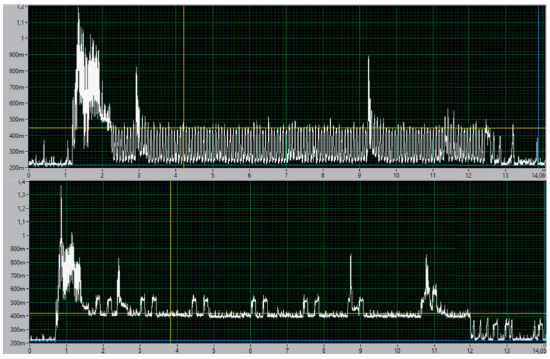


Figure 8: Single WebSocket transaction in 4G with 80ms DRX cycle (above) and without DRX (below).

a mode where the difference in packet size has significant importance, i.e. when other radio signaling related properties are not dominating. These scenarios, from those we tested, include 4G when DRX is supported as illustrated in Figure 9, 3G when smaller data allows staying in CELL_FACH channel, and in 2G if transmission frequency allows avoiding TBF setup and teardown costs. From this it follows that CoAP does not bring savings when packet size difference between these RESTful protocols does not play a major role, as illustrated in Figure 10. Such networks are 4G without DRX, 3G with a transmission rate high enough to always push the radio to the CELL_DCH state, and 2G with low transmission rates causing TBF signaling to become a dominant factor.

As data was acquired over several weeks, radio conditions in live networks used varied. Readings were obtained from approximately the same location, preserving the physical distance to cell phone towers and minimising variations in network coverage and latency. All measurements were observed and remeasured in cases where very significant disturbances were present, such as excess retransmissions. The phone's UI registered excellent field strength throughout, between

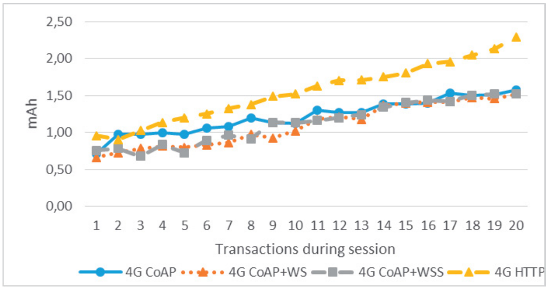


Figure 9: Power consumption in 4G DRX for 1s intervals.

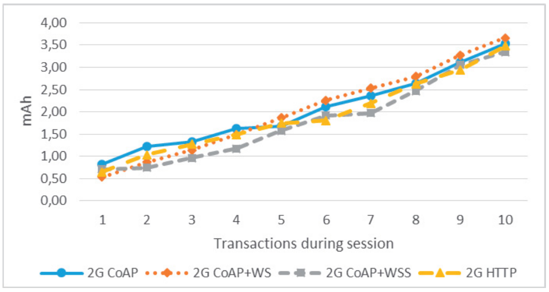


Figure 10: Power consumption in 2G for 10s intervals.

80-100%. The biggest variation measured was for CoAP in the 2G network, at about 13% for one time transactions between one second and ten second interval measurements. These two particular measurements are shown in Figure 11, showing almost identical energy patterns, but with different power consumption peak levels (1000mA versus 800mA). This registered a consumption of 0,83mAh for one test and 0,63mAh in the other.

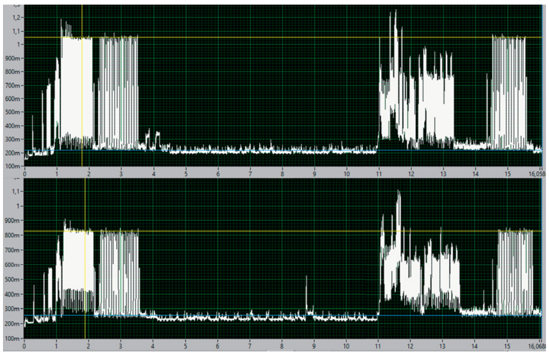


Figure 11: CoAP observe with one notification, measurements for 1s (below) and 10s interval (above) cases. This illustrates how modem power consumption levels can vary at different measurement dates and radio conditions.

VII. Conclusions and future work

We studied the energy consumption on an endpoint performing RESTful interactions in various kinds of cellular networks prevalent today. Our measurements employed a Lumia smartphone, but eliminated all other factors such as display and background tasks, to focus on the relative power consumption of REST protocols in cellular networks. Measurements were discarded which indicated unrelated smartphone background activity. Consequently, we believe the measurements are proportionately indicative of cellular communication patterns of most IoT nodes that perform constant, periodic data transmissions of just a few bytes. Nevertheless, because no previous data on the subject exists, the measurements we amassed were extensive, and only the main findings can be presented here.

In the right network conditions, CoAP potentially allows RESTful operations with less energy costs in 3GPP cellular networks, when compared to HTTP. In no case does CoAP consume more energy than HTTP, visibly providing solid performance by being essentially either superior or on par with HTTP. We were positively surprised how feasible energy consumption-wise CoAP+WS and even CoAP+WSS (using a strong cipher and variable length padding) were, as highlighted for CoAP+WSS in Figure 3. These REST protocols were on par, or fared better than HTTP, with most scenarios indicating a closer correlation to energy profiles for CoAP, than HTTP. Based on this, we expect the energy consumption of DTLS-based CoAP (*coaps://*) to equal or better CoAP+WSS. HTTP, on the other hand, at best is equal to CoAP, but at its worst is significantly poor as highlighted in Figure 3. This indicates to us that for implementations which have no control of the radio technology used but wish to conserve battery, the use of CoAP, even with WebSockets if needed, is a safe bet. That said, if the deployment scenario is known well-enough, HTTP might be a better option as it would not consume significantly more, but would otherwise be more proven and ubiquitious.

2G cellular networks proved to be most efficient for transporting small messages, at the location of our tests. Almost ubiquitous availability and good energy consumption of 2G, when compared to 3G or especially to 4G, would seem to make 2G the best choice for energy constrained nodes. That said, CoAP's performance in 4G networks having DRX enabled was very good. With the used DRX cycle of 80ms the consumption figures were already close to that of 2G. With a longer DRX cycle the consumption could be even better, and could actually make 4G with DRX the least power consuming option.

It would be useful to test 3G Fast Dormancy and E-DCH improvements, which should significantly improve 3G's performance. Those improvements could bring 3G closer to 2G and 4G with DRX, and also smoothen differences between HTTP and CoAP by making the entry to CELL_DCH less common, and allowing quicker transitions back to idle mode.

VIII. Acknowledgements

We thank our employers, Nokia and Tampere University of Technology, for the opportunity to work on this topic. Jukka Kujala, Jonne Siren, and Hannu Bergius from Nokia aided in getting us started with measurements and understanding some of the measurement results. The C# CoAP implementation on the phone relied on code written by Klaus Hartke.

References

[1] 3GPP. "Technical Specification Group GSM/EDGE Radio Access Network;Overall description;Stage 2", TS, 43.051, V11.0.0, Sep 2012
[2] 3GPP. "High Speed Downlink Packet Access (HSDPA);Overall description;Stage 2", TS, 25.308, V11.7.0, Dec 2013
[3] 3GPP. "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN);Overall description;Stage 2", TS, 36.300, V11.8.0, Dec 2013
[4] ITU, "The Internet of Things", International Telecommunication Union Internet Reports, 2005.
[5] D. Guinard, V. Trifa, and E. Wilde. "A resource oriented architecture for the web of things." Internet of Things (IOT), 2010, pp. 1-8. IEEE, 2010.
[6] R. Fielding. "Architectural Styles and the Design of Network-based Software Architectures". Doctoral dissertation, Univ. of California, 2000.
[7] Z. Shelby, K. Hartke, and C. Bormann. "Constrained Application Protocol (CoAP)", IETF draft-ietf-core-coap-18, Work-in-Progress, Jun 2013.
[8] T. Savolainen, K. Hartke, and B. Silverajan. "CoAP over WebSockets", IETF Internet-Draft draft-savolainen-core-coap-websockets-01, Work-in-Progress, Oct. 2013.
[9] Wan Jung; Sang Il Kim; Hwa Sung Kim, "Ontology modeling for REST Open APIs and web service mash-up method," 2013 Int'l Conference on Information Networking, vol., no., pp.523,528, 28-30 Jan. 2013
[10] Supermechanical, "Twine: Listen to your world, talk to the Web", http://supermechanical.com/twine/features.html, 2013
[11] K. Au-Yeung, T. Robertson, H. Hafezi, G. Moon, L. DiCarlo, M. Zdeblick, and G. Savage. "Networked system for self-management of drug therapy and wellness." Wireless Health 2010, pp. 1-9. ACM, 2010.
[12] A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk. "Guidelines for HTTP-CoAP Mapping Implementations", IETF draft-ietf-core-http-mapping-02, Work-in-Progress, Oct 2013.
[13] I. Fette, A.Melnikov. "WebSocket Protocol", IETF RFC 6455, Dec 2011.
[14] H. Haverinen, J. Siren, P. Eronen, "Energy Consumption of Always-On Applications in WCDMA Networks," Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th , vol., no., pp.964,968, 22-25 April 2007 doi: 10.1109/VETECS.2007.207
[15] J.-H. Yeh, C.-C. Lee, and J.-C. Chen. "Performance analysis of energy consumption in 3GPP networks", Wireless Telecommunications Symposium, 2004, pp. 67-72, 2004
[16] J.-H. Yeh, J.-C. Chen, and C.-C. Lee. "Comparative analysis of energy-saving techniques in 3gpp and 3gpp2 systems," Vehicular Technology, IEEE Transactions on, vol. 58, no. 1, pp. 432-448, 2009.
[17] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications", 9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC '09, pp. 280–293, 2009
[18] GSMA. "Fast Dormancy Best Practises ", TS.18, Version 1.0, Jul 2011
[19] L. Costantino, N. Buonaccorsi, C. Cicconetti, and R. Mambrini. "Performance analysis of an LTE gateway for the IoT," 2012 IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM), vol., no., pp.1,6, 25-28 June 2012
[20] 3GPP. "General Packet Radio Service (GPRS);Service description;Stage 2", TS, 23.060, V12.3.0, Dec 2013
[21] 3GPP. "General Packet Radio Service (GPRS);Mobile Station (MS) - Base Station System (BSS) interface;Radio Link Control / Medium Access Control (RLC/MAC) protocol", TS, 44.060, V11.7.0, Nov 2013
[22] F. Pauls, S. Krone, W. Nitzold, G. Fettweis, and C. Flores. "Evaluation of Efficient Modes of Operation of GSM/GPRS Modules for M2M Communications", Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th, pp. 1-6, 2013
[23] 3GPP. "Radio Resource Control (RRC); Protocol specification", TS, 25.331, V11.8.0, Dec. 2013
[24] 3GPP. "Enhanced uplink; Overall description; Stage 2 (Release 11)", TS, 25.319, V11.8.0, Dec 2013
[25] C.S. Bontu and E. Illidge. "DRX mechanism for power saving in LTE", Communications Magazine, IEEE, vol 47, number 6, pp. 48-55, 2009
[26] A. Pironti. "Length Hiding Padding for TLS", IETF draft-pironti-tls-length-hiding-02, Work-In-Progress, Sep 2013.

Publication IX

Home Network Security: Modelling Power Consumption to Detect and Prevent Attacks on Homenet Routers

B. Silverajan, M. Vajaranta, and A. Kolehmainen

11th Asia Joint Conference on Information Security (AsiaJCIS 2016), pp. 9.16, IEEE.

DOI: 10.1109/AsiaJCIS.2016.10

Home Network Security: Modelling Power Consumption to Detect and Prevent Attacks on Homenet Routers

Bilhanan Silverajan, Markku Vajaranta, Antti Kolehmainen
Tampere University of Technology, Finland
Email: firstname.lastname@tut.fi

***Abstract*—Future home networks are expected to become extremely sophisticated, yet only the most technically adept persons are equipped with skills to secure them. In this paper, we provide a novel solution to detect and prevent attacks on home routers based on anomalous power consumption. We developed a means of measuring power consumption that could be used in a wide variety of home networks, although our primary focus on is on profiling Homenet-based residential routers, specifically to detect attacks against homenet routing infrastructure. Several experimental results are presented when the infrastructure is exposed to various types of attacks, which show strong evidence of the feasibility of our approach.**

***Index Terms*—IETF Homenet, Home Network Security, Power Consumption.**

I. Introduction

Networks in the home today consist of relatively simple setups. In the majority of homes, a broadband router or cellular gateway delivers connectivity to devices in the home, either wirelessly over Wi-Fi, or using an Ethernet cable. In all these instances, a single subnet is offered, usually behind a NAT with private IPv4 addresses. Additional gateways are rarely used, except as repeaters.

In recent years however, the home has rapidly emerged as a natural convergence point for technological developments and innovations. It is not only commonplace to have connected homes contain smart consumer devices, sensors, remote surveillance systems and home automation, but increasingly having mobile devices and even vehicular networks joining into a home network when needed. Additionally, better connectivity options and more advanced networking possibilities, such as support for IPv6 that provides end-to-end communication without the need for NATs, are now also made possible from service providers and network operators.

The Home Networking Working Group (Homenet WG) was subsequently chartered by the Internet Engineering Task Force (IETF) as a response to the rapidly increasing arrays of devices, computers, sensors and gateways that are constantly being added into residential networks, as well as a means to simplify end-to-end communication, service integration and network management by the home network owner, network operators and service providers. Among others, the Homenet WG proposed that home networks need to adopt an architecture that allows them to scale and evolve organically as the complexity of the services and devices grow.

One of the most significant recommendations made is that, with the introduction of IPv6 for home networking, a Homenet-compliant residential network (simply referred to from now on as "homenet"), should support multiple networks and subnets. A homenet can therefore consist of multiple routers forming a proper routing infrastructure complete with its own routing protocol. The other is that, as the owners of a homenet do not normally comprise technically adept persons, minimal (and ideally zero) configuration of addressing and networking needs to be performed: Users simply connect their devices to their home subnet of choice, and the homenet infrastructure should automatically handle all the intrinsic details for addressing, routing, service discovery and reachability.

In terms of home network security, several aspects of protecting home networks and devices, such as data privacy, access control and end device protection have been investigated in current research publications. However, apart from well-known administrative practices such as ensuring the use of good passphrases and passwords for wireless connectivity, employing firewalls and access control lists in the gateway, little if any security research on home gateway security actually exists. For example, Geon Woo Kim *et al.* [1] discuss the need to protect home networks from a variety of malware, Distributed Denial-of-Service (DDoS) and eavesdropping attacks through the introduction of a framework providing guarantees of authentication, authorisation and a rule-based security policy engine for undertaking actions when security infractions occur. Mohamed Abid [2] studies the use of biometric authentication to enable and personalise user access into a home network. Also, Shaojun Qu [3] discusses remote authentication and authorisation to provide secure access into the home environment.

On the other hand, Lucas Dicioccio *et al.* [4] reveals that, while on the whole, even if the number of connected devices in four out of five home networks amount to less than a dozen, home gateways in particular are always active at any given time. This is exacerbated in homenet-based residential networks, when a proper routing infrastructure having multiple routers remains active at any given time. Therefore, even if the idea of an automatically configured routed network in the home provides convenience and significant advantages to various kinds of users and smart devices, the homenet routers themselves can become highly susceptible to malicious activity unknown to the network owner. This creates new security challenges, as now home routers and gateways can be exposed to attacks to intercept and subvert routing and traffic, or inject malicious router into the home network.

Therefore, the role of security for the protection of a homenet's routing infrastructure is important, and can fall into several considerations. Firstly, access control is needed to prevent unauthorised eavesdropping, and permitting only authorised routers to join the core network. This relies on proper policies and credentials for channel security, such as symmetric keys, strong passphrases or certificates. Secondly, authentication and verification of routing messages must be performed, which allows a router to distinguish and discard malicious or spoofed packets, particularly when a shared wireless medium is used. Related to authorisation as well as authentication is the need to ensure the completeness, correctness and integrity of routing information. This deters attackers from being able to insert a malicious router into the homenet and subsequently inject traffic or cause data destruction to subvert the routing network.

In conjunction with these considerations, the requirement of availability needs to also be addressed, particularly so for routers that could be battery powered. Such routers can exist as gateways to vehicular networks, or to extend the homenet over a wider area for a limited period of time. Any sort of availability requirements have typically referred to resilience against Denial-of-Service (DoS) attacks intended at disabling services and network infrastructure. When battery-operated routers are taken into account, availability also refers to resilience against attacks designed to drain energy, by extensive or prolonged communication or computational activity [5]. As these can be launched at virtually any layer of the communication protocol stack, they can often go unnoticed until power states are diminished to near critical levels.

While attacks on simple home gateways have been mounted remotely over the Internet, wireless attacks on homenets can be realistically compromised by an attacker physically nearby. Also, residential networks are normally not managed by highly technical users, who often lack the type of knowledge and the sophisticated tools to cope with detecting and preventing attacks on routers.

Hence, we propose a novel approach to protect and detect malicious activity in a homenet, particularly with regards to the routing infrastructure. Our approach relies on profiling the power consumption of homenet routers. By studying the energy patterns of the routers in various scenarios, such as during normal activity or when various kinds of attacks are in progress, unusual behaviour in the routers can be correlated with spikes or elevated levels of the consumed energy. This allows network owners to detect attacks in progress to be detected with a high level of confidence.

Using anomalous power consumption as a means to detect attacks is a promising area of research that is at the moment still in its infancy. However, the idea has been applied in several mobile and wireless domains. For example, research has been successfully performed on anomalous battery drains on mobile phone to detect malware as well as unknown software bugs [6], [7]. Timothy K. Buennemeyer *et al.* [8] describes a battery-sensing intrusion protection system which detects irregular communication activity over IEEE 802.11 Wi-Fi and 802.15.1 Bluetooth. However, to the extent of our knowledge, applying anomalous power consumption as a metric for threat detection in routers and routing systems, has not been performed yet.

Therefore, the aim of our paper is to detail our empirical power measurements of homenet routers and findings under different kinds of attack scenarios designed to disrupt network routing or drain the power from a battery operated network. In so doing, we attempt to provide evidence of the feasibility of our approach, which can then be used to ensure further resilience and robustness of homenets. Since password-cracking brute-force attacks as well as DoS attacks on home networks have been well documented, our focus is instead on the securing the routing infrastructure of the homenet. Therefore, our investigation centers around measuring the power consumption of homenet routers when the wireless channel security is being compromised, or when the routing protocol used by a homenet, called Babel, is targeted.

Section II provides a brief background of the Homenet architecture, relevant protocols as well as expected deployment scenarios. Section III details the experimental setup for our measurements. Section IV provides a detailed explanation of each experiment and obtained measurements, while Section V then discusses our findings. Our conclusions are subsequently given in Section VI.

II. IETF Homenet

In this section, we describe work of the IETF Homenet WG to standardise home networks which is relevant to this paper. An exhaustive discussion of the entire architecture, however, is out of scope, and the interested reader is encouraged to refer to the relevant working group documents.

The Homenet WG's intent is to research and standardise networking protocols and other mechanisms useful for residential home networks [9]. Supporting IPv6 natively, providing automatic networking and service discovery as well as surviving uplink disruptions were perceived as important goals. The home network is also envisioned to grow large enough to require multiple network segments and subnets within the home, therefore a critical requirement of the architecture is to allow the existence of several routers, which then need to be orchestrated to perform actual routing in the network, using one more more well-known interior gateway routing protocols.

For proper operation, ISPs supporting homenet-based residential networks are required to enable IPv6 and then support DHCPv6-based prefix delegation, so that a requesting home router would be supplied an IPv6 address prefix, instead of a single IPv6 address. This roughly corresponds to an ISP supplying an IPv6 address block to the home network, and the router subsequently delegating IPv6 addresses and address prefixes to other home devices and routers as necessary.

When more than 1 router is present in the homenet, an interior routing protocol is used within the home. Currently the routing protocol of choice in Homenet WG is Babel [10], an ad-hoc multi-hop mesh networking distance vector protocol. Babel possesses properties such as loop avoidance, rapid convergence and high performance. A Babel-based mesh network is resilient to link disruptions, as the protocol adapts and repairs the mesh topology based on measured link quality, ensuring a high level of end-to-end reachability. Babel also has a low memory footprint, and works well over both fixed Ethernet links as well as wireless 802.11-based radio. These latter properties make Babel an ideal candidate for homenet, since a residential gateway and routing infrastructure consists of inexpensive off-the-shelf consumer-grade hardware which are not as powerful as enterprise-level network equipment and generally possess the ability to provide IP connectivity over Wi-Fi and fixed Ethernet. In some cases, these can also be resource-constrained or battery-powered routers, allowing the homenet to encompass vehicular gateways and extend to peripheral residential areas for limited periods of time.

Finally, homenet routers obtain and distribute information about the capabilities, routing protocols and services in the homenet using the Homenet Control Protocol (HNCP) [11] .

While homenet is designed to work with IPv6, the technology is IP agnostic to end-devices, and IPv4 connectivity works just as well too. In a homenet deployment consisting of multiple routers, it is envisioned that typical deployments would rely on a border router communicating with an ISP to obtain an IPv6 prefix for the homenet. Other internal routers would communicate with the border router over fixed Ethernet, or over Wi-Fi using ad-hoc mode and create a mesh-based routing network running Babel. Devices in the home then are supplied connectivity over Wi-Fi using infrastructure mode over a different radio. Alternatively, depending on the router hardware, if only one physical Wi-Fi radio interface is available, it is also possible to advertise 2 different Service Set Identifiers (SSIDs), and hence virtual wireless interfaces: the first for joining into the babel routing mesh using ad-hoc mode, and the second for supplying connectivity to devices in the home using infrastructure mode. The border router also periodically transmits router advertisement messages into the home network, and therefore both end-devices and internal routers can automatically configure their IPv6 addresses using Stateless Address Autoconfiguration (SLAAC), in addition to obtaining private IPv4 addresses using DHCP.

In default modes, neither Babel, nor the ad-hoc mesh network running over Wi-Fi, require any security extensions for proper interworking. Any router part of the same mesh is able to communicate and exchange routing information with any other existing router, while the entire Wi-Fi mesh network can be set as an open ad-hoc network which broadcasts a common SSID specifically for connecting access points. Nevertheless, it is prudent to employ security measures in both Babel as well as the wireless mesh network. For Babel, message authentication can be enabled with a Hashed Message Authentication Code (HMAC) cryptographic authentication [12]. When HMAC is used, two compliant hash algorithms must be supported, both having a 160-bit digest: RIPEMD-160 and SHA-1. Additional hash algorithms may also be supported as described in RFC 7298. For Wi-Fi, WPA2-PSK authentication, which uses a human-readable passphrase, can be employed.

III. Experimental setup

Our test environment consisted of several portions. Firstly, we created an ISP capable of providing Internet connectivity to various home networks via IPv4 and IPv6. A DHCP server delivers a single IPv4 address to home border routers (as most ISPs do today), while IPv6 prefix delegation consisting of a */56* prefix is provided to supply the home network with global IPv6 addresses. Secondly, we then deployed a Homenet-compliant residential network with four wireless TP-Link AC-1200 routers, which each have a Qualcomm Atheros QCA9558 CPU, 16MB of flash and 128MB of flash memory. Routers were positioned several meters apart from each other approximately equidistant, in order to form a fully connected mesh. Each router was capable of dual band Wi-Fi on both 2.4Ghz and 5GHz radio interfaces. The stock firmware was replaced with the latest OpenWRT "Designated Driver" distribution, based on Linux kernel version 4.1.16. The *hnetd* (for HNCP) and *mdnsd* (for multicast service discovery) packages were installed. Because the *babeld* (for Babel routing) provided as a package in OpenWRT does not support HMAC-based authentication, we cross-compiled a custom babeld for OpenWRT from the source code provided by the *Quagga* RE project, in which HMAC authentication was supported. The 2.4 GHz radio interfaces were dedicated towards creating the wireless mesh network in which the Babel routing protocol was utilised. While the 5 GHz radio interfaces can advertise Wi-Fi connectivity to client devices, for the purpose of our measurements, we did not enable the interface to eliminate any measurement bias from communication with end-devices. Thus, the power consumption figures obtained corresponded directly to traffic originated and exchanged among the routers themselves within the Babel-based wireless mesh network.

Fig 1 depicts the test environment. In addition to the routers, the setup also consisted of a commercial industrial-grade load generating tool called Ruge [13] that is capable of crafting forged and flooding packets and simulating DoS attack loads to intended target networks or hosts. Ruge is a LAN-based tool and hence used directly for attacks in the homenet in which physical access to the router is possible. For wireless attacks, a laptop was used as a relay together with Ruge.

In addition to these components, the setup also consisted of three Energy Monitoring Modules (EMM), each of which were connected to a homenet router. The EMMs were built in-house and monitored the precise power being supplied (both voltage and current) non-invasively to the routers. The design of the EMM was adapted from the Energino energy consumption monitoring toolkit [14], which provides real-time, precise, energy consumption statistics for any DC appliance. Fig 2 shows the constructed EMM.

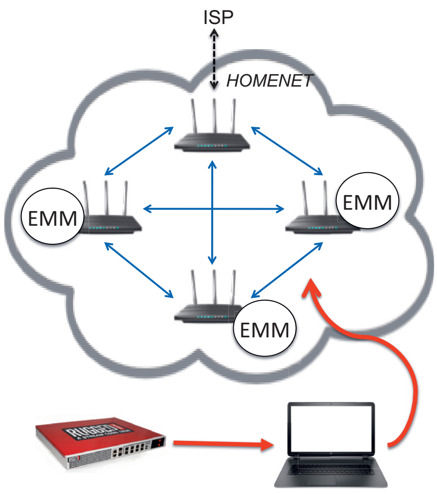


Figure 1: Setup for measuring power consumption. Routers are interconnected in a full mesh topology. Blue arrows indicate authentic Babel traffic. Red arrows depict forged traffic.

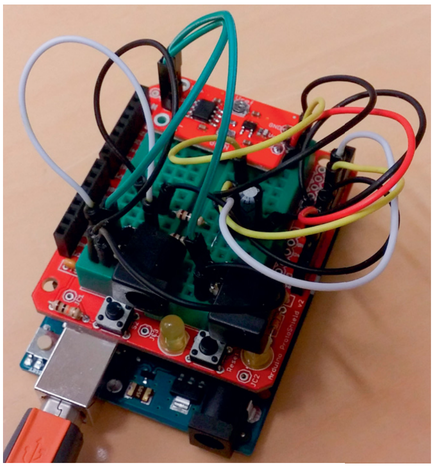


Figure 2: Energy Monitoring Module.

IV. Measurements

This section presents our methodology and measurements per experiment. Various kinds of attacks were targetted at the homenet. In some cases, power measurements were taken when Babel was used without HMAC authentication. In the others, HMAC authentication was enabled, using the SHA1 algorithm. In IETF protocol design, HMAC-SHA-1 is the preferred keyed-hash algorithm [15].

For each experiment, several sets of measurements were taken to ensure data correlation. Each run was conducted once the routers were in steady-state both before and after attacks, and measurements were taken approximately for an hour. Additionally, datasets for the three access points were verified to ensure correct calibration of the EMMs. For each reading taken from the serial interface of the EMM, the data consisted of the average voltage, average current, an approximate sample size of 800 voltage/current samples in the averaging window of a single reading, each for an approximate time of 200ms in the averaging window.

For each set, we start by describing the acquisition of our dataset and then follow with an initial analysis of the results. A deeper discussion of these results is presented in the next section V.

A. Baseline Measurements

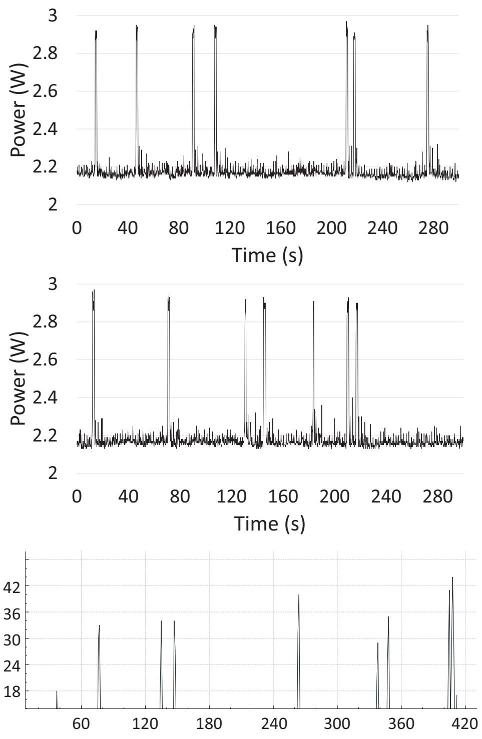


Figure 3: Baseline graphs from top-down. a) Power consumption with Babel, HMAC enabled. b) Power consumption with Babel, no HMAC. c) mDNS traffic triggering peaks in energy consumption. y-axis shows packet count, x-axis shows elapsed time in seconds.

In this experiment, we took detailed power measurements of access points in which no malicious traffic was introduced. By default, Wi-Fi channel security was achieved by the use of WPA2-PSK. Two types of power measurements were taken. In the first instance, the energy was monitored when Babel was used with HMAC authentication enabled, while in the second

Table I: Babel Protocol Messages and Sizes in a 4-router mesh

Babel Packet	No HMAC	With HMAC
Hello	74	106
IHU	122	154
Update	157	189

instance, Babel was used without HMAC. The aim of the experiment was to firstly to study the impact of enabling Babel message authentication on the overall power consumption, and secondly, to establish a baseline measurement with which subsequent experiments and power consumption levels can be compared against.

Figs 3a and b show the power consumption levels for a single router with and without HMAC enabled on Babel. For a 300s duration of the experiment, the average power consumption for running Babel with HMAC enabled, was 2193mW, while that of the same routing protocol without any HMAC was 2192mW.

These two values show that there is virtually no difference, if any, on a homenet router's power consumption, when HMAC authentication is enabled for Babel routing during normal operation. This is so, even when accounted for slightly larger packet sizes as well as computational time to check message authenticity. Table I shows Babel packet types (Hello, I-Heard-U, and Update) and their respective sizes in the mesh network in which every router has 3 neighbours. For larger numbers of routers in the mesh network, the size of the Update packet would correspondingly increase. Consequently, should Babel be running in a fixed Ethernet configuration, Update packets would be smaller in size. As can be seen, with HMAC enabled, the overhead is a constant 32-byte structure to every Babel packet.

Periodic glitches and spikes can also be consistently observed in both sets of measurements. From network traces taken during the measurements, the cause was pinpointed to be large bursts of packets transmitted and received at fixed intervals over the radio interface, as shown in Figure 3c. A specific investigation using Wireshark revealed that these were multicast DNS (mDNS) traffic. mDNS is used in homenets for automatic service discovery. Table II shows the relative amount of traffic seen during a 300s interval corresponding to measurement periods. It can be seen that in total, even if it is bursty in nature, mDNS traffic is significant, accounting for more than half of all traffic, having 232 packets with a total of 84224 bytes. This is compared to 244 Babel packets having a total of 50089 bytes (or 57897 bytes with HMAC authentication enabled). By contrast, HNCP traffic is far less noisy, accounting for a total of 93 packets and 12818 bytes which accounts for between 8% to 9 % of the total traffic.

B. Wireless Channel Attacks on the Mesh Network

We undertook wireless online channel attacks to show the power consumption of routers when Wi-Fi de-authentication attacks are in progress. For Babel routing the homenet routers used a WPA2-based mesh network protected with a strong passphrase. Firstly the wireless traffic was passively monitored

Table II: Protocol, number and bytes sent and received in 300s

Protocol Type	Bytes Sent	Packets Sent	Bytes Received	Packets Received
mDNS	16628	44	67596	188
Babel	18063	98	32026	146
Babel+HMAC	21199	98	36698	146
HNCP	1922	19	10896	74

using the *airodump-ng* packet capture tool in order to obtain the Base Service Set Identifier (BSSID) of an already running mesh network. During the same period, the BSSID of the all communicating routers connected to the mesh were also retrieved.

Subsequently two de-authentication attacks were mounted simultaneously from a laptop using the *aireplay-ng* tool to inject frames, for approximately 6 minutes.

- The first was aimed at disrupting the mesh network operation and causing the mesh network to become unstable, with the routers constantly re-authenticating themselves into the network. This attack was mounted by injecting de-authentication messages using the BSSID of a specific router (router 1) as the source and sending it to the BSSID of the mesh network. The effect of this attack can be seen in Figure 4.

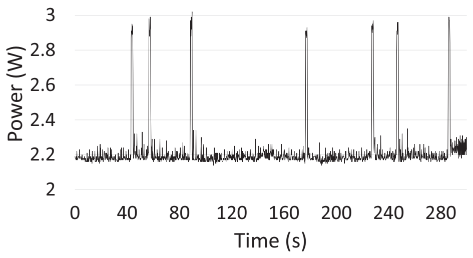


Figure 4: Power consumption of Router 1 during WPA2 de-auth attack.

- In the second attack, in addition to sending the de-authentication messages to the BSSID of the mesh network, they were also sent to the BSSID of a second router (router 2) communicating with router 1. This directly targeted the link and connectivity between the two routers, the aim being to cause an existing connected router to keep rejecting connection attempts from a targeted victim. The effects were then observed on router 2 as seen in Figure 5.

In both cases, routers reflected a higher level of power consumption compared to baseline levels. The average power consumption seen in Router 1 during the Wi-Fi De-authentication attack, was 2216mW, an increase of about 23 mW for each 300s period of the attack. When Router 2 was additionally targetted in the second attack, it exhibited a notably higher increase in consumption of 183mW. This is clearly visible in

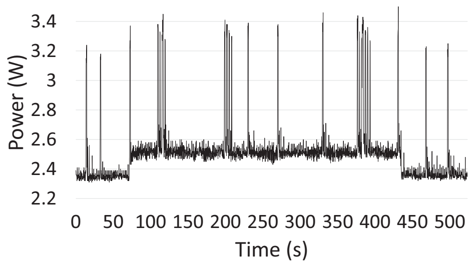


Figure 5: Power consumption of Router 2 during WPA2 deauth attack.

Figure 5 which shows the energy profile of the router before, during and after the attack.

C. Traffic Injection Attack in Babel Network with HMAC enabled

In this experiment, it is firstly assumed that an attacker has penetrated the mesh network itself, either via physical access and tampering with a router, or as an outcome of a successful offline dictionary attack on a weak WPA2-PSK passphrase. We look at the impact to the power consumption of an existing router when an attacker attempts to infiltrate a network in which the Babel routing has been enabled with HMAC authentication, with the HMAC-SHA1 key assumed to be unknown to the attacker. We look at how the response of existing routers can be exhibited in our measurements, when either a malicious router is attempted to be introduced or malformed Babel routing messages are injected into the network.

In order to facilitate our testing and measurement, we used Ruge to craft Babel routing messages. Babel employs Type-Length-Value (TLV) encoding in its packets to exchange routing information. Babel nodes can also solicit Acknowledgement requests for any transmitted packets. For this scenario, Babel Hello and I-Heard-U (IHU) messages were used for the injection attacks. As Ruge is a wireline tool, injecting Babel routing messages wirelessly into the mesh network required the assistance of a laptop to capture generated messages from Ruge over an Ethernet link and save the packet capure traces into a file. The laptop was subsequently used to joining the homenet wireless mesh network. The *packETH* command-line tool wirelessly replayed these generated Hello and IHU messages every 4 seconds in an infinite loop. Fig 6 shows 3 power consumption traces of the same router for three kinds of activity for 300 seconds.

The lowest blue line in this Figure indicates the baseline power measurement of 2193mW, which corresponds directly to Fig 3a. The green line is the obtained measurement when invalid Babel messages without a HMAC have been used in an injection attack. These packets are received but discarded immediately by the router. Here the power consumption is calculated to be 2283mW, an increase of 90mW over non malicious traffic. The red line obtained in the measurement indicate the power consumed by a router receiving, processing and subsequently discarding Babel messages which possess a forged HMAC. Injected Babel packets appended with a forged HMAC TLV structure induce recipients to act on the incoming malicious traffic on a per-packet basis before discarding the forged packets, thereby introducing a packet processing overload onto the existing routers. Consequently, the power consumed on average for this duration was calculated to be 2318mW, an increase of 125mW from the baseline reading, and a slight increase of 35mW caused by the computational overhead from the green line.

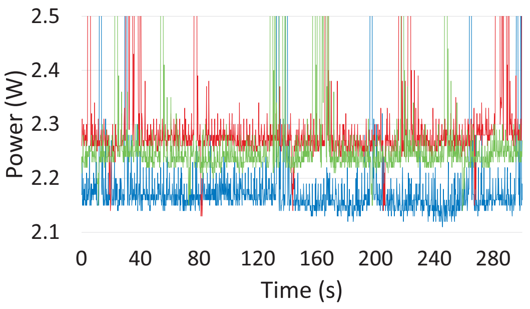


Figure 6: Differences in power consumption in an HMAC-enabled Babel network. Blue line indicates baseline power consumption. Green line indicates malicious Babel traffic with no HMAC packet structure, Red line indicates malicious Babel traffic with forged but invalid HMAC.

D. Route Flooding Attack in a Babel Network with no HMAC

In this experiment we explore the worst case scenario of a homenet which becomes subjected to a flooding attack. This can occur if both the wireless channel security as well as the routing infrastructure become compromised. This scenario is an extension of the previous scenario from subsection IV-C. This could occur, for example, if the home network owner did not secure the Babel routing protocol with HMAC authentication and relied purely on protecting the network using WPA2-PSK. In this sequence of attack, it is assumed that the attacker has penetrated the mesh network itself, either via physical access and tampering with a router, or as an outcome of a successful offline dictionary attack on a weak WPA2-PSK passphrase.

In addition to the Babel Hello and I-Heard-U (IHU) messages from the previous subsection, Update messages were also generated from Ruge and used in this attack to flood the network. The messages which were generated in Ruge for flooding into the mesh were used in the following sequence of events:

1) When the attack commences, a Babel Hello packet is sent into the network
2) After a 4 second wait, an IHU packet is sent.

3) After a 1 second wait, steps 1 and 2 are repeated in an endless loop.
4) After an initial 12 second wait from step 1, Update messages are constantly flooded into the network every 3 ms. Each message updates current routes for the IPv4 /24 network by increasing the network address by a single bit for each transmission. Updates start at 10.0.0.0/24. One network advertisement is done every 3ms.
5) Sending route update messages for the entire /24 IPv4 network takes 15 seconds after which sending Updates starts anew.

With the above steps, routing tables and information for the entire homenet mesh was updated constantly, inducing a heavy stress and CPU load onto the routers.This forces routers to consider between 200 and 650 routes, depending on the router load status. As before, a laptop was used as a wireless relay injecting malicious traffic into the network. Hello and IHU messages were transmitted using *packETH* while Update messages were wirelessly replayed using the *tcpreplay* tool. Because the malicious Babel traffic was injected into the homenet where routing was not secured using HMAC authentication, all the existing homenet routers were unaware that the traffic was forged. Fig 7 graphically depicts the resulting router's energy consumption profile. From its baseline power consumption of 2192 mW, the router begins consuming an average of 2802 mW during the attack. The graph also shows an interval during the sequence of events where there is a momentary pause before the flooding attack replay loop resumes, and reaches a similarly consistent state of high power consumption for a second time during the attack.

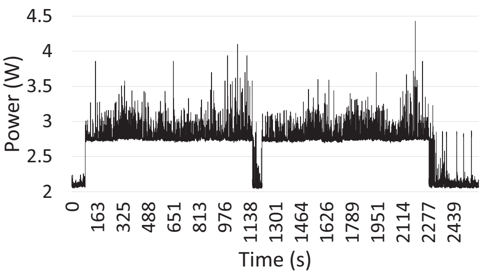


Figure 7: Power consumption during route injection attack, no HMAC enabled

V. Findings

Based on our measurements and observations, the obtained results remained consistent across all our experimental runs. The experiments conducted focused solely on the power consumption profiles of the routers working with Babel in wireless ad-hoc mode. However, we wanted to ensure that mandatory features of the Homenet Architecture, and the components which implemented them were not disabled at any stage of our experiments. One such component was mDNS, which implements service discovery. From our findings, mDNS consumed a significant portion of energy, even when minimal services existed in the homenet.

Additionally, because all homenet routers participate in multicast-based packet transmission and reception, it is highly likely that as more routers are added to a mesh-based homenet, the increase of multicast traffic could be in the order of *O(n)*. Implementation optimisations, protocol improvements, tuning of mDNS advertisements and broadcast intervals may yield better energy efficiency. However, investigation of such aspects are out of scope of this paper.

The approach of using power consumption as a metric for attack detection in homenet appeared promising and feasible. It can be seen that various kinds of attacks had a definite impact on the energy consumed by a homenet router. With flooding attacks, significant differences were seen, compared to the base level measured. However, even for Wi-Fi based authentication attacks, while the measured levels of additional power consumed were relatively small, it was noticeable that because of the nature of the mesh topology, the power consumption pattern of more than 1 router was normally affected by attacks targetting even a single homenet router. Consequently, it would be possible to observe the power consumption patterns and perform correlation across several routers to detect an attack in progress, even if individual power consumption levels by themselves may not give a clear indication. Particularly for covert attacks, using power consumption anamolies can be quite effective. On the other hand, our findings note that a homenet cannot only rely on homenet routers enabling HMAC authentication for Babel routing, without having a proper wireless channel security, for example by having a WPA2-PSK wireless network with a weak passphrase. When energy is an asset that needs to be conserved, then such a practice would be counter-productive and exposes an additional attack surface, as in addition to injecting malicious Babel traffic, an adversary can append a forged HMAC to each injected packet, forcing a router to both process the injected message as well as perform computations on the forged HMAC. An additional point of input is that homenet-based configurations are also suitable for deploying ad-hoc mesh and wireless sensor networks where gateways are more resource constrained in terms of computational ability as well as power. Using energy footprints, power draining attacks, which force a battery-operated router into consuming significantly more energy thereby reducing its viable performance and lifetime, can be similarly detected.

For the EMMs, we saw several advantages in developing our own implementations based on low-cost but highly accurate current and voltage sensors. For example, our approach allowed the ability to non-invasively monitor the power consumption of the routers without having to install any special hardware or software on the router itself. Additionally, EMMs can be colocated in multiple locations without being physically encumbered in a specific location, as each setup is portable and can be easily powered using an external battery supply. However, significant amount of time was necessary for sensor calibration, and ensuring consistency by double-checking measurements across the several homenet routers

Table III: Theoretical lifetime values for routers in operation on batteries

Router Model Voltage Battery Capacity	TP-Link AC1200 12V 20000 mAh	TP-Link MR3020 5V 12000 mAh
Babel + WPA2-PSK (No Attacks)	1389 h	1200 h
Wi-Fi De-authentication	1370 h	822 h
Packet Injection no HMAC appended	1333 h	800 h
Packet Injection Forged HMAC appended	1316 h	790 h
Routing Flood Attack	1087 h	652 h

equipped with EMMs in our setup. As an EMM is based on a Hall-Effect current sensor, it was suspectible to magnetic influence that had an effect on the output values and as such could introduce some variation to the measurements. The other issue effecting the measurement accuracy is the electrical noise. The RC-filter added to the module lowered the noise on the ACS712 output/ADC input and stabilised the readings.

Finally, even though the measured power consumption levels were based on a specific hardware model, we performed quick verifications that these results could be extrapolated and applied in other router platforms as well. As an example, we measured the base power consumption of a TP-Link MR-3020 homenet router. This is based on a resource-constrained design, having a very small form-factor, 32MB of RAM and 4 MB of flash memory, whose main purpose is to serve as a personal temporary hotspot or a wireless repeater. The router was then connected to an external 12000 mAh battery pack, flashed with OpenWRT and extended for use as a battery operated homenet router. Using the values derived from our measurements, we were able to calculate theoretical lifetime values for both the AC1200 router used throughout this paper, as well as that of the MR3020, if the entire power was solely battery-based and all the energy from the external batteries were used in keeping a similar-sized homenet mesh network up and connected without any clients connected. This is shown in Table III. The expected lifetime of these routers when under attack are also calculated, based on figures obtained from our power measurements. Real figures however would be far lower, owing particularly to home network traffic and Internet usage patterns from end devices and services. Additionally, our assumption is that commonly used home routers would lack dedicated hardware support for cryptographic functions.

VI. Conclusions

Anomalous power consumption is an emerging research area, aimed at highlighting unusual ongoing activity which can escape detection from conventional methods. Our hypothesis is that by calibrating and profiling the power consumption of a homenet router during normal operations, a higher than likely possibility of an intrusion in progress (such as a brute force attack) can be detected resulting from high radio or CPU activity, or higher-than-normal gateway activity during quiet hours. The work done in this paper shows that using power measurement as a metric for attack detection is feasible for both novice and professional technical network users.

In future, we intend to continue developing our Energy Monitoring Modules so that sensors with higher accuracy could be incorporated and obtained measurements can become more precise. At the same time, sensitivity could be increased to detect malicious behaviour which do not consume significant amounts of power. We intend to profile additional attacks on the homenet infrastructure with a wider variety of router hardware thereby corroborating theoretical lifetime values of constrained routers. Future work would consider other mesh topologies with regards to security in homenets. User generated traffic in the homenet and its impact on detecting malicious activity on the routing also remains an area for future work.

VII. Acknowledgements

This work is funded by the EIT Digital HII Active Project.

References

[1] G-W. Kim, D-G. Lee, J-W. Han, S-C. Kim, and S-W Kim. "Security framework for home network: Authentication, authorization, and security policy." In Emerging Technologies in Knowledge Discovery and Data Mining, Springer LNCS Vol 4819, pp. 621-628, May 2007.

[2] M. Abid. “User identity based authentication mechanisms for network security enhancement”. Ph.D. Dissertation, Institut National des Telecommunications, 2011.

[3] S. Qu, and H. Liu. "Security Research on the Interfaces of Information Appliances Described by XML." In Network Computing and Information Security, Springer Vol 345, pp. 661-668, 2012.

[4] L. DiCioccio, R. Teixeira, and C. Rosenberg. "Measuring home networks with homenet profiler." In Passive and Active Measurement, Springer LNCS Vol 7799, pp. 176-186, 2013.

[5] T. Martin, M. Hsiao, Dong Ha and J. Krishnaswami, "Denial-of-service attacks on battery-powered mobile computers," Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on, 2004, pp. 309-318.

[6] H. Kim, J. Smith, and K. G. Shin. "Detecting energy-greedy anomalies and mobile malware variants." In Proceedings of the 6th ACM international conference on Mobile systems, applications, and services, pp. 239-252. 2008.

[7] X. Ma, P. Huang, X. Jin, P. Wang, S. Park, D. Shen, Y. Zhou, L. K. Saul, and G. M. Voelker. "eDoctor: Automatically diagnosing abnormal battery drain issues on smartphones." In Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pp. 57-70. 2013.

[8] T. K. Buennemeyer, M. Gora, R. C. Marchany and J. G. Tront, "Battery Exhaustion Attack Detection with Small Handheld Mobile Computers," Portable Information Devices, 2007. PORTABLE07. IEEE International Conference on, Orlando, FL, 2007, pp. 1-5.

[9] T. Chown, Ed., J. Arkko, A. Brandt, O. Troan and J. Weil, "IPv6 Home Networking Architecture", IETF RFC 6126, Oct. 2014.

[10] J. Chroboczek, The Babel Routing Protocol, IETF RFC 6126, Apr. 2011.

[11] M. Stenberg, S. Barth, and P. Pfister. "Home Networking Control Protocol", IETF RFC 7788, Apr 2016.

[12] D. Ovsienko."Babel Hashed Message Authentication Code (HMAC) Cryptographic Authentication”, IETF RFC 7298, Jul 2014.

[13] Ruge. Rugged IP load generator, Rugged Tooling. http://www.ruggedtooling.com

[14] The Energino project. http://www.energino-project.org

[15] T. Polk, L. Chen, S.Turner, and P. Hoffman. “Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms”, IETF RFC 6194, Mar. 2011.