Provision of Adaptive and Context-Aware Service Discovery for the Internet of Things

by

Talal Ashraf Butt

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

18th October 2013

Abstract

The Internet of Things (IoT) concept has revolutionised the vision of the future Internet with the advent of standards such as IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) making it feasible to extend the Internet into previously isolated environments, e.g., Wireless Sensor Networks (WSNs). The abstraction of resources as services, has opened these environments to a new plethora of potential applications. Moreover, the web service paradigm can be used to provide interoperability by offering a standard interface to interact with these services to enable Web of things (WoT) paradigm. However, these networks pose many challenges, in terms of limited resources, that make the adaptability of existing IP-based solutions infeasible. As traditional service discovery and selection solutions demand heavy communication and use bulky formats, which are unsuitable for these resource-constrained devices incorporating sleep cycles to save energy. Even a registry based approach exhibits burdensome traffic in maintaining the availability status of the devices. The feasible solution for service discovery and selection is instrumental to enable the wide application coverage of these networks in the future.

This research project proposes, Trend-based Service Discovery Protocol (Proposed Solution) (TRENDY), a new compact and adaptive registry-based Service Discovery Protocol (SDP) with context awareness for the IoT, with more emphasis given to constrained networks, e.g., 6LoWPAN. It uses Constrained Application Protocol (CoAP)-based light-weight and Representational state transfer based (RESTful) web services to provide standard interoperable interfaces, which can be easily translated from Hyper Text Terminal Protocol (HTTP). TRENDY's service selection mechanism collects and intelligently uses the context information to select appropriate services for user applications based on the available context information of users and services. In addition, TRENDY introduces an adaptive timer algorithm to minimise control overhead for status maintenance, which also reduces energy consumption. Its context-aware grouping technique divides the network at the application layer, by creating location-based groups. This grouping of nodes localises the control overhead and provides the base for service composition, localised aggregation and processing of data. Different grouping roles enable

the resource-awareness by offering profiles with varied responsibilities, where high capability devices can implement powerful profiles to share the load of other low capability devices. Thus, it allows the productive usage of network resources. Furthermore, this research project proposes Adaptive Piggybacked Publishing (APPUB), an adaptive caching technique, that has the following benefits: it allows service hosts to share their load with the resource directory and also decreases the service invocation delay.

The performance of TRENDY and its mechanisms is evaluated using an extensive number of experiments performed using emulated Tmote sky nodes in the COOJA environment. The analysis of the results validates the benefit of performance gain for all techniques. The service selection and APPUB mechanisms improve the service invocation delay considerably that, consequently, reduces the traffic in the network. The timer technique consistently achieved the lowest control overhead, which eventually decreased the energy consumption of the nodes to prolong the network lifetime. Moreover, the low traffic in dense networks decreases the service invocations delay, and makes the solution more scalable. The grouping mechanism localises the traffic, which increases the energy efficiency while improving the scalability. In summary, the experiments demonstrate the benefit of using TRENDY and its techniques in terms of increased energy efficiency and network lifetime, reduced control overhead, better scalability and optimised service invocation time.

Acknowledgements

This research project for me is like an adventurous journey, which guided me to discover myself. I am grateful to Almighty ALLAH that He gave me this opportunity to learn, think critically, investigate, evaluate concepts and ideas. Furthermore, I am thankful to a number of people who have guided and supported me throughout the research process and provided assistance for my venture.

Foremost, I would like to express my sincere gratitude to Dr. Iain Phillips and Dr. Lin Guan, my supervisors, whose selfless time and care were sometimes all that kept me going. In addition, their out of office hours meetings and favours will always keep me indebted. Furthermore, their provoking attention to detail drove me to learn many key skills, which I appreciate the most. I feel being a kid at the start of my research and then raised up by both of them, who have taught me that how devotion, enthusiasm and patience are important to achieve a goal.

I would like to thank Dr. George Oikonomou and Dr. Shafique Ahmad Chaudhry for their support. Both of them lifted my spirit by giving in more informal and personal meetings. I would like to thank Loughborough University and specifically to Department of Computer Science for funding my research study, as my dream to become a researcher is materialised by their support. Moreover, I am indebted to the financial support of Churches International Student Network (CISN), The Leche Trust and The Charles Wallace Pakistan Trust. Furthermore, I am grateful to the Department's staff, who facilitated my research. I would like to extend my gratitude to fellow research students and staff for making this journey a memorable one.

I dedicate this research work to my beloved parents and wife, who constantly encouraged me and motivated me to believe in myself. I want to admit that I could've never made it, if my father hadn't dreamt about it. Furthermore, I want to dedicate my work to ABBA jee and his family, who always believed in me and nurtured my potential. In addition, I want to thank my beloved Baba Muhammad Yahya Khan for guiding me to understand the reason of my existence. Finally, I want to thank all my family members and friends, as they have supported and helped me along the course of this research project by providing the moral and emotional support.

Publications

- **Talal Ashraf Butt, Iain Phillips, Lin Guan and George Oikonomou.** "TRENDY: An Adaptive and Context-aware Service Discovery Protocol for 6LoWPANs." In Proceedings of the third international workshop on the web of things (WoT 2012), ACM.

- **Talal Ashraf Butt, Iain Phillips, Lin Guan and George Oikonomou.** "Adaptive and Context-aware Service Discovery for the Internet of Things." In the 6th conference on Internet of Things and Smart Spaces (ruSMART 2013), Internet of Things, Smart Spaces, and Next Generation Networking, Lecture Notes in Computer Science, Springer Berlin Heidelberg.

Contents

Acronyms

6LoWPAN IPv6 over Low power Wireless Personal Area Networks

ACK Acknowledgement

AES Advanced Encryption Standard

AODV Ad-hoc On-Demand Distance Vector Routing

APPUB Adaptive Piggybacked Publishing

BXML Binary XML

CoAP Constrained Application Protocol

CoRE Constrained RESTful Environments

CSMA Carrier Sense Multiple Access

DA Directory Agent

DAO DODAG Advertisement Object

DHCP Dynamic Host Configuration Protocol

DIO DODAG Information Object

DIS DODAG Information Solicitation

DNS Domain Name System

DODAG Destination Oriented Directed Acyclic Graph

EUI-64 64-bit Extended Unique Identifier

EXI Efficient XML Interchange

FWD Forward Query

GL Group Leader

GLD Group Leader Done

GM Group Member

HTTP Hyper Text Terminal Protocol

IETF Internet Engineering Task Force

IoT Internet of Things

IS-IS Intermediate System to Intermediate System

LLN Low Power and Lossy Network

M2M Machine-to-Machine

MAC Medium Access Control

MANET Mobile Ad-hoc Network

MIME Multipurpose Internet Mail Extensions

MTU Maximum transmission unit

NAPTR Name Authority Pointer

NRP Not Reported

OF Objective Function

OGC Open Geospatial Consortium

OLSR Optimized Link State Routing Protocol

OSPF Open Shortest Path First

PubSub Publish and Subscribe

QoS Quality of Service

RDC Radio Duty Cycling

RDF Resource Description Framework

REST Representational state transfer

RESTful Representational state transfer based

RGM Remove Group Member

RPC Remote Procedure Call

RPL IPv6 Routing Protocol for Low power and Lossy Networks

RQL RDF Query Language

RTCP RTP Control Protocol

RTP Real-time Transport Protocol

SA Service Agent

SD Service Discovery

SDP Service Discovery Protocol

SDR Service Discovery Request

SIP Session Initiation Protocol

SLIP Serial Line Internet Protocol

SOAP Simple Object Access Protocol

SSC Some Service Changed

SSDP Simple Service Discovery Protocol

TCP Transport Control Protocol

TRENDY Trend-based Service Discovery Protocol (Proposed Solution)

UA User Agent

UDP User Datagram Protocol

UPD Update

uPnP Universal Plug and Play

WADL Web Application Description Language

WBXML WAP Binary XML

WoT Web of things

WSDL Web Service Description Language

WSN Wireless Sensor Network

WSNs Wireless Sensor Networks

YGL Your Group Leader

YGM Your Group Member

Chapter 1

Introduction

The IoT vision has drastically changed the way we foresee the future Internet [10]. Integration of constrained networks with the broader Internet can open new avenues for these networks, as this extends the ability of these networks to share resources and information with other networks. These low cost, battery operated devices with limited data rates are becoming smarter and making this kind of integration more feasible.

Arrival of new standards has allowed constrained networks to integrate seamlessly with the Internet. IP-based devices can be connected easily to other IP networks, without the need for complex translation gateways. This gives the opportunity for the reusability of existing infrastructure and proven standards, instead of using proprietary solutions. Moreover, IP connectivity enables WSNs to utilise the broad body of existing IP tools and standards such as firewalls, proxies, caches. Furthermore, the use of IP outperforms existing systems in average duty-cycle, per-hop latency, and data reception rate with a higher traffic load [56]. However, the inherent nature of these networks still poses many challenges in adapting existing IP-based standards.

Devices can offer their functionalities by abstracting them as services. Moreover, the provision of Service Discovery (SD) and service selection can assist the user applications to find the required services. This changes the traditional view of these constrained networks by making them directly accessible over the Internet. Furthermore, web services can provide a standard interface to offer interoperability with other existing similar solutions. However, the design of such a solution faces the underlying issues of these networks, for example, small packet size and sleep cycles. In constrained environments, services are mostly hosted on battery operated devices, which can fail without any notification. Thus, these networks require an efficient, reliable and interoperable solution for SD. This research project analyses existing solutions with the IoT requirements and challenges of its constrained domains, to identify the potential gaps. Consequently, the project proposes a

compact, adaptive and context-aware SD solution that considers the inherent challenges, and offers the desired features to encompass them.

This chapter highlights the significance of the research problems and related challenges, and then summarises the original contributions made by this project.

1.1 Motivation

The Internet Engineering Task Force (IETF)'s 6LoWPAN [93] standard has made it possible for constrained networks to connect directly to the Internet. This integration has opened new horizons for these devices. Recent developments have changed the profile of WSNs from isolated and application-specific, to more interconnected and directly accessible 6LoWPANs. However, these networks still have constrained properties, including limited packet size, intermittent communication, high packet loss, restricted power and throughput. Besides these challenges, the major benefit of integration is resource sharing, which can be achieved by abstracting the resources as services. A service can provide any data reading or measurement such as temperature reading of a sensor, which can be shared inside a 6LoWPAN or with external IP network. Services can be further composed to create higher-level services. Furthermore, mashups can be generated to provide new functionality with the combination of different services [136]. To make use of all potential benefits; discovery of available services is highly crucial and demands suitable SD, and selection solution.

The employment of existing IP-based SD solutions is a first choice to enable the inherent interoperability. However, the large packet sizes and heavy control overhead of these solutions make these infeasible for constrained networks such as 6LoWPANs. The difference of both networks can be understood by their Maximum transmission unit (MTU), which is 1280 bytes for IPv6 and is only 127 bytes for 6LoWPAN (which uses IEEE 802.15.4). This implies that even in the case of single IPv6 packet transmission, a 6LoWPAN network needs to send multiple packets, which will entail heavy traffic load and bandwidth utilisation. Similarly other available distributed solutions demand excessive use of broadcast or multicast, which is unsuitable in networks with devices with sleep cycles and intermittent connectivity.

A 6LoWPAN based network mostly consists of heterogeneous devices; therefore, a solution should be able to deal with the heterogeneity. Web services offer the solution with high interoperability, customisation and flexibility by providing a standard interface to services. These characteristics make web services a key technology to enable WoT vision that depicts a view where a collection of web services that could be discovered, composed and executed [136]. Although web

services remarkably change the way how services are accessed, but are a poor match for constrained networks because of the inherent complexity [114]. Both Simple Object Access Protocol (SOAP) based big web services, and Representational state transfer (REST) in their traditional forms are impractical for such networks. However, constrained nodes can benefit from IETF's CoAP [13] standard, which offers compact and optimised RESTful web services.

The IoT vision has significantly changed the way we approach the solution for a Wireless Sensor Network (WSN). Figure 1.1 shows the IP stack with some 6LoWPAN related protocols from different perspectives.

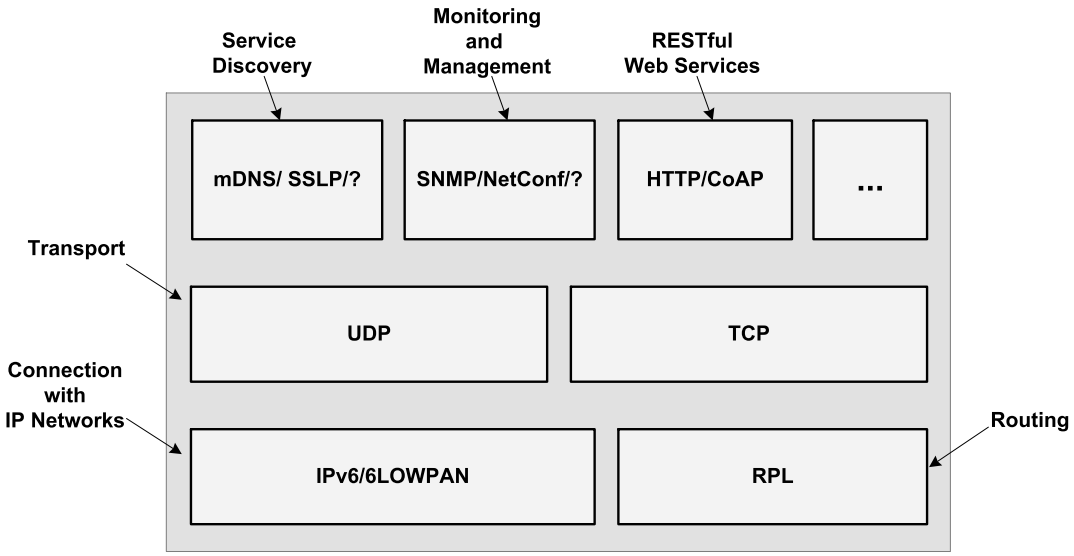


Figure 1.1: Protocols for 6LoWPANs from different perspectives

1.2 Research Challenges

This thesis focuses on provision of adaptive and context-aware SD and selection for IoT environments while accounting for challenges, including heterogeneity, interoperability, scalability and energy efficiency. Following are the research questions which raise the research challenges addressed by this thesis:

Research question 1: *What constitutes a better SD solution for the IoT compared to existing solutions?*

IoT environments contain heterogeneous networks and devices, which pose new challenges and requirements for a SD solution. This merger of constrained networks with the IoT for example, 6LoWPANs, requires a special emphasis.

Thus, a review of existing SD solutions is required in the context of these challenges.

- *Literature review of existing SD solutions:* A critical analysis of existing SD solutions and discussion of their applicability for constrained domains is required to find out the research gaps and to take better design decisions.
- *The IoT requirements for a SD:* Clear understanding of the new IoT perspective is needed to be analysed in the context of future scenarios. The general and constrained network specific SD requirements and challenges are needed to be scrutinised and well defined.

Research question 2: *How IP-based WSNs consisting of resource-constrained nodes can offer efficient and interoperable SD and selection to user applications in the IoT?*

New technologies have enabled WSNs to integrate with the Internet, but work still needed to make them an active part of the web. The discovery of the available services in a WSN is important to access the functionality within a dynamic environment. Therefore, the mechanism for discovery of services is essential to assist the user to discover required services. A user application query for SD can result in a list of matching services, even though, application actually looking for one appropriate service. A SDP should address this requirement by offering context-aware service selection as this is a norm in IoT scenarios. A discovery depending on multicast or broadcast is not an efficient choice in terms of efficiency and sometimes error prone in networks with intermittent connectivity. Consequently, challenge is to design an interoperable SD and selection solution that satisfy the requirements of the IoT, while providing the required efficiency to meet the challenges posed by constrained nodes and networks.

- *Interoperability:* Web service interface is always preferable to allow the desired interoperability in the IoT; however, the challenge is to enable this paradigm for its constrained domains. The WoT vision has recommended the use of RESTful web services as a standard interface for the services hosted by smart devices [43]. However, offering the web service interface using traditional solutions put more burden on constrained nodes, and is not feasible for many low capacity nodes and networks.

Research question 3: *How a SD solution can become context-aware and enable service composition without creating heavy traffic load on the network?*

The IoT user applications can look for one best service to issue queries or to execute a command on a collection of nodes sharing some context (for example, switching the actuators on in some area). The nodes in a WSN exist in an environment where collaboration between them can reduce the overhead, and consequently, enable the creation of new high-level services. However, the challenge is to enable such a functionality in new IoT paradigm with diverse networks that have constraints. The SD solution needs to find those common context attributes before applying some mechanism to collectively invoke services. The collection of context information and approach to enable service composition should be efficient for constrained domains of the IoT.

Research question 4: *How to deal with mostly unavailable nodes (because of sleep cycles) and high number of service invocations by the users?*
Energy is a precious resource for constrained wireless sensor nodes; therefore, these nodes use different Radio Duty Cycling (RDC) algorithms to save energy. Furthermore, IP-based WSNs offer services to the broad range of users and applications, so there is a potential for an overwhelming number of requests. The sheer number of requests face more delays when nodes keep on sleeping for most of the time, and consequently more energy is consumed by devices in a multi-hop network. The question arises that how this trade-off of energy conservation and better user response can be managed.

1.3 Aims and Objectives

The main aim of this research project is to design a compact SD solution for the IoT with low control overhead, minimised energy consumption and reduced SD and invocation delay. New technologies, including, 6LoWPAN, are shifting the existing paradigm of the Internet by allowing resource-constrained networks to become an active part of the IoT. The target of this work is to analyse and cover the new challenges for SD posed by this new paradigm. However, these networks and devices are still resource-constrained and require an efficient and compact SDP. The solution needs to balance the trade-off of interoperability and efficiency while ensuring its applicability for a wide variety of networks in the IoT. The objectives of this research project are:

- To investigate the SD process and scrutinise design options available at different stages.
- To explore existing SDPs in literature, categorise them and examine their features.

- To scrutinise existing research efforts for providing SD for 6LoWPANs, and the research gaps left by those schemes.
- To investigate the SD requirements for the IoT in context of some use-case scenarios, so the design of proposed protocol can cover new challenges.
- To design and test new SD solution that:

    1. has *open and extensible framework* to address the diverse capabilities of the networks in the IoT.
    2. should consider *interoperability* with an existing approach.
    3. should *address new requirements* posed by the IoT.
    4. should allow context-aware SD by allowing sophisticated queries.
    5. should provide context-aware service selection to assist user applications for choosing more efficient hosts.
    6. is *independent of underlying protocols* at different layers, including, routing, Medium Access Control (MAC) and RDC etc.
    7. should have *compact* implementation and message sizes.
    8. is *efficient* in terms of control overhead, energy consumption and SD and invocation delay.
    9. should be adaptive to network dynamics, so can provide better user response even during high demand.
    10. should provide *reliable and accurate* information, including, service information and cached values.
    11. is *scalable*, by conforming that the control overhead is not proportional to increasing number of devices.

1.4 Research Methodology

Research gap identification is the first stage of this work, which consists of literature review and new IoT requirements gathering. Firstly, the concepts related to emerging IoT environments are developed to identify the basic requirements and challenges posed by its constrained sub-domains. This is followed by an extensive literature review to analyse existing SD protocols. The protocols are categorised and then scrutinised for different stages of SD. The identified IoT SD requirements are used in the scrutiny. The design guidelines is the outcome of this stage for the conception of a better design for new solution.

The solution design stage proposes a solution to achieve the objectives (Section 1.3) according to the design guidelines produced in the first stage. The protocol is implemented and tested in the experiments stage. The results of the evaluation are fed back to the design stage to tackle the unsolved issues. Consequently, new algorithms are devised to improve the solution and to make it more feasible, efficient and suitable for IoT environments. These algorithms are integrated with the existing solution and tested individually. The generated results are used to improve the proposed mechanisms. Finally, the solution composed of all algorithms is tested, verified and validated at experiments stage before finalising the design stage. Figure 1.2 illustrates all stages in a simplified flowchart.

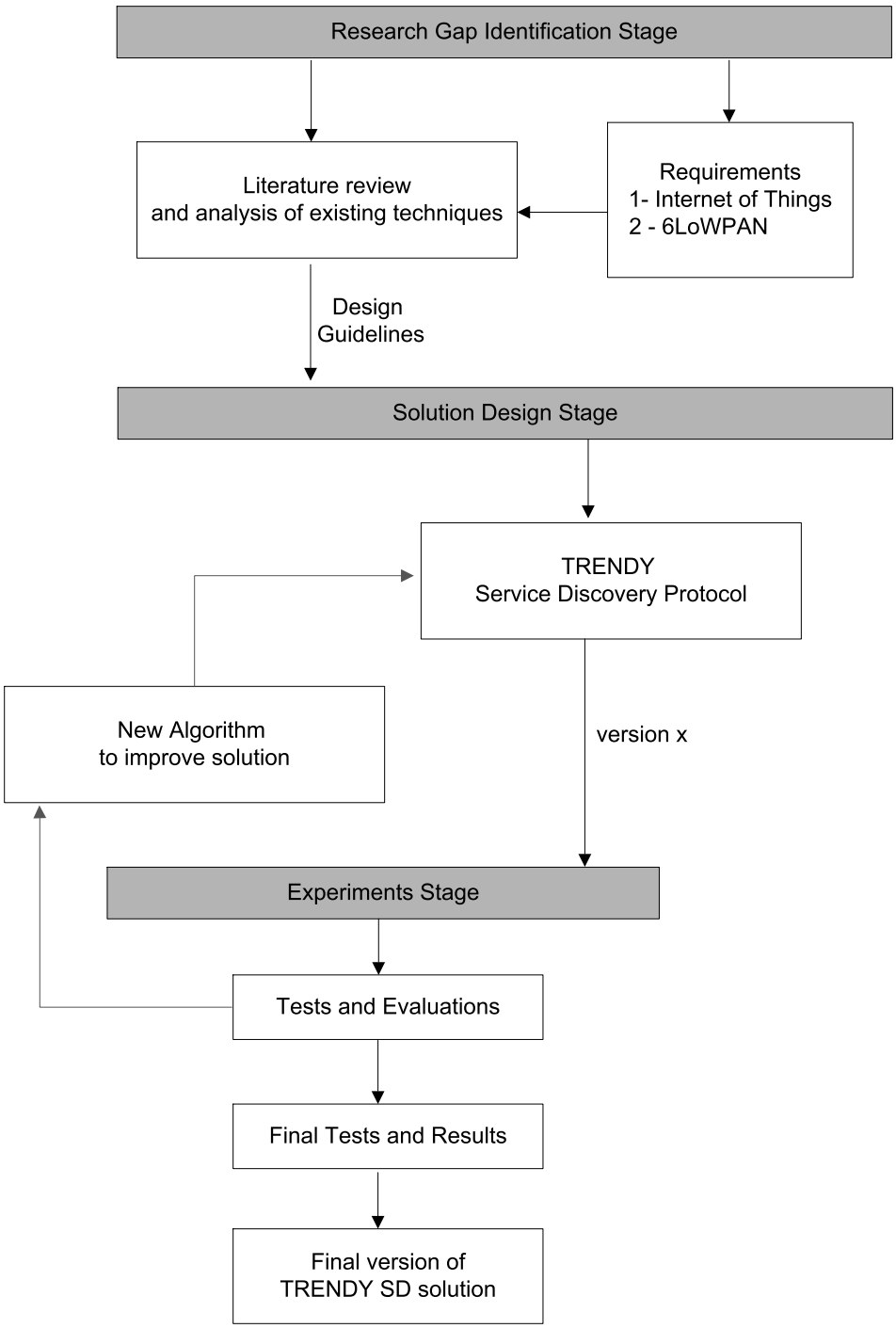


Figure 1.2: Research Methodology

1.5 Original Contributions

In this section, the main contributions of this thesis are described with regard to the posed research questions and challenges.

Contribution 1: *Analysed SD requirements for the IoT*

The demands of IoT environments are very diverse. However, to devise a better solution a review of existing solutions is needed by focusing on the requirements of IoT and its constrained sub-domain. At first, Chapter 2 covers the detail of IoT environments by explaining related technologies. Subsequently, a literature review of existing SD solutions in context of 6LoWPANs is conducted in Chapter 3. This chapter also discusses the details of general challenges and new perspectives in context of SD. Furthermore, Chapter 4 describes some IoT use case scenarios and emphasises the general IoT and specific 6LoWPAN requirements for SD.

Contribution 2: *Proposed a SD and selection solution for IP-based Wireless sensor networks*

This project proposes TRENDY, a context-aware SD and selection solution for IP-based WSNs (Chapter 5), which uses the context information of nodes to select the best matched service. Its service selection mechanism is evaluated in different experiments using different RDCs and topologies (Section 6.5). Moreover, CoAP is employed as a communication protocol that enables the web service paradigm to provide service invocation and desired interoperability.

Contribution 3: *Proposed an adaptive timer to reduce control overhead*

An adaptive timer technique is proposed (Chapter 7) to minimise the overhead of status maintenance. It uses the service popularity (number of times a service is discovered in a time window) as criteria to adaptively increase or decrease the interval between status updates. The timer mechanism is evaluated in the number of experiments using different number of nodes and topologies.

Contribution 4: *Proposed a grouping technique to provide scalable architecture and to enable service composition*

Chapter 8 covers the detail of a grouping approach, which uses the context information to group nodes. It creates an application-level overlay of the network to localise the status maintenance and provide basis to enable service composition while localising the traffic within a group. It makes the solution resource-aware by defining distinctive roles, which consist of

different modular features to deal with the heterogeneity. This allows nodes to implement different level of functionality depending on their capabilities. An evaluation is done to analyse the impact of grouping w.r.t. the size of a group, and the number of groups in a network.

Contribution 5: *Proposed adaptive caching technique to deal with sleepy nodes*
An adaptive caching technique is proposed by this research project (Chapter 9) that reduces the burden on network's bandwidth and delay in service invocations. This technique helps the nodes to deal with the high number of service requests while conserving energy with sleep cycles. Furthermore, it's cache publishing ensures that the delay of service invocations tend to zero when the number of request increases drastically. This technique is evaluated using different network loads, RDCs and topologies.

1.6 Thesis structure

This thesis begins with the introduction of those technologies that have enabled constrained networks to become an active part of the Internet. Chapter 2 introduces the technologies needed to merge WSNs with the Internet. This chapter covers the discussion of existing application protocol paradigms and their integration in constrained networks to enable the WoT paradigm. Chapter 3 introduces the role of SD by explaining its objectives and entities. The literature is reviewed in this chapter, by classifying existing SDPs by architectural design. Furthermore, it covers the related issues and some new perspectives in the SD domain. Chapter 4 presents the IoT SD requirements in the context of different future scenarios with an emphasis to 6LoWPANs. Chapter 5 describes the SD and selection solution proposed by this research project. This chapter discusses the design details of the protocol, including aims, architecture, and protocol's overview. The protocol specifications including details of algorithms and message formats are also explained in this chapter. Chapter 6 presents the experimental tools, performance metrics, experimental setup and design for evaluation of the solutions. It also covers the evaluation of the service selection mechanism of TRENDY. Chapters 7, 8 and 9 present the detail of three proposed techniques: adaptive reporting timer, context-aware grouping and adaptive caching. These chapters cover the detail of design, performed experiments and generated results of the respected techniques. Chapter 10 combines the four contributions together to form an adaptive and context-aware SD solution for the IoT. This chapter also discusses the experiments performed and generated results by combining different techniques. The proposed solution is then compared with some existing SD solutions in Chapter 11. The

thesis is concluded with the discussion of the research project's contributions and future work in Chapter 12.

Chapter 2

From the Internet of Things (IoT) to the Web of Things (WoT)

2.1 Introduction

The IoT vision extends the current Internet to previously unconnected physical things. These everyday objects are connected to the virtual world and empower remote users to control them. This paradigm transforms the computing truly ubiquitous - an idea coined by Mark Weiser [131]. New technologies have made this integration a reality by enabling even very low capable devices and isolated WSNs to become an active part of the Internet. The WoT [42] paradigm extends this integration to the application layer [45]. This chapter explains the technologies, which enable constrained networks to become the part of the Web. The details of different application protocol paradigms and related protocols are covered in this chapter. In the end, it describes the employed technologies by this research project for implementation and evaluation.

2.2 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks)

6LoWPAN [90] is an IPv6 adaptation layer that defines mechanisms to connect resource constrained devices with IP [76]. These devices mostly communicate over low power, lossy links such as IEEE 802.15.4. It uses a compression format [55] to compress the IPv6 packets. The adaptation layer is introduced at the edge router, for translation of packets to and from IPv6 network.

A 6LoWPAN consists of one or many stub networks. A stub network is a small network to which packets can be sent or received, but it doesn't behave as a transit to other networks. The connection to other IP networks is maintained

by edge routers as shown in the Figure 2.1. The edge router plays a pivotal role in 6LoWPANs as it routes the traffic in and out; additionally, it handles the 6LoWPAN neighbour discovery, compression and IPv4 inter-connectivity. A typical 6LoWPAN can consists of one or more edge routers, and nodes with host or router roles. In case of multiple edge routers in the same LoWPAN, all edge routers need to share a common backbone link.

2.2.1 Architecture

The architecture of a 6LoWPAN network consists of edge-router(s), router and servers, as shown in Figure 2.1. The nodes send their data out from the network via edge-routers. The edge-routers can also act as a proxy to allow the use of new compressed application protocols and can enable caching.

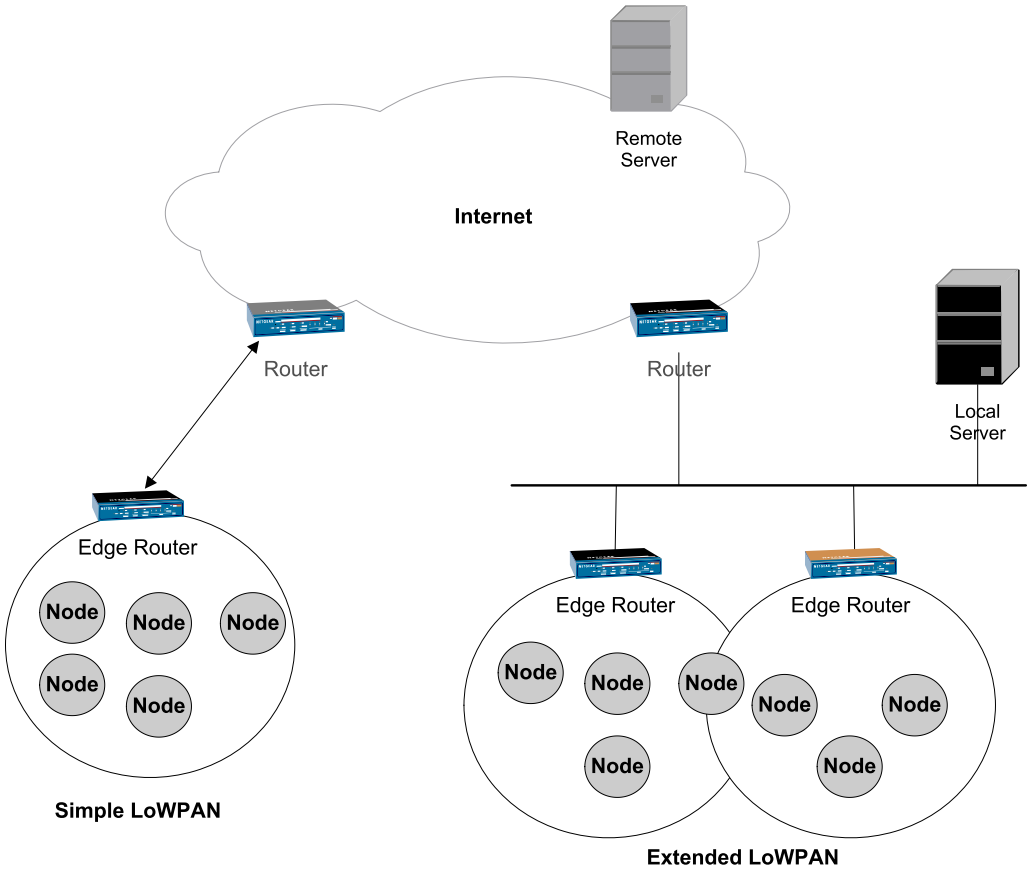


Figure 2.1: Architecture of a 6LoWPAN

Figure 2.2 shows a comparison of 6LoWPAN protocol stack with its IP counter part.

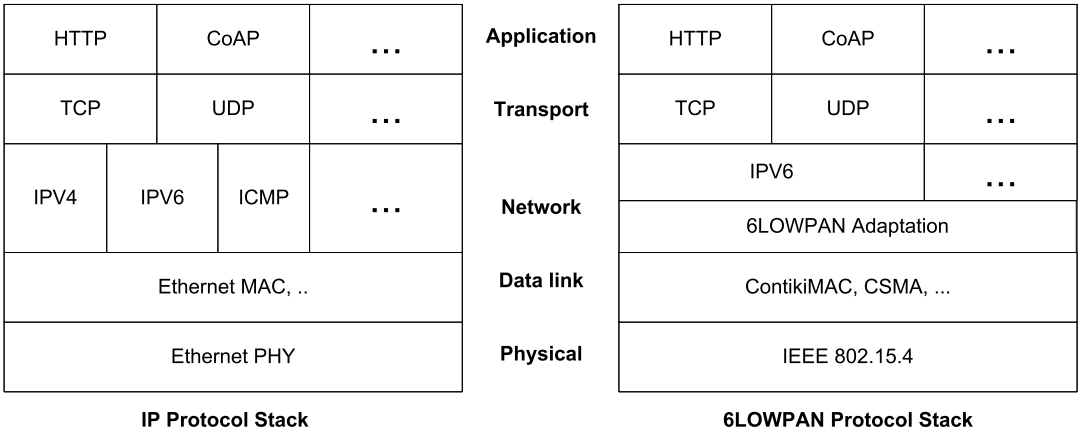


Figure 2.2: IP and 6LoWPAN protocol stacks

2.2.2 Design Considerations

This section describes number of design considerations, which should be taken into account while designing an application protocol for 6LoWPANs [112].

Link layer: 6LoWPANs use the IEEE 802.15.4 low power radio technology for wireless communication. This radio is quite distinctive in nature from IEEE 802.11 WLANs, so the unlike attributes of IEEE 802.15.4 are highly considerable. It uses Carrier Sense Multiple Access (CSMA) as a MAC with multiple retransmission. The rate of packet loss also increases if there is radio interference. There is no built-in multi-cast support, which can be crucial in the context of sensor networks. Furthermore, links are asymmetrical, so the packets successfully sent in one direction do not guarantee delivery from the other side. The most constraint feature of IEEE 802.15.4 is its limited bandwidth as at the physical layer payload size is 127 bytes and offer only 60-80 bytes ideally for a User Datagram Protocol (UDP) payload [114]. The data rates are also typically between 20 and 250 kbit/s and are shared by all nodes on the channel, and will drop further over multi-hops. Therefore, end-to-end reliability is important for applications, because of the lossy links. The application protocol for 6LoWPANs should use compact binary headers and payload formats.

Networking: UDP has the most favourable qualities (for example, connection-less behaviour) to be considered for 6LoWPANs. On the other hand, Transport Control Protocol (TCP) does have some use, but this will require a new enhanced, and compact version of TCP. This option is interesting to be considered as most of the IP-based protocols rely on TCP because of its

reliable connection-oriented byte stream. Large packet transfer can be achieved by using the fragmentation feature of 6LoWPAN, but this will increase the traffic and message response time.

Host Identification Issues: The identification of the devices is a key for end-to-end communication. There are several ways in which a device can be identified such as a serial number, the IPv6 number of node, 64-bit Extended Unique Identifier (EUI-64) number, or by its domain name. The IPv6 address can change for a mobile device when it moves or point of attachment changes. An EUI-64 [52] is a unique serial number of the device; it is reliable identifier but needs to be resolved by the node.

Compression: Traditional IP protocols are not designed for constrained environments; therefore, these do not address the inherent challenges in these environments. For example, features like human readability and protocol extensibility are not of great priority in Machine-to-Machine (M2M) communication. Similarly, constrained devices are not able to implement complex protocols. Consequently, existing IP-based protocols require compression to be applicable for constrained devices and environments.

Security: 6LoWPANs use link layer encryption for securing the links in a network. IEEE 802.15.4 also secures each link with a built-in 128 bit Advanced Encryption Standard (AES) encryption feature. In a multi-hop network, intermediate hops to have the same encryption key, which makes the application data vulnerable on these hops. Applications that deal with important data demand more security, for example, enterprise and defence systems. If an application is dealing with sensitive data, it can apply end-to-end application layer security. This enables only the involved end-points to encrypt and decrypt the data accurately.

2.3 RPL (IPv6 Routing Protocol for Low power and Lossy Networks)

IPv6 Routing Protocol for Low power and Lossy Networks (RPL) [133] is an IETF standard and focused on Low Power and Lossy Network (LLN) a class of network in which devices including routers have constraints on memory, processing power and energy (battery powered). Furthermore, the types of traffic flows in these networks include point-to-point (between devices inside a network), point-to-multipoint (from a central control point to a subset of devices inside a network), and multipoint-to-point (from devices inside a network towards a central control

point). In the start, the designers of RPL identified the routing requirements for LLNs, which were used to evaluate existing protocols including Open Shortest Path First (OSPF), Ad-hoc On-Demand Distance Vector Routing (AODV), Intermediate System to Intermediate System (IS-IS) and Optimized Link State Routing Protocol (OLSR). However, they couldn't find a match for the unique requirements of LLNs such as constrained devices and unreliable lossy links with high loss rates, low data rates, and instability. Subsequently, RPL was designed that is tailored to deal with the requirements of wide range of LLN application domains.

2.3.1 Protocol and Topology Construction

RPL is a proactive distance vector protocol and does not rely on any particular features of a specific link layer mechanism. It requires bi-directional links (may have asymmetric properties) between devices to build one or more Destination Oriented Directed Acyclic Graph (DODAG). A DODAG is a set of vertices without any cycle in them where each node has a path towards a single root. All the routes are optimised for traffic to or from the root that represents the sink for the topology. This makes RPL more suitable for LLNs where topology is not predefined by point-to-point wires. Furthermore, RPL allows a root to define an Objective Function (OF) that is used by rest of the nodes to optimise the paths to achieve objectives e.g., minimizing energy, minimizing latency, or satisfying constraints. A leaf node can have multiple paths towards the root, which satisfies an important routing requirement for LLNs.

Objective Function (OF): Each RPL's DODAG specifies an OF [122] that is used by nodes to optimise the routes. The OF is defined by an Objective Code Point (OCP) in a DODAG Information Object (DIO) message. It defines the rules for nodes to translate one or more metrics and constraints [125] into a value called rank. The metrics could be combination of node attributes e.g., hop count, node residual energy, or link attribute including throughput, latency, link quality level or expected transmission count (ETX). This rank approximates the node's distance from the root in a DODAG.

Upward traffic: RPL specifies DODAG Information Solicitation (DIS), DIO and DODAG Advertisement Object (DAO) messages to build a DODAG, which are defined as new ICMPv6 messages. The DIS message, which is analogous to IPv6 RS (Router Solicitation) message, is used by the nodes to discover DODAGs in their vicinity. The root of a DODAG (usually a border router node) wraps OF and other configurations for the DODAG in a DIO message and advertise it using link-local multicast. Other nodes receives the DODAG

configuration and spread it in their vicinities. When a node joins a DODAG, it computes the rank of its neighbours using OF and decides about its parent node to maintain an upward route towards the root node. These routes enable the multipoint-to-point traffic flow from nodes in a network towards a sink (root node).

Downward traffic: RPL's DAO messages are communicated by nodes to maintain routing information in the downward direction that is used for point-to-multipoint and point-to-point communication. These messages carry related information including IPv6 destination address, prefix or multicast group. RPL has two modes of downward traffic: storing (nodes have routing tables to destinations) or non-storing (fully source routed as nodes do not store any information about routes). In storing mode, the point-to-point traffic will go upwards until a mutual parent node is found to route the traffic downwards towards the destination. On the contrary, in non-storing mode point-to-point traffic needs to go upwards all the way to root before moving downwards, as no node in the path has any routing information to route the traffic.

Controlling RPL's Topology: The DODAG's OF is a key to control the routing topology of RPL. It influences the decision of nodes to select their parents, to maintain routes towards root. Therefore, any implementation or deployment can specify administrative preferences by changing the OF to control traffic and configure a DODAG formation to better support application requirements [61, 133].

Adaptability: RPL uses on-demand loop detection using data packets to deal with the low-power and lossy nature of constrained networks. Thus, RPL's control traffic is adaptive to the stability of a DODAG, because maintaining a routing topology that is constantly up-to-date with the physical topology can waste energy. Therefore, the infrequent changes in connectivity or loop detection in a DODAG are only addressed by RPL when there is data to be sent. RPL employs Trickle timers [72] to adapts to the changes by determining the frequency of DIO and DAO messages. The ranks (computed using OF) of nodes are examined while making any routing decision (upward or downward) to discover any inconsistency, e.g., any loop in a DODAG. When a node receives such a packet, it institues a local repair operation and consequently changes the frequency of DIO and DAO messages.

2.4 Application Protocol Paradigms

According to Shelby and Bormann, Internet application protocols function in four different paradigms [112]. Those paradigms are end-to-end, real-time streaming and sessions, Publish and Subscribe (PubSub) and web services. The explanation of each of those is described in this section.

2.4.1 End-to-End

In this paradigm, only application end-points take part in the application protocol exchanges. Both application end-points use Internet socket model between them, which is based on the transport layer to provide a byte stream service or an IP datagram. There can be exception of those application protocols, which allow the use of intermediate nodes to cache, modify or inspect application protocols. For example; HTTP uses these intermediate nodes to perform web-content caching, which are known as proxies.

In case of 6LoWPANs, most of the devices are battery operated and thus their availability can be intermittent, so end-to-end paradigm's role is significant in the realisation of protocol compression. This can be achieved in two different ways. In the first approach the compressed format can be supported on the both application end-points. The second approach is of placing the functionality in the edge router, which can then act as a proxy. The later approach can be useful as it eliminates the need of modification of the applications on constrained nodes.

2.4.2 Real-time streaming and sessions

The applications which involve sensor video or audio have the requirement to deal with real-time data streams. Generally the Internet protocols deal with the real-time data with best-effort approach, which means without any Quality of Service (QoS). This approach introduces considerable jitter, and packets may arrive out-of-order, which should be considered by real-time applications.

There are different Internet protocols which are used to deal with the real-time traffic including Real-time Transport Protocol (RTP), RTP Control Protocol (RTCP) and Session Initiation Protocol (SIP). The RTP adds the time-stamp and sequence number to the stream and uses RTCP to control it. SIP is used to automatically setup and configure the relationship between sender and receiver.

2.4.3 Publish/Subscribe

This is an asynchronous paradigm in which messaging is done in a way that sender sends data without having any knowledge of the actual receiver, and receiver subscribes to a topic which is based on the content of the data. In this centralised architecture, sender is a publisher that publishes the data with a topic to centralised brokers and receiver subscribes with the topic of interest to get the data from a broker. A central broker decouples the application end-points, which gives a lot of flexibility to the network. In wireless embedded Internet, the concept of PubSub is of great significance in those applications which are data centric, where the source of the data is not important.

2.4.4 Web service

A web service is a software system designed to support interoperable M2M communication over a network. It works between clients and servers and uses HTTP to operate. The concept of web services is the idea of having a simple URL available on the servers with resources or services to be called from them. There are two different kinds of web services; Service-based (SOAP) web services and Resource-based (REST) and web services.

2.4.4.1 Simple Object Access Protocol (SOAP)

The SOAP based web services are a defacto in enterprise M2M systems. It uses one URL to identify a service that can implement several Remote Procedure Call (RPC) calls. These web services use HTTP and RPC for message negotiation and transmission between clients and servers. The SOAP messages are in XML format, and the sequence of messages is explained in Web Service Description Language (WSDL) [27]. Following is an example of a simple SOAP message's header:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
```

```
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

The below example shows an example of a URL with some methods, which can be used in a single SOAP message. A RPC call example to execute different methods:

```
URL:
http://sensor1.lboro.ac.uk/soap

Methods (Analogy is verb):
getSensorValue(sensorNum)
setSensorValue(sensorNum)
```

There are many issues involved in the adaptation of SOAP based web services. Its XML format is verbose and thus poses a serious challenge to use these web services in 6LoWPANs. Furthermore, it employs HTTP as a transport protocol, which demands for more complex processing power at both ends. For example, the above message header has length of 408 bytes, which is too large for a 6LoWPAN frame. In this case, sending this message over a 6LoWPAN, will require up to six fragments to transmit. On top of this, HTTP needs TCP to operate, which requires more reliable connections. These limitations need serious optimisation and change in design to work in 6LoWPANs. SOAP based web services are used in various efforts [39, 91, 92, 116, 117] by employing gateways.

2.4.4.2 Representational state transfer (REST)

Another dimension of realising the web services is using resource based REST [36] design. This architecture is much simpler and straight forward, as a message is not being wrapped in a separate body. Instead, all the resources on the server have corresponding URLs assigned. HTTP is used to request different methods on a resource. In this case, client will only send a HTTP message with the corresponding URL and the method. For example, if a server has a light sensor with `/light` URL, a HTTP GET can be sent with this URL to get the content of the resource. The response can be in any of the Multipurpose Internet Mail Extensions (MIME) type, which is described in the response so that the other end can parse the message.

Following are the principles of REST architecture [36]:

- **URI (Universal Resource Identification):** The REST paradigm uses model objects as HTTP resources, with a unique URL for each resource. Con-

sequently, RESTful web services expose all the resources with corresponding URIs. The users target the desired resource by specifying its URI.

- **Uniform interface:** HTTP is used as an application protocol by the RESTful web services. It allows different methods of GET, POST, PUT and DELETE to be use to get, set or delete the value of a resource. The Web Application Description Language (WADL) is used to describe the interfaces of the resources. The content of the resource is defined by the MIME, normally XML is common in M2M applications.

- **Self-descriptive messages:** The representations of the resources can be described in any common format, e.g., HTML, XML, UTF8, PDF and GIF. The decoupled nature of the resource representations allows the freedom of using any commonly understandable format between endpoints.

- **Stateless operations:** Every interaction with the resource is stateless, thus this eliminates the need to carry all related information.

Some examples of REST messages are given below which represent the analogy of nouns.

```
http://s1.lboro.ac.uk/sensors/light    ->Method:Get
http://s1.lboro.ac.uk/sensors/temp     ->Method:Get
http://s1.lboro.ac.uk/sensors/acc ->Method:Post, Payload:Value
http://s1.lboro.ac.uk/config/sleeptime ->Method:Put, Payload:Value
http://s1.lboro.ac.uk/config/waketime  ->Method:Put, Payload:Value
```

RESTful web service paradigm is employed in different research efforts [119, 132] because of its simplicity and suitable characteristics for embedded constrained networks. The research effort [43] which coined the idea of WoT has also utilised the RESTful web service paradigm.

2.5 CoAP (Constrained Application Protocol)

The IETF Constrained RESTful Environments (CoRE) working group is working on realisation of the REST architecture in an optimum and more suitable form for the most constrained nodes and networks (such as 6LoWPANs). The CoRE working group is designing CoAP [13], which is an alternative (to HTTP) compact way of enabling RESTful web services in constrained environments. The applicability of CoAP is especially considered for commercial applications which include building automation and other M2M applications. Following are the features of the CoAP,

which make it more suitable and interoperable solution for the wide range of constrained networks and especially for industry-specific networks:

- **Constrained web protocol fulfilling M2M requirements.** CoAP is a simple and compact protocol for constrained node, as it is deployable on nodes with 8-16 bit micro-controllers, 64-256K of flash and 8-12K of RAM.
- **Low header overhead and parsing complexity.** CoAP protocol is optimised for the extremely restricted throughput (order of tens of kbits/s), limited bandwidth (60-80 bytes for application layer payload), and a high ratio of packet loss.
- **Simple proxy and caching capabilities.** Caching is supported by the protocol. If a proxy is employed, it can cache recent responses to later reply on behalf of a sleeping node.
- **Different types of message exchanges.** CoAP follows the REST architecture, so it allows creating, reading, updating and deleting a resource on a device. The normal protocol transaction consists of a single request and response exchange.
- **UDP binding with optional reliability supporting unicast and multicast requests.** Constrained networks have high rate of packet loss, so CoAP supports UDP as transport protocol with back-off mechanism for reliability. It describes an option for sending larger chunks of data using UDP. Moreover, it supports multicast with no reliability.
- **URI and Content-type support.**The Internet has a large list of media types; CoAP supports the subset of these media types.
- **Stateless HTTP mapping** The basic goal of CoAP is to integrate seamlessly constrained networks with the Internet. Therefore, CoAP defines stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way, or HTTP simple interfaces to be realised alternatively over the CoAP.

This section covers a brief introduction to the CoAP protocol.

2.5.1 Transaction ID and messages types

Each CoAP end-point has a transaction ID (unsigned integer) which is initially randomised and then changed each time for a new confirmable or Non-Confirmable message. Transaction ID of a response is matched for every corresponding request.

The Token Option is used to match a response with a request. Furthermore, every request has a client-generated token which is echoed back by the server in every response. This option is used to deal with the delayed response, as a transaction ID will remain same for one response and its corresponding request, but varies if a transaction involves more than one requests or responses.

There are four different message types used by the CoAP transactions as defined by the protocol. These messages are transparent to the request/response carried over them.

1. **Confirmable (CON):** This message type is used when an acknowledgement is required in response. The ACK or RST message types are used in response to CON.

2. **Non-Confirmable (NON):** This type of message is suited for a scenario where no acknowledgement is required e.g., if a sensor is sending readings frequently, there is no need of acknowledgement.

3. **Acknowledgement (ACK):** The ACK is used to inform the sender of the message that message has been successfully received. The message with ACK type can also carry a payload to save the communication overhead.

4. **Reset (RST):** A Reset message explains that the receiver has lost the context information of some Confirmable message, and is unable to process the response.

2.5.2 Methods

One of the prime requirements for CoAP is to easily map to HTTP, that's why it supports basic methods including GET, POST, PUT, DELETE which are akin to their HTTP counterparts. All of these methods can manipulate resources and have the similar safe (only retrieval) and idempotent (same effect even after being executed multiple times) properties. The response has mentioned any unsupported method code, the response should have the response code of "method not allowed" (CoAP 4.05 equivalent to HTTP 405).

1. **GET:** This method is used to get the information of a resource, which is identified by a URI. In case of success a 2.05 (Content) or 2.03 (Valid) response SHOULD be sent. It is safe, idempotent and cacheable method.

2. **POST:** The POST method is used to create a new sub-ordinate resource under the given resource URI. On the success of the POST, response code of 2.01 (Created) in case of resource creation and 2.04 (Changed) when existing

resource is updated, is sent back. When a new resource is created, the response includes the URI of the new resource in a sequence of one or more Location-Path Options and/or a Location-Query Option used to send the URI of the created resource. The POST is not safe and neither idempotent.

3. **PUT:** The PUT message is used to create or update a resource specified by a given URI. If the resource exists, it is updated by the modified version which is appended in the message body and 2.04 (Changed) response code is returned. If no resource exists, the receiver may create a new resource with the given URI, and 2.01 (Created) response code is sent back. In case of any problem, the respective error code is sent in the response. The PUT is not safe, but is idempotent.

4. **DELETE:** This method is used to delete a resource which is specified by the sent URI. The 2.02 (Deleted) code on success or an error message is sent back in response. The DELETE is not safe, but is idempotent.

2.5.3 Options

CoAP defines different options to be used with requests and responses. Most common option `Content-Format` is used to describe the format of the information in the payload. CoAP has specified very compact and extensible Type-Length-Value (TLV) style option format. This section covers the basic handling of the options and the concepts of URI and content types, which are related to options.

- **Options processing:** CoAP has elective and critical types of options. There is a place of option count in the header, which is set to 0, if there is no option. If any option is specified in the message, it is placed in the message after the header. In case of unknown options, the elective types of options can be skipped, and an error response code 4.02 (Bad Option) with the critical option number in the payload is sent back in the response.

- **Universal Resource Identifier (URI):** CoAP has four options to deal with the Universal Resource Identifier (URI), which is an important feature of the REST architecture (on which the web is based). The CoAP URI consists of scheme, authority, path and query options. The protocol reconstructs the URI using those options. The example of the CoAP URI is

  ```
  coap://[IP address]:port/s/light?status
  ```

  CoAP supports this URI by using four different options in the following way

- **coap** is given in URI scheme option
- [**IP address**] is described in the URI authority option
- **/s/light** is mentioned in URI path option
- **?status** is specified in the URI query option

CoAP does not support "." Or ".." in URIs, IRIs, and does not fragment "#" processing. Every CoAP request contains a URI-path option and can have a URI-QUERY option. The URI-scheme and URI-authority are optional between CoAP end-points. If URI-authority is needed to reconstruct the URI but not given in the message, an error code 4.02 (Bad Option) is sent in the response.

- **Content types:** The CoAP specifies a subset of MIME types as its content types. As part of optimisation, designers have changed the string definition to 1-byte code definition, which describes the content type of the payload. The default value of this option is "text/plain" which is assumed, if content type option is not given in the message.

2.5.4 Message Format

CoAP uses a very simple and compact binary header, which is followed by options in the Type-Length-Value (TLV) format. Any bytes after header and options are considered as payload that can be calculated by using the datagram length. The format of a CoAP header is shown in the figure 2.3. The header fields include the Ver as CoAP version, T as transaction type, OC as the number of options in the message, Code as a method or response code. All options come after the header, then a payload is appended at the end.

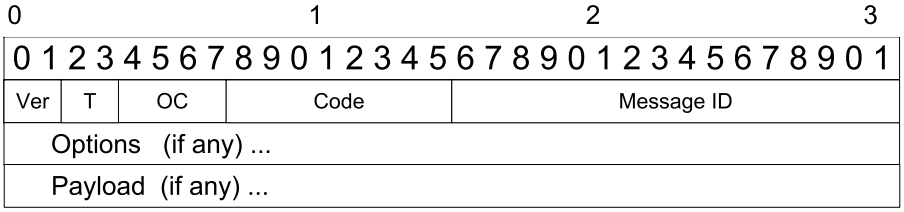


Figure 2.3: CoAP Packet format

The fields in the header are defined in as follows:

- **Version (Ver):** *2-bit unsigned integer.* Indicates the CoAP version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.

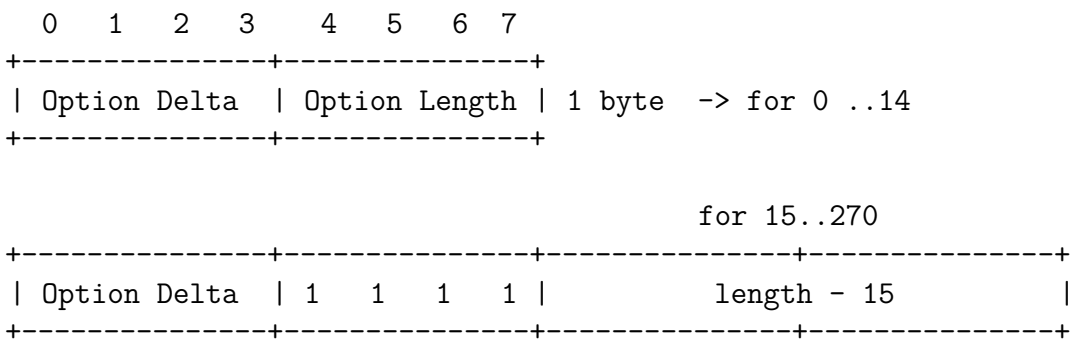


Figure 2.4: CoAP Option header format

- **Type (T):** *2-bit unsigned integer.* Indicates if this message is of type Confirmable (0), Non-Confirmable (1), Acknowledgement (2) or Reset (3).
- **Option Count (OC):** *4-bit unsigned integer.* Indicates the number of options after the header. If set to 0, there are no options and the payload (if any) immediately follows the header. The format of options is defined below.
- **Code:** *8-bit unsigned integer.* It is further split into 3-bit class (most significant bits) and a 5-bit detail (least significant bits), comparable to HTTP codes as `c.dd` where `c` is a digit from 0 to 7 for the 3-bit sub-field and `dd` is two digits from 00 to 31 for the 5-bit sub-field. Indicates if the message carries a request (0.01-0.31 as 1-31) or a response (64-191 as 2.00-5.31), or is empty (0 as 0.00). All other code values are reserved. In case of a request, the Code field indicates the Request Method; in case of a response a Response Code.
- **Message ID:** *16-bit unsigned integer.* Used for the detection of message duplication, and to match messages of type Acknowledgement/Reset and messages of type Confirmable

The number of options is defined in the header, while options are appended after basic header, in a sorted manner. The option with the least number will come first, and rest of options follow the same rule. The critical (must be understood or error generated) options are recognised by the odd value and electives are given even values. The option header consists of option delta and length. The option delta is used to identify the type of the option. Each option header has the format as shown in figure 2.4. The option header requires 1 byte or 2 byte (if option length is more than 14 bytes).

2.5.5 UDP binding

UDP is the default transmission protocol for CoAP. The other transmission protocols like TCP or SCTP can also be used with CoAP. The requirements for the UDP were to provide the bare minimum features with some reliability, to avoid the need of creating a compact TCP with a new feature set. CoAP defines following reliability support for UDP:

1. It gives some reliability by providing simple stop and wait retransmission reliability with exponential back-off for Confirmable messages.
2. The transaction IDs are used for the response matching
3. It also supports multicast communication.

2.5.6 Interaction Model

The client/server model is followed as an interaction model by CoAP. It is similar to HTTP, as a client sends a request for an action on a resource (which is identified as a URI), and then server sends back a response consisting of response code and resource representation. However, a CoAP implementation plays the roles of both server and client, and is called an end-point.

To fulfill the M2M requirements, CoAP supports asynchronous interchanges over a UDP. This is achieved using four transaction messages (Confirmable, Non-Confirmable, Acknowledgement, Reset). The asynchronous transactions are completed by following the same request and response interchanges, with the permission of deferred response.

This allows to think of CoAP with two layers, i.e., a request/response interaction using methods and response codes, and a transactional layer used to deal with UDP and asynchronous transactions. This conceptual view of CoAP transactional model is shown in figure 2.5.

2.5.6.1 Synchronous response

The synchronous response of CoAP is similar in functionality to HTTP. In CoAP, a client sends a confirmable request message and then waits for the acknowledgement message which also carries the response. The scenario is shown in the figure 2.6.

The optimisation here is the relationship between the confirmable message (CON) and an acknowledgement message (ACK). The ACK message also carries the response in payload, which minimises the protocol communication overhead. The transaction IDs are used to match the specific acknowledgement message (ACK) to the confirmable message (CON).

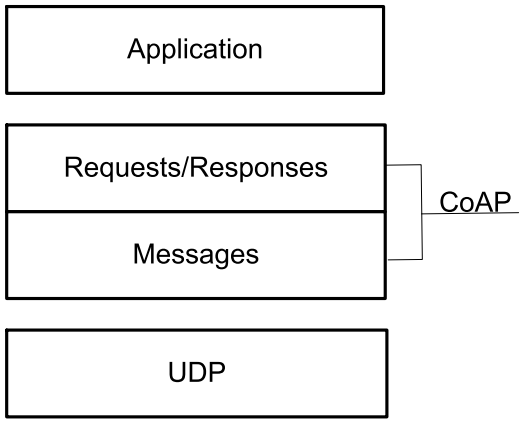


Figure 2.5: Abstract layering of CoAP

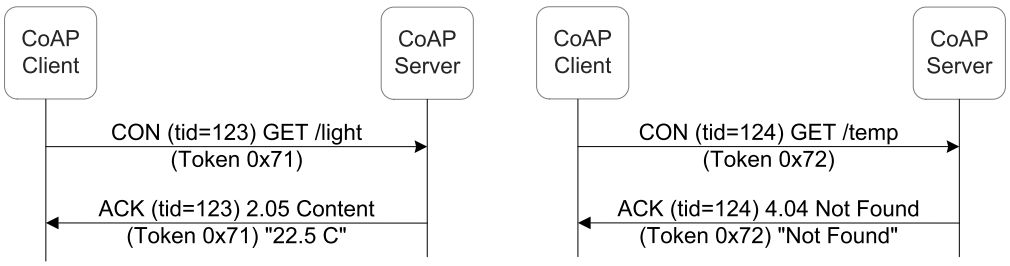


Figure 2.6: CoAP Synchronous Scenario

2.5.6.2 Asynchronous response

The asynchronous response of CoAP is crucial in a situation where a server might need some time to send a response. In this case, the client uses the token message of the request. The server knows that it can't send the response immediately so it keeps the record of the token and simply informs the client about the delay of the response with an ACK (without any resource representation appended). This alleviates the risk of client to retransmit the request repeatedly. When the server knows that the resource representation is ready, it will make a confirmable message to send the resource representation with the same token but with a new transaction id. The client uses the ACK with the received transaction id, to inform the server that it has received the response. The scenario is shown in the Figure 2.7.

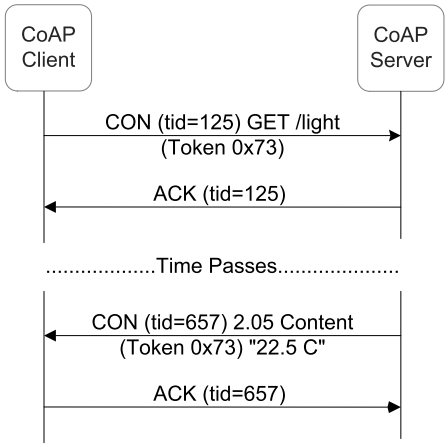


Figure 2.7: CoAP Asynchronous Scenario

In the special failure situation, when the client is not at all able to process a Confirmable message (i.e., not even able to provide a suitable error response), it sends the special Reset message as shown in Figure 2.8.

2.5.7 Resource discovery

The CoAP end-point supports the CoRE Link Format [111] of ascertainable resources. This resource discovery feature is crucial in the M2M application, where no human is involved. This standard defines Web Linking using a link format to utilise by constrained web servers to describe hosted resources, their attributes and other relationships between links. The attribute value pairs of this format can easily represent the desired information. The idea behind using this format comes from providing interfaces to collections of resources offered by RESTful design paradigm.

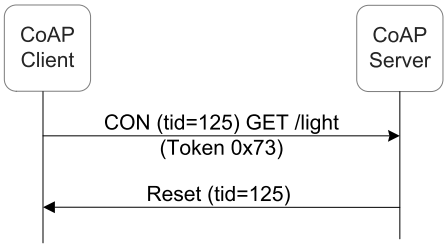


Figure 2.8: CoAP Reset Scenario

The linking between those resources works like pages on a website. CoAP defines that each device can offer a web resource with URL "/.well-known/core", to offer resource discovery.

2.5.8 Caching and Proxying

CoAP end-points are constrained in terms of bandwidth and processing power. To optimise the performance and life-time of the network, CoAP specifies the caching feature like HTTP. The proxies provide the features of caching, and also send responses on the behalf of the sleeping nodes.

2.6 Protocol Integration Approaches

Traditional IP-based protocols are not directly applicable to resource-constrained environments. This section describes two different approaches to integrate complex protocols into these environments [112].

2.6.1 Gateway approach

This is a traditional approach widely used in non-IP wireless embedded networks such as ZigBee [5] and other vendor-specific solutions. In this approach, a gateway is introduced at the edge of the network that deals with the protocol translation. In 6LoWPANs, this can be achieved by implementing the gateway into the edge router. Therefore, the gateway becomes an end-point for the protocol, which controls the devices and sends responses on their behalf. This allows a proprietary protocol to work within a 6LoWPAN, which makes the gateway dependent on the content of the protocol. Consequently, whenever a new use of the network is added or the format of the application is modified, all the gateways are needed to be upgraded. Resultantly, this causes evolvability and scalability problems for the

network and translation increases the delay as each message needs to be translated twice.

2.6.2 Compression approach

This approach leads to the compression of existing protocols, so that those can be made suitable for use over 6LoWPANs. The integration can follow either end-to-end or proxy approaches. In end-to-end approach, both application end-points should support the compressed format. Whereas an intermediate proxy can seamlessly translate the messages to, and from constrained end-points in proxy based approach.

Both HTTP and TCP are infeasible to be used directly in a 6LoWPAN. In literature, various efforts have used HTTP and TCP, by optimizing existing standards. Some efforts [29, 30] have used an optimised TCP's version. Another effort [59] has used HTTP's simple form to provide RESTful web services. The NanoWS [112] (Nano Web Services) has also tried to transmit binary XML transfer using UDP over 6LoWPANs. Furthermore, the SENSEI project [102] is researching to find more efficient ways of using web services inside wireless embedded sensor networks. Another example is CoAP protocol (Section 2.5) that offers a compact version of the RESTful web services, while offering interoperability to HTTP by seamless translation.

Several standardisation efforts are going underway for the compression of XML. The W3C is working on a standard which will perform compact binary encoding of XML and is called Efficient XML Interchange (EXI) [109] format. A standard WAP Binary XML (WBXML) [83] format was developed for mobile phone browsers. Another draft proposal called Binary XML (BXML) [41] from the Open Geospatial Consortium (OGC) aims to compress the large set of geo-spatial data. EXI is more suitable technology for 6LoWPANs, as it gives compact representation without-of-band schema knowledge.

2.7 Technologies for Experiments

The selection of the technology for experiments is the key to create useful experiments and analysis. This section discusses the features of the selected technologies to emphasise the reasons for their consideration.

2.7.1 Operating System: CONTIKI

This research project employs the CONTIKI [31] operating system for the devices. CONTIKI is an open source, highly portable, multi-tasking operating system

for memory-efficient networked embedded systems and wireless sensor networks. It is designed for the micro-controllers with limited memory. There are many alternatives [35] including the widely known TinyOS. The main reasons to choose CONTIKI was its high portability, better feature set and an IP stack with RPL [133]. It works with both IPv4 and IPv6. Following is the list of CONTIKI's features:

- Advance IP networking with uIPv6/6LoWPAN protocol stack that supports RPL.
- Hybrid threading model with protothreads.
- Power profiling mechanism to keep track of energy consumption of each node.
- Text-based shell interface for sensor network interaction and sensing.
- Dynamic loading of the modules.
- Extensive simulation support with COOJA and MSPsim.

2.7.2 Simulator: COOJA

A simulator plays a pivotal role in the code development and testing of any technique in constrained networks. The testing and verification of certain algorithm is difficult and tedious without a simulator. Simulators can perform the simulation at application, operating system or hardware levels. The time and overhead increases, when a node simulates at a hardware level because of the implementation of device drivers. On the other hand, the more high-level simulation does not model the node's hardware; consequently, it restricts the development to the high level algorithms only.

COOJA [99] is a novel simulator to be used with CONTIKI OS that enables the cross-level simulation (simultaneous simulation at many levels of the system). It is developed in Java but allows to run node software written in C, by using JNI (Java Native Interface). The list of COOJA features are:

- It allows simultaneous simulation at different levels (application, OS, machine code levels) that makes it distinctive among other simulators [60]. This way it combines low-level simulation of sensor node hardware and simulation of high-level behaviour in a single simulation [120].
- It works with the Contiki OS (compatible with the choice of OS made in previous section).
- It can simulate heterogeneous types of nodes in the same network.

- It is easily extensible as new plug-ins and radio mediums can comfortably be changed or replaced.
- It gives a lot of details of the node's hardware.

The rich features of COOJA; especially its support for hardware-level simulations and having heterogeneous nodes in the same network, native CONTIKI support and its extensibility were the key reasons to employ it as a 6LoWPAN simulator for this research.

2.8 Summary

This chapter has covered the detail of technologies, which enable the WoT. The discussion started with 6LoWPAN architecture and issues related to its integration with the Internet. Furthermore, RPL is covered with some detail of its topology creation and existing application protocol paradigms are discussed in the context of 6LoWPANs. The emphasis is given to web service paradigm by explaining both REST and SOAP based web services. This chapter has argued that existing web services are a poor match for 6LoWPANs, because of there large messages and heavy dependence on traditional IP-based protocols. However, compact CoAP based RESTful web services can be leveraged by these resource-constrained networks. Subsequently, some approaches are explained to integrate existing protocols in the 6LoWPANs. In the end, the chapter described the chosen experimental tools.

The next chapter focuses on SD details with the discussion of existing solutions in literature and analyse them from the perspective of 6LoWPANs.

Chapter 3

Service Discovery (SD) in Literature

3.1 Introduction

The IoT environment is begin to form with the advent of new technologies such as 6LoWPAN. Heterogeneous devices ranging from tiny sensors to powerful devices can now become an active part of the Internet. These devices provide a variety of information and services. The management and dynamic discovery of these services are the tasks of a SDP. Moreover, the demand of efficient and reliable discovery of network wide services complicate the problem. For example, a user needs to know the service's location and communication protocol before he can access it, any time and anywhere. This chapter covers the details of different aspects related to SD, including its objectives, existing solutions in literature, design challenges and new perspectives. In addition, some comparative analysis of existing solutions is presented from the perspective of 6LoWPANs.

3.2 SD Objectives

Services encapsulate the functionalities offered by devices. These provide the abstraction from underlying heterogeneous hardware or implementation details. In addition, services can be orchestrated to create new higher-level functionality, and they can be deployed and executed in remote locations, in-situ on an embedded device if necessary [126]. The function of a SDP is to find the software entities or agents that can provide access to required services. An efficient SDP ensures availability of optimal services to users and application [3]. The objectives of SD are:

- **Dynamic Discovery:** The target of SD is to dynamically find a service

provider according to the request properties. To fulfil this goal, a SDP needs a language to describe services, a way to store the service information somewhere, and a protocol to search for services.

- **Self-configuration:** SD has to function without human intervention, i.e. with no administration. This demands the capability of dynamic adaptation of changes in topology and service descriptions, while enabling a user with reliable and latest information regarding available services.

3.3 SD Entities

The process of SD is mapping the service description to service location. The architecture of SDPs identifies the building blocks and the links between the participating components (entities) [79]. At least following two entities participate in the SD process.

Client (User or User Agent (UA)): The entity, which is interested in searching and using a service usually hosts certain applications, which access specific services. The service request is initiated by this entity.

Server (Service Provider or Service Agent (SA)) : The entity, which hosts and offers services.

Protocols may use service repositories, to facilitate the service mapping procedure. Following participating entity is common within SDPs.

Directory (Server, Service Broker or Directory Agent (DA)): The directory node improves the performance of a SDP by acting as a broker or registry for the discovery process.

3.4 SD Classifications

There is a plethora of SDPs proposed in the literature [2, 79, 87, 88, 129]. Existing protocols are primarily focused on some specific environments (Ad-Hoc, Mobile Ad-hoc Network (MANET), etc) mostly other than constrained networks. However, the description highlights the interesting features and approaches used for providing SD. This section classifies the protocols according to their architecture, and summarises the design details.

3.5 Centralised architectures

A Centralised approach introduces only one centralised directory, which keeps the service information of the network. Clients generally use to discover the directory using broadcast or multicast. However, unicast is used by the clients to advertise their services. In a SD scenario, client first contacts a central directory to obtain the matching service, and then contacts the specific node to invoke it.

This approach is more optimal in the service search context, but it creates a single point of failure. Centralised architectures depend upon the availability of a central directory that has a tendency of becoming a bottleneck, as all status maintenance traffic flows towards a single point. The scope of SD is also limited to the devices in the local domain. However, the boundaries of the domain can be defined administratively to IP subnet or to range of a wireless network. This section covers the details of different SDPs based on a central directory.

3.5.1 SLP (Service Location Protocol)

SLP [47] is an IETF standard, and provides a scalable and flexible framework for SD through an IP network.

Entities and protocol: There are three major software entities in the SLP framework: UA, SA, and DA. The SA advertises the attributes of hosted services. The service registration with the DA has a lifetime and service needs to re register after time-out. The SA also replies to the request for services using unicast. The UA initiates a service request on behalf of client application and receives a service URL from the DA or SA. SLP can work with or without a DA. In the absence of a DA, the UA's service requests is spread in the whole network, and any SA offering the service would send a reply back. SLP without a DA works well for small networks. In case with a DA, it forms a centralised architecture by placing a DA as tier between UAs and the SAs, which communicate indirectly through DA. The main function of a DA remains to improve the performance of SLP. The significance of having a DA becomes more apparent in large-scale networks, where the requirement of multicast traffic between UAs and SAs is omitted. The DA works as a registry which stores all SA advertisements.

Service description and queries: SLP provides service templates [46], which is an attribute set of searchable service types. The templates include a specification of attribute: types, default and allowed values. These templates are used to keep the information of services differentiating between them and

communicate configuration information to UAs. SLP also offers filters for attributes using operators like AND, OR and substrings.

SLP allows multiple DAs; therefore, is scalable. It is flexible, because services can be deployed in small networks without any special configuration. It can work without Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP), or routing. Nevertheless, in order to get registered, the address of the central directory needs to be known by every SA. It is not dependent to any programming language. Its centralised architecture is vulnerable to a single point of failure. There is no service selection and service invocation mechanisms are specified by SLP. Polling is used by SLP to fetch the information from the network and eventing is defined by SLP. However, some enhancements [62] are introduced for SLP to adopt subscribe/notify.

3.5.2 SLP-based adaptations and optimised solutions

SLP is not directly applicable to 6LoWPANs as it needs optimisations with more consideration to the message size. As in 6LoWPANs, SLP will have 60-80 bytes at the application layer. Following is the discussion of some research efforts which have used SLP ideas to offer SD solution for constrained networks.

SSLP with proximity: The Simple Service for Location Protocol (SSLP) [63] which supports many features of SLP is a draft proposal in the IETF 6LoWPAN work group. SSLP provides a simple and lightweight framework for the discovery of network services in 6LoWPANs. It uses Tokenised XML strings to minimise the packet exchange. SSLP can also inter-work with SLP in external IP networks. This can be achieved by using simple translation of SLP to SSLP at edge-routers, which enables the clients to discover and control services inside a 6LoWPAN even from outside the network. A proximity-based discovery solution [18] follows the same framework and introduced a TA (Translation Agent) to translate the SSLP to SLPv2 compatible type and vice versa. This protocol employs multiple local registries and uses multicast for the advertisements and status updates. However, its dependency on gateway marginalise the benefit of using IP for end-to-end communication, and the translation between protocols is complex and incurs a large overhead.

nanoSLP: nanoSLP [59] is a miniaturised form of SLP that does not support DA and multicast. The communication between UAs and SAs is allowed using uni-cast or multicast. In nanoSLP, the resource values are piggybacked in the response to a SD message to save one round trip time. The format

of the request message is also enhanced by using URL instead of a service name. Consequently, one request can be addressed to several nodes. This is a useful feature for a WSN, where nodes are addressed by a geographical location (geocasting). The complex search is achieved using one byte encoded comparison field, called Flags. The drawback is that as the network grows, the burden of multicast increases considerably. Furthermore, nodes have sleep cycles as well, so the discovery process may return without considering the sleeping nodes.

nanoSD: nanoSD [69] borrows some ideas of nanoSLP. It has introduced a mapping tree structure to enable compression of service identifiers into optimal binary strings. Resultantly, it reduces the packet overhead and also decreases the processing and lookup time. Moreover, this structure generates WSDL style service descriptions for the backward compatibility with the web-based back-end systems. The WSDL style service descriptions enable sensor nodes to communicate with the variety of other devices using web services through a gateway.

3.5.3 JINI (Java Intelligent Network Interface)

JINI [9] is a SDP, which relies on the Java language and can be considered as an extension of the Java environment from a single machine to the whole network. It consists of service providers, lookup servers and clients.

Service Registration: A service provider uses multicast to find the lookup server to store services, and respond to clients who need a service.

Service Description and Discovery: Services are stored in the lookup servers in the form of service objects, which have different attributes. The object has Java interface for service invocation with other descriptive attributes. The lookup process copies the object to the client, which is then available for other clients by object interface. A service is provided to a user on lease basis, which requires renewal to prolong the validity. The lease time can be varied in the case of a busy server.

JINI has a rich service description format and supports subscribe/notify mechanism, where a subscriber of an object gets notification of the changes. The major drawback of JINI is that it requires each device to either run a Java virtual machine or to associate itself with a device (proxy) that can execute a JVM on its behalf. It also needs reliable, stream-oriented communication and a multicast facility. Furthermore, JINI makes the assumption about the capabilities of RAM and CPU

power of the devices connected to the network. It also requires the Java Virtual Machine (JVM) and Java Remote Method Invocation (RMI). A service proxy poses an extra challenge, that all the devices should have the common interface, which is not feasible, as no implementation standard is defined for JINI.

3.5.4 Salutation

Salutation [105] is an architecture for SD, proposed by Salutation Consortium that is non-proprietary and primarily solves the problem of SD and can be utilised by heterogeneous devices and network technologies. The development of a transport-independent SD solution was the main aim of the designers.

Entities and Protocol: Its architecture consists of three components: Functional units, Salutation manager (SM), and transport manager (TM). A functional unit defines services using a descriptive attribute unit. A service provider locates a nearby SM and registers itself with it. The SM is a registry of service providers and their services, similar to JINI lookup service. During the discovery client contacts local SM to locate a service. The search is then performed with the coordination among the SMs. The RPC is used for the communication. The TM is responsible for providing reliable channel independent of underlying transport, but it can support one network transport at a time. Therefore, if devices are using many different transport protocols, each is handled by a separate TM. The SM not only acts as a service registry, it also discovers services and manages the sessions.

Salutation is independent of employed transport protocol. Salutation-lite is the lighter version of Salutation for constraint devices and networks. It primarily focuses on SD and waives optional functions. The Salutation consortium was dissolved in 2005 and its specification is no more maintained on the website. Consequently, it can't cope with the new challenges posed by drastically different technologies including 6LoWPAN.

3.5.5 FRODO (Framework for Robust and Resource-aware Discovery)

FRODO [121] SDP targets the home environment with a limited number of devices. It focuses on two main issues of resource-awareness and robustness. It defines its own communication protocol which makes it independent of the underlying network.

Resource Awareness: FRODO address resource-awareness by specifying three different types of devices which include 300D (powerful devices), 3D (medium complex devices), and 3C (simple devices). The architecture of FRODO consists of users (who use services), managers (who provide services) and registries (who store service information). It elects one of the most powerful 300D node as a registry. Another node is selected as a backup registry, which takes the responsibility of a central directory in the case of failure. The availability of a backup registry provides the robustness by making FRODO resilient to single point of failure. Both solicited/unsolicited registration mechanisms are used in FRODO; it depends on the device's capabilities. The state is maintained by polling the registry in case of a resource constrained devices like 3D and 3C. On the other hand, the powerful devices like 300D periodically poll the registry by themselves.

3.5.6 SLEEPER

Sleeper [15] is a SDP, which uses proxy advertisement and discovery to reduce the workload from power constrained nodes. It has a special emphasis on limiting the power consumption of devices. The advertisement structure supports several modes of SD, including conventional service advertisements, meta discovery, taxonomic-based discovery, location-based discovery and federated discovery. Both pull, and push modes of advertisement are used by the protocol. Service popularity attribute i.e. number of times a node is called for a service in a recent time window, is used as a decision criterion to select the respective mode of advertisement.

Energy efficiency: Sleeper uses an algorithm to select a proxy node with respect to capability criteria. The other service nodes can be in a state of online, offline and standby. In the online state, a service node advertises its services, service popularity and capabilities to the proxy. The node can go to stand by mode until it receives the signal for service provisioning. The proxy periodically broadcasts the service information and popular advertisement. Sleeper is analytically compared to Universal Plug and Play (uPnP), Simple Service Discovery Protocol (SSDP), and DEAPspace and the analysis promises comparable response time with the benefit of power saving [15]. However, the criteria for selection of proxy nodes and design issues of proxy nodes in a geographical network are not addressed by the protocol.

3.5.7 Splendor

Splendor is a SD model which emphasises security and supports privacy. Location awareness is used to decrease the discovery network infrastructure [138].

Protocol: The initial communication between clients and a directory is done using multicast. The protocol assumes that all parties know the multicast address in priori. All service information is registered on predefined directory nodes. A client can directly query to a directory node to discover services with unicast requests. The directories cache service information and respond to queries.

Security: Services and clients use certificates to verify and authenticate the directories. Splendor supports mobility by keeping the soft states of services and with periodical directory advertisements. It also stores services represented by proxies as a hard state in directories. The directories explicitly query proxies about status of services. The use of proxies also enables Splendor to offer security and privacy features.

3.6 Distributed architectures

In this architecture, many directories are maintained in a network which hold the service information of the nodes in their vicinity. DNS based protocols ensure scalability by using a hierarchical architecture. Cluster-based protocols do not use the preselected directories. Instead, directories are elected based on suitable capability, which can be battery power, node coverage, memory, or processing power. Hash-based protocols built peer-to-peer overlay networks and offer distributed hash tables (DHT). The significance of using distributed directory-based architecture increases, as size of a network becomes larger.

This section covers the detail of various distributed directory based approaches.

3.6.1 Domain Name Server (DNS) based

DNS is employed to translate human readable names to IP addresses. However, it assumes that sets of Internet host names (domain names) and addresses do not change frequently. Consequently, registration just follows a static process type. New pairs of domain names and addresses are stored at a designated server. The lookup process starts when a client sends a query to local host. At first, the local host checks its DNS entries. If it can't provide the mapping, it passes the query to DNS server one level up in hierarchy. The location information of the next DNS

server is configured for a host. Following is the detail of some of the protocols which follow DNS architecture for SD.

GloServ: GlovServ [8] is a global SD architecture to locate local and wide-area services. The architecture of GloServ is similar to DNS, which contains root name servers and authoritative name servers to manage the information of services. The DNS like structure is used to provide scalability. Services are defined in Resource Description Framework (RDF) and queried using the RDF Query Language (RQL). The RDF also enables a service to include contextual information. The service classification on high level is based on general categories like events, services, people and places.

A SA can register its service to GloServ by specifying the particular point in the hierarchy. URNs (Uniform Resource Names) are used to reference services in GloServ, because services can support different protocols such as HTTP, SIP and others. RDF specifies the mapping of a service to URN and details of its properties. A UA which already holds a copy of the updated service hierarchy, queries directly for a particular service on behalf of a user. The formats used by this protocol are verbose and require a number of packets to communicate a single message in constrained networks such as 6LoWPANs.

Electronic Number Mapping (ENUM) based SD: This is an attempt to adopt the DNS based architecture in 6LoWPANs using ENUM-based numbers to provide SD [7]. ENUM assigns E.164 numbers [14] (unique identifier) to services to make them globally accessible. The architecture consists of sensor nodes and few master nodes. Each sensor node is associated with a master node. The gateway is used to communicate with the outside network. Sensor nodes are computationally constrained devices, which depend on master nodes for communication with the gateway. Therefore, master nodes not only forwards the sensor nodes' requests to gateway, but also passes services destined for them. The protocol translates E.164 numbers to domain name and sends it to DNS server to get matching Name Authority Pointer (NAPTR) [86] records. These records are resolved into URIs which are returned to the clients.

The protocol assumes that DNS lookup will always be able to find matching NAPTR records, and sensors (multi-hop distance away from master node) must know the route towards the master node. However, the use of E.164 numbers and NAPTR format reduces the message size and results in low latency and bandwidth utilisation.

3.6.2 Clustering based

In this approach, a set of nodes is elected as act like distributed directories. These nodes act like hubs for the neighbour nodes, as all SD messages travel among them. This approach restricts the use of multicast and consequently, reduces the network traffic significantly. To address the issues of both latency and energy conservation, Clustering in the wireless sensor network has proven to be the most efficient [82].

Clustering for Service Discovery (C4SD): Clustering for service discovery (C4SD) protocol [81] selects the cluster nodes and cluster heads based on their capabilities. The cluster heads act as directories, and other nodes register services with the cluster heads in their vicinities. A service requester just needs to query the cluster heads to discover services, which reduces the communication cost. Furthermore, the lightweight clustering algorithms build the distributed directory of service registration based on only those neighbours who are one hop away. It reacts rapidly to topological changes and resultantly provides the basis for energy-efficient SD. The protocol also considers low maintenance overhead while reducing the chances of issues e.g., chain reaction problem.

However, solution still exhibits the potential of facing chain reaction problem. Moreover, re-clustering and re-registration are costly in a network with the large number of clusters.

Sailhan: Sailhan [106] is a scalable SDP for large-scale (100s of nodes) MANET. The architecture of Sailhan is structured as a virtual network of distributed and dynamically deployed directories.

In this approach, directories are elected among all the nodes in a network with the criteria of available resources and context parameters. Each directory keeps the information regarding available services in neighbour nodes within a fixed number of hops. The WSDL standard is used to define a service description. Directories co-operate with each other by exchanging their profiles between them. The shared profile consists of capacity of the node and condensed list of service information created using bloom filter technique. A client sends a query for a service to its local directory. At first, local directory tries to find the service in its cache. In case of failure, it initiates global discovery by forwarding the query to selective directories. The criterion of the selection is based on the information provided by shared profiles.

The protocol assumes a MANET with nodes holding the same network interface, with IP-level connectivity using the underlying routing protocol. However, it also poses a prerequisite of some gateway nodes which hold

interfaces to several networks, either ad hoc or infrastructure-based. This defines a bridging between hybrid networks (MANETs and infrastructure-based networks), according to the specific networking capabilities of the wireless nodes [22]. This protocol is designed specifically for MANETs; consequently, it doesn't address the challenges posed by 6LoWPANs.

SANDMAN: SANDMAN [108] is a cluster based SDP designed for MANETs that concentrates on power saving. It groups the nodes with similar mobility patterns into clusters. A Cluster Head (CH) is elected among the nodes to stay awake permanently and responds on behalf of other nodes in the cluster. The other nodes sleep most of the time, and periodically wake up to update CH about their presence and service status. The protocol also offers the load balancing technique to re-elect the CH occasionally.

The evaluation results show that SANDMAN saves energy up to 40% of service requests. The rationale is that more energy can be saved by increasing the size of a cluster, but this also increases the delay. As the cluster size increases, requesting client has to wait more for the sleeping node to wake up. However, the performance can be improved by adaptive selection of idle and sleep times.

Service Rings: Service Rings [67] is a scalable SDP designed for MANETs. The network is clustered into rings, which creates a hierarchal ring structure. The nodes that are physically closed and offer similar services are grouped into rings. Every ring contains a Service Access Point (SAP) and know only about it predecessor and successor rings. SAP in the last ring is called world ring. A node keeps information about all services offered by the ring in its SAP. The protocol devises algorithm for ring restructuring, splitting and merging to optimise ring's structure. Rings use those algorithms to decide dynamically, about their structure, to monitor network traffic efficiently. A chosen SAP periodically initiates a ring check message, which is circulated in the ring to verify the consistency of the network. In the case of a breakage or a partition, the selected SAP starts a ring repair or integration algorithm.

Service rings protocol has not defined any format for service descriptions. SAP in every ring is used to store service information, and are queried to discover a service. It also supports the mobility as rings are maintained with periodic ring check messages. The efficiency of the protocol depends on the effective selection of the nodes in every ring (with the similar services and efficient links between them), and the appropriate size of a ring (neither too small nor too big).

LANES: LANES (A Lightweight Overlay for Service Discovery in Mobile Ad Hoc Networks) [66] is a SDP which is based on Content Addressable Network (CAN) [103] structure. It creates a two-dimensional overlay structure of the network by grouping nodes into lanes. One dimension is responsible for disseminating service advertisements, whereas other dimension distributes service requests. Every node in the lane pro-actively broadcasts its services to other nodes, and keeps the cache of services in the lane. However, the lanes are loosely coupled with each other and reactively forward the queries to other lanes using any-cast.

Service descriptions are distributed to

$$\sqrt{N} nodes$$

where N is the total number of nodes in a network. The protocol is independent of any service description format. A periodic ping message is used to check and maintain the lanes. Furthermore, protocol defines the algorithms to split and merge the lanes to achieve their optimal size.

Virtual Service Overlay (VSO): Virtual Service Overlay (VSO) [94] approach creates a virtual overlay of a network based on service types. It creates a service-centric overlay to provide better energy conservation, instead of network centric model which focuses on the physical location of nodes. Nodes use Service Capability Messages (SCAP) to get the knowledge about network wide available services. Every node broadcasts the SCAP message and maintains a table of other peer nodes offering the similar services. The peer nodes multicast "vDA Interest" messages among them to elect a virtual DA. This elected DA matches, compares and aggregates services provided by the similar peer nodes. In addition, it also searches for the closed match services offered by a resource-rich node in the neighbourhood. The virtual DA is re-elected periodically for load balancing.

VSO emphasises on providing better service quality by aggregating services. It also claims to improve overall energy conservation and congestion control.

3.6.3 Hash-based P2P

The protocols categorised in this class rely on peer-to-peer (P2P) overlay networks that are constructed using Distributed Hash Tables (DHT). The examples include Chord, Pastry, Content Addressable Networks (CAN), Tapestry. DHT is used to store a key-value pair on designated nodes. The hash function is applied to the

message and is passed to the responsible nodes in the fixed number of hops. Every node keeps a routing table with network addresses of neighbour nodes.

The main advantage of using DHT based protocols is their scalable and efficient lookup mechanism. The lookup is performed within O(log(N)) hops, where 'N' is number of nodes in the overlay network [79]. However, the downside is that it generates considerable network traffic and a maintenance overhead, which makes it less suitable for the WSNs [80]. The search is based on a unique identifier, which is only useful for searching services with a unique name. Conversely, effective attribute-based searches are still not supported by DHTs.

Following is the detail of a Tapestry which is based on hash-based approach.

Tapestry: Tapestry [137] is a peer-to-peer (P2P) decentralised architecture, which provides a fault-tolerant, scalable and robust wide-area infrastructure to locate the required services. It efficiently constructs an overlay network using distributed algorithms. It assigns Globally Unique identifier (GUID) to an object's unique identifier. The object ID with a "salt" value is hashed to identify appropriate roots. Tapestry dynamically maps each GUID to a unique live node called the object's root. Tapestry assigns multiple roots to each object. The message to a node is routed to its root. If in the way, any node found with the location mapping to the destination, then it is immediately passed to the node that contains the object. In other cases, the message keeps on traversing the hierarchy towards the root. The root to a node always got the required mapping.

3.7 Directory-less architectures

The protocols with directory-less architecture use multicast or broadcast to discover a service. These protocols incur the overhead when services are discovered. Therefore, this kind of solutions can be efficient in a small-scale network. However, these solutions are not feasible for a medium or large-scale network, as the control overhead can overwhelm the network. Furthermore, node's sleep cycle in constrained networks such as 6LoWPANs can result in a discovery based on incomplete information. Some popular protocols of this domain are described in this section.

Bonjour: Bonjour [118] is Apple's version of Zeroconf [40] networking, which is based on a combination of the multicast domain name system (mDNS) [21] and DNS-SD [71]. It targets service and device discovery among computers and other networked devices. Bonjour provides zero-configuration by assigning IP addresses to the devices, without the aid of DHCP server. A new

service provider in a network uses multicast to advertise its presence to other devices. Clients receiving the advertisement cache service records for a specified time. In discovery scenario, a client multicasts a query with the details of domain name, service type and preferred communication protocol. Service providers with the matching DNS entries respond with service records.

Bonjour generates a significant amount of traffic by using multicast extensively. This issue is addressed by the protocol by using exponential back-off, which increases the time interval between queries and announcements. The reliance of protocol on multicast results in heavy traffic, more energy consumption and congestion. Therefore, bonjour is only suitable for very small scale networks and high bandwidth usage makes it a poor match for resource constrained networks.

uBonjour (Compressed Bonjour): uBonjour [64] is bonjour's compact form, based on mDNS and DNS-SD. This protocol works like bonjour, but combines different messages to decrease the control overhead. Even though mDNS/DNS-SD message sizes were recently optimised for 6LoWPANs [65], uBonjour still relies on the availability of IP multicast and entails more communication overhead. The compression requires more changes and processing at the gateway to decode the information.

Universal Plug and Play (uPnP): uPnP[1] was proposed by Microsoft for device and SD in small office and home environments. The main features of uPnP are its zero-configuration and automatic discovery of heterogeneous devices. It is independent of any programming language or operating system. UPnP uses protocols, including HTTP, TCP/IP and SOAP. Zeroconf feature enables it to assign IP addresses automatically to devices, if no DHCP server is available. Its architecture consists of devices, services and control points. Every device needs to provide an XML device description document, which contains the list of services and properties of the device. A service stores its state information in state table, and supports a control server to update its state table. The subscribers to a service are also managed by its event server. Control points act as directories and are optional. UPnP's discovery is based on Simple Service Discovery Protocol (SSDP), which uses HTTP and SOAP over TCP multicast (query request) and UDP unicast (using a service by specifying URL).

UPnP generally operates better over reliable networks, so it is unsuitable for networks with lossy links and other constraints. It is based on HTTP,

[1] http://upnp.org

and SOAP protocols, which are heavy and verbose for a resource constrained networks. Moreover, it does not support attribute-based querying for services. The extensive use of multicast makes uPnP non-scalable and also exhausts the energy of the devices in a network.

DEAPspace: DEAPspace is a peer-to-peer SDP for single-hop short-range wireless systems [95]. The DEAPspace uses the proactive SD framework. Devices periodically broadcast their services and keep information of all services available in their one hop proximity. Each device also specifies the time period, during which the service is valid. After expiration, the device should renew its service description, or it is marked unavailable for its neighbours.

A performance evaluation [74] shows that time for discovery is better in the case of DEAPspace when compared to other broadcast-based push models. However, the periodic broadcasts consume much of the network bandwidth, which makes DEAPspace unsuitable for constrained devices and networks.

Group-based Service Discovery (GSD): GSD [16] is based on peer-to-peer caching of service advertisements and forwarding of service requests. Service descriptions are defined in DARPA Agent Markup Language (DAML+OIL). It uses the semantic information of DAML to create a service hierarchy to create service groups. The protocol does not broadcast service requests, but instead it exploits semantic information to selectively forward to the nodes with similar services. The more flexible service matching mechanism is also provided by the protocol using DAML format of service information.

Each node periodically advertises its list of services to all nodes in its vicinity (particular number of hops) and caches their service information. The service group information is also disseminated with the advertisement. On receiving a service request, the node multicasts the request to its other group members. The parameters like advertisement expiration period and range of hops can be specified according to the network. Semantic service definitions have merits of grouping of nodes and better service matching, but these also increase the energy consumption and require large packet size.

Konark: Konark [50] is peer-to-peer SD and delivery architecture for multi-hop ad hoc networks. It is based on lightweight HTTP servers, and uses SOAP to handle service delivery. It also defines WSDL based service description language. Konark supports both push and pull modes of SD. The proactive or reactive service advertisements are used on the need basis. A service advertisement specifies its time-to-live and enables its caching on each node.

In Konark, each device runs a lightweight HTTP server and uses multicast to advertise services. Every node maintains a service registry to store service information. The registry is based on a tree structure with a number of levels to classify services. A service request can query for a generic or specific service in each category, so it can carry simple keywords or more precise service description. The multicast is used to send forth service requests to a fixed group; corresponding nodes with the matching service will respond back using unicast.

Service advertisements of Konark allow semantic matching and are lighter than WSDL, but still increases the energy consumption. The use of multicast for service requests is also costly for restricted networks where bandwidth is limited and links are unreliable. Konark doesn't address the issues of energy consumption and delay, which have significant importance in constrained networks.

Bluetooth SDP: Bluetooth SDP [12] targets the bluetooth devices with limited complexity. Therefore, it is optimised and partly addresses SD. It provides a simple API for enumerating the devices in one hop range and browsing through available services. It offers no feature of service advertisement, registration, or invocation. Services are described in attribute-value pairs, which are searched by service type without a priori knowledge of service attributes. The features like selection, access and usage of services are also not defined by the bluetooth SDP. Moreover, services can become unavailable without any notification.

3.8 Cross-layer design

The cross-layer SDPs extend the routing mechanism to find the routes and discover services offered by that node. It is obvious that by piggybacking the service information onto routing messages decreases the number of messages needed for SD and routing. Resultantly, the less traffic increases the available bandwidth, and energy is also saved.

The cross-layer SD was first proposed by R.Koodli [68]. A similar approach based on Ad-hoc On-demand Distance Vector (AODV) [38] extended the functionality of the routing protocol. SD-AODV and SD-DSR [48] extended AODV and Dynamic Source Routing (DSR) to support SD. These approaches outperform the application-layer SDP based on SLP. A proactive routing protocol Destination-Sequenced Distance Vector (DSDV) was extended in the similar way [34]. The comparison has shown that reactive SD-DSR performed best in terms of messaging

overhead, when compared to SD-DSDV and SLP. Some extensions of routing protocols [127] and [51] have also considered the issue of energy consumption.

SPIZ (Service advertisement and discovery Protocol with Independent Zones) [96], a hybrid integrated protocol (based on multiple criteria such as mobility and service popularity) performed better than SD-AODV solution. The extension of On-demand multicast routing protocol (ODMRP) that uses multicast and only resend advertisements if service changes [75]. An approach [49] used High Efficiency Service Discovery (HESED) multicast routing protocol for both service requests and responses. LIFT (Limited Flooding of requests within a clusTer) SDP [134] divides a network into clusters with some High Capability Devices (HCD) as a cluster leader. The SD-AODV is extended with the cluster-based SD approach, which further decreases the energy consumption. Another approach Multi-path Cross-layer Service Discovery (MCSD) [110], finds multiple routes to service agents and provides better service availability and network layer performance.

In summary, the cross-layer design and optimisations can be used to improve the performance of a protocol. The main idea is to control and exchange of information over different layers in the architecture for efficiency. However, there are cons of using cross layered architecture in a WSN. Architecture violations make the protocol hard to maintain by increasing the complexity of later updates. Furthermore, the luxury of designing protocols at different layers independently is also violated by the cross-layer design.

3.9 Discussion from 6LoWPAN's perspective

WSNs are resource-constrained environments, which consist of highly resource and power limited nodes with very low data rate communication. The nodes communicate over IEEE 802.15.4 radio channel and have few kilobytes of RAM and about 100 kilobytes of ROM. The mostly battery operated nodes are not recharged quite often, thus need to run for longer periods. This domain poses more difficult requirements for SD than others. The limited bandwidth and power do not allow heavy traffic and necessitated acknowledgements for guaranteed packet delivery in a lossy environment. The limit to typical packet size is 127 bytes, which is further reduced at the application layer to only 60-80 bytes (Section 2.2.2).

These limitations along with the node's memory and processing capabilities restrict the use of a resource and computational hungry protocols. The nodes also have sleep cycles to conserve power, which needs to be considered by a SDP. These reasons do not allow most popular protocols like UPnP and SLP to work directly for 6LoWPANs. To make those functional, their features should be minimised, and some compression mechanism should be used for their verbose formats. The trade-

off between the computational complexity and compression ratio should be managed for applicability. Some extreme approaches focuses just on performance gain for these constrained networks include the cross-layer optimisations (Section 3.8) and merging of SD and data collection functionality like TinyDB system [77]. Another suitable approach is to use attribute based naming scheme [87], which also work on lower layer in the protocol stack to improve the performance of the network. In addition, other functions like data aggregation and in-network processing can be used, which make use of intermediate nodes to do the aggregation of data to save the power and bandwidth of the network. These approaches only benefit the specified single application and also lose the potential of using layered architecture from extensibility perspective.

Traditionally, there are three types of SDPs: Centralised, Directory-less and distributed directory based approaches. The comparison of different architectures is given in Table 3.1.

The centralised directory and cluster based architectures (Section 3.5 and 3.6.2) are mostly suited for a network with the element of heterogeneity. These architectures exploit the power of more resource-rich devices by making them servers, to assist the other nodes by sharing computational load. In some approaches, the gateway registers the nodes in a network using special optimised protocol [19]. It acts on behalf of registered nodes and can support multiple external SDP architectures. The distributed hash-baed P2P (peer-to-peer) approaches (Section 3.6.3) can also be employed in WSNs. This type of approach treats all nodes equally, but uses multicasts and broadcasts, which can overwhelm the network by generating heavy traffic.

Directory-less protocols, as the name suggests, work without any specific nodes to act as registries and make use of flooding mechanisms to discover a service. In this approach devices cache the service information of the neighbours to enhance the SD process. UPnP, Bluetooth SDP, DeapSpace and Konark are popular SDPs of this domain. However, the extensive use of multicast or broadcast make this kind of SDPs unsuitable for the resource constrained 6LoWPANs.

Distributed architectures consist of multiple registries, which act as a service information container of their vicinity. These registries collaborate between them by passing the request to each other for discovery. CS4D, Service rings and Lanes are the examples of this technique. Sailhan has optimised the forwarding by periodic information sharing among registries. Other approaches, including SANDMAN conserve energy of the nodes by keeping the nodes sleep while keeping their cluster heads awake to respond on behalf of them. VSO approach has defined the criteria of creating clusters based on the type of services, rather than node's locality. This method can be useful in scenario where similar kinds of services are available in

Table 3.1: Comparison of SD architectures

Architecture	Number of registry	Discovery of registry	Pros	Cons
Directory based				
Centralised	One	Multicast	Unicast for discovery	Single point of failure High status maintenance traffic Limited scalability Limited scope (local domain)
Distributed			Scalable	
DNS	Multiple	Pre-selected and static	Query travels the hierarchy of registries	Requires DNS infrastructure Verbose DNS queries and format
Clustering	Multiple	Dynamically Elected and Re-elected	Query passed between registries	Re-clustering for load balancing Chain reaction problem
P2P	Multiple	Each node has a hash table	Hash function is applied to query, which passed amongst the nodes to reach the required host	Considerable traffic overhead No attribute-based search is possible
Directory-less	No	Each node	Multicast or broadcast	Extensive use of multicast/broadcast High control overhead No information is kept regarding sleeping nodes

different clusters, and most of these services are queried together. Distributed SDPs save energy in the case of service status maintenance, but overhead of creating clusters can overwhelm the network. Moreover, the chain reaction problem further argues against the clustering for a SDP for 6LoWPANs.

The directory based infrastructure can be more feasible for 6LoWPANs. The adaptation layer needs an edge router to connect a PAN to the Internet. Edge router can be used as a main directory for 6LoWPANs. The well-known industry-standard IP based SDPs include SLP, UPnP, JINI and Salutation. Table 3.2 shows the comparison of most popular SDPs with respect to 6LoWPAN SD requirements. All these protocols are not directly applicable to constrained 6LoWPANs and most of them are based on TCP/IP, which creates further problems in their adaptation [89]. JINI lacks the versatility as it is language-dependent and needs service information in the form of Java objects. In the same way, NanoSD assumes that each participating node has enough memory to keep the cache of service information. However, the extensive use of multicast and broadcast make this kind of SDPs unsuitable for resource constrained 6LoWPANs.

There are some efforts, which have tried to adapt the SLP for 6LoWPANs. NanoSLP, a miniaturised version of SLP, is an attempt to use SLP with the reduced features and only supports the directory-less architecture. Another attempt was the SSLP [63], which proposes a compact version of SLP for 6LoWPANs. A proximity-based approach [18] which uses DPAs (Directory Proxy Agents)as local registries has employed SSLP inside a 6LoWPAN and provided the interoperability with SLP by using a Translation agent (TA). However, this solution involves complexity and delay of translation, each time message is translated to or from SLP. Furthermore, the casual sharing of resources between distributed directories increases the traffic burden. uBonjour [64] is bonjour's compact version, based on mDNS and DNS-SD. Even though mDNS/DNS-SD message sizes were recently optimised for 6LoWPANs [65], uBonjour still relies on the availability of IP multicast and entails more communication overhead for service registration and status maintenance. ENUM-based approach [7] also requires some master nodes. This technique introduces compression for 6LoWPANs, and has assumptions, which depend on the availability of the required information and database entries. A TCP/IP based web portal's mechanism [104] is employed using regional locals (master nodes) for the local SD and data servers are used in a network, which provides a complex mechanism which increases cost burden. RSDPP (Real-World Service Discovery and Provisioning Process) [44] is an industry focused SOA (Service Oriented Architecture) based middle-ware solution that uses SOAP-based or RESTful web services. DiscoWoT [84] provides RESTful web services using HTTP based SD with existing or injected strategies. However, this scheme has

Table 3.2: Comparison of SDPs with 6LoWPAN Solution requirements

	Bonjour	**SLP**	**JINI**	**Salutation**	**UPnP**	**Bluetooth**
Adaptive	No	No	No	No	No	No
Compact Size	uBonjour	SSLP	No	Salutation Lite	No	No
Compact message	No	SSLP	No	Salutation Lite	No	No
Sleepy nodes	No	Yes	Yes	Yes	No	No
IP usage	Yes	Yes	Yes	Yes	No	No
Heterogeneous	Yes	Yes	Yes	Yes	Yes	Yes
Interoperable	DNS	No	No	No	No	No
Energy efficient	No	No	No	No	No	No
Service Selection	No	No	No	Yes	No	No
Service Composition	No	No	No	No	No	No
Scalable	Small	Small to medium	Small to medium	Small to medium	Small	Small

not described any solution for the management and status maintenance of the registered devices. The IETF Resource directory [113] uses CoAP as an underlying communication protocol for SD. In summary, a compact and efficient SD solution still required for 6LoWPANs.

In summary, a research gap for an optimised SD solution for 6LoWPANs still exists. This project proposes a compact and optimise registry based SD solution with context awareness for the IoT, which is more focused on constrained domains such as 6LoWPANs. It uses CoAP-based [114] RESTful web services to provide a standard interoperable interface which can be easily inter-worked with HTTP. The modular design of the protocol features allows its implementation on constrained devices. High capability devices can benefit by implementing profiles to share the load of other devices. Thus, it allows the productive usage of network resources.

3.10 Service Discovery (SD) Process

This section discusses different stages of SD process.

3.10.1 Service description

Every SDP needs a way to describe services. Therefore, a service description mechanism is essential for any SDP to function. Different protocols use unlike formats of service description to enable users to search for a particular service by setting fields and values according to the format. A service description generally contains the identity and type of a service with optional attributes.

Generally, SDPs use classes to classify services into service types. Each service class (e.g., print class for print-related services) contains a list of attributes, which are assigned with values to identify the instance of the class. A service description is usually stored as a set of attributes/values pairs [79]. The same approach is used by various SDPs, as all use distinct naming and conventions. Some protocols including Salutation, SLP, JINI and UPnP store the attribute/value pairs as lists and do not relate attributes to each other. On the other hand, INS/TWINE [11] relate attributes and store them in a tree-like hierarchy based on a dependency between them. In Salutation and SLP, predefined set of attributes can be used for each service type. Many approaches, including UPnP, Web Services and TWINE use XML based representations to provide openness and extensibility.

Features like naming, invocation and status query are closely related to service description. New services must be identified by an unused name. In order to avoid the naming conflict, many protocols define attribute naming conventions. These are used to find a service provider according to the query requirements. Some protocols like INS use special conventions, e.g., if some attribute is not set, it is inferred that it can take any value. Different protocols can use various vocabularies for the same service, because of the lack of compatibility between the protocols.

3.10.2 Service advertisement/registration

The host nodes are required to advertise their services to make them discoverable by consumers.

SDPs manage service descriptions in different ways. Some SDPs store service descriptions only in node's local cache. In this case, the query initiates the search that floods the whole network to find the required services. Another similar approach is to store the service information of all services on each node of a network. This way advertisements can flood the whole network, thus is not feasible for nodes with low capabilities. However, the search becomes straightforward, as a node simply has to look into its local cache. In some approaches, this process is optimised by only storing service descriptions in a subset of a node collection. In this way, advertisements are distributed to n hops or multicast to a specific group of nodes. By using multicast instead of broadcast and storing the advertisement

nearby (e.g., just one hop), protocols reduce the memory requirements and increase the probability of finding a service in requester's vicinity. However, multicast is still expensive in terms of traffic, as creation and maintenance of multicast groups will introduce new overhead especially in multi-hop networks.

In some protocols, service descriptions are stored in special selected nodes, which are called directory nodes. These nodes are responsible for keeping service descriptions available in a network, facilitating SD. The directory nodes can be preselected designated nodes (e.g., in SLP and JINI), or can be selected dynamically according to the network conditions, as in Service rings. In the case of dynamic selection, each group chooses a node as group head, which keeps the record of service description of its group members (Section 3.6.2). The subset of directory nodes creates an overlay which reduces the network traffic in advertisement and discovery phases.

In the case of directory-based architecture, the registration procedure starts when nodes register services with directories. On the other hand, nodes in directory-less architecture, forward or cache the service information of each other. The directory-less approach is mainly based on a flooding mechanism using broadcast or multicast, which is unsuitable for constrained environments with mostly sleeping nodes both efficiency and scalability perspectives. While the directory-based approach eliminates these issues, but requires status maintenance overhead and can limit the scalability. In summary, the directory-based technique with some optimizing techniques to deal with the issue can be a better candidate for resource constrained environments.

3.10.3 Service discovery

The discovery stage is responsible for searching the relevant registered services in a network that meet the query requirements. The search in networks without directory nodes depends on the way services get registered.

The discovery can be done with active or passive search. The active search floods the whole network with the search message; this is the case where services are registered in local cache of nodes. In SDPs which store service information on all nodes, search will only look into the localised cache. This type of discovery is a passive search, which is efficient in detecting the changes, but expensive in terms of memory and traffic requirements to keep the record of all updates of services in a network. On the other hand, the active search only floods the network when SDP sends the search message, so it has an advantage of less network traffic. Some SDPs support both active and passive search [50, 124]. In another approach, SDP first looks in the node's local cache and then selectively forwards search requests.

This selective forwarding technique can depend on some ontology description [17]. In the case, when a node has not enough information, it can simply broadcast the requests to its neighbour nodes. This approach scales well compared to simple flooding, but the high mobility can cause problems when a node moves from the targeted region.

In directory-based solutions, all service providers register their services to directory nodes. In this case, query-based approach is used. The client queries to the directory nodes and receives an immediate response. In Service rings, the protocol searches only Service access point (SAP) nodes. In Lanes, groups of nodes form an overlay network that creates lanes of nodes. Each group is called a lane, and nodes in the same lane share one directory replicated in cache of each node. In this way, client can query to any of the nodes in the lane, instead of one specific node.

3.10.4 Service selection

SD can result in a number of matched services according to the request query. The service selection phase, not addressed by most of SDPs, helps to determine the appropriate service from the discovered service list.

The process of service selection can be manually completed by the user, or it can be performed using optimisation algorithm implemented on directory nodes or on the client side. In an automatic protocol, selection is based on some criteria or metric. The best offer metrics can consider different factors, including route specific (e.g., hop count, bandwidth, delay) or service specific (e.g., server load, remaining energy, capacity) [129]. In literature, approaches have considered some contextual information e.g., lowest hop count [123], best rendezvous point (RP) [37] etc., as a criterion for service selection. In Intentional Naming System (INS) [1], a service lookup is resolved to a service location at a delivery time. It adopts the approach in which an application defined service weight is used to select a best-suited service from the list. This approach requires service side knowledge and also helps to balance the load among services. However, there is a probability that user will not get the intended service because a user specified it in attributes. In general, most of the SDPs simply ignore the selection support.

3.10.5 Service Invocation

SD is completed when an application gets the response with a service identifier and address of its host. However, the client who initiated the query needs to invoke a service using its service interface. Service invocation varies in different

protocol scenarios because each protocol defines service interfaces in a distinctive way. Existing SDPs, provide three varied support levels [139].

1st level At the first level, protocol only provides service location, and leaves the job of communication and operation to applications.

2nd level The second-level protocol defines the underlying communication mechanisms as well, for example; Salutation uses RPC and JINI use download-able Java code. UPnP and WSDL [23] communication mechanisms are based on XML, SOAP, and HTTP protocols and formats.

3rd level At the third level, UPnP and Salutation define application operations specific to an application domain in addition to the communication mechanism and service location.

3.11 Challenges

The design of a SDP needs to consider several aspects for the provision of service publishing and searching. This section covers the details of major issues related to SD with the relevant solutions.

3.11.1 Scalability

The scalability issue arises with the increasing number of services and clients in a network. The control overhead of a SDP can overwhelm the network by generating burden due to dynamic interactions in the large-scale network.

The directory based architectures with the load balancing and query efficiency can significantly increase the scalability. Multiple directory nodes to handle service registration can avoid the problems of bottlenecks by managing service registrations. In the case of a homogeneous network, directory nodes can be re-elected to increase the lifetime of nodes. Service grouping with multiple directories can further guarantee the elimination of the bottleneck problem by assigning different directories to specific nodes. In this regard, SLP deals this by proposing the concept of `scopes` and Virtual service overlay (VSO) [94] allocates unique directory agents for service groups.

The caching approach increases scalability by keeping the values of services on the directory nodes, and responding on behalf of nodes to avoid overloading nodes [15]. The number of directory nodes can also be increased, if the current load is difficult to handle by the directory nodes. DNS based hierarchical structures ensure the scalability of a large-scale network.

3.11.2 Energy, Memory and Bandwidth Constraints

The focus on energy, memory and bandwidth limitations is of great concern in resource constrained wireless sensor networks. Most of sensor nodes consist of 8-12 KB of RAM, about 100 KB of ROM and a battery as the main source of energy. Consequently, the lifetime of a constrained network directly depends on depletion of the batteries of the nodes. The network bandwidth (20-250 Kb in 6LoWPANs) and maximum packet size (127 bytes in 6LoWPANs) are limited in the wireless sensor networks. A SDP should ensure that the scarcest nodes are capable of implementing the protocol, and its messages are compact enough to fit in a single packet.

3.11.3 Reliability and accuracy

The reliability refers to fault tolerance, failure recovery, and provision of accurate and up-to-date service information.

FRODO provides support of a redundant backup directory to prepare for the recovery in case of failure of the main directory. In the distributed directory architecture, multiple directory nodes are maintained, which collaborate and re-elect new directory nodes in case of failure. Another approach is to use multi modal functioning as SLP switches to directory less approach if no directory node is available .

Proactive mode and negative announcement (advertising unavailability) can be used by SDPs to provide accurate and up-to-date information. This issue becomes more important to be addressed when a network has nodes with sleep cycles or high mobility. In addition, soft states of service description and caching are used to get an update in a specified time interval.

3.11.4 Heterogeneity and resource-awareness

SDPs need to support the integration of heterogeneous nodes in a network. Resource-awareness enables a SDP to delegate different roles to the nodes depending on their capabilities, so high-capability nodes can share the burden of constrained nodes in the vicinity. For example, FRODO ranks the devices into three levels and allocates the jobs accordingly, and Splendor uses proxies to offload the low capacity nodes.

3.11.5 Security and privacy

Security is the important issue which is mostly overlooked by SDPs. There are multiple aspects of security, including authentication (verifying an identity),

authorisation (granting permission), trust (level of confidence), confidentiality (encrypted information), and Integrity (data items are not amalgamated). The privacy, on the other hand, is the degree to which a user will allow the environment to interact with it. JINI and SLP define some security features, whereas Splendor provides a solution to both security and privacy.

3.12 New perspectives

The conception of WoT has made SD an indispensable function for meaningful integration of WSNs with the Internet. Brand new perspectives in the SD process can be conceived by inclusion of context awareness and adaptivity in response to the future demands of searching most appropriate services. Therefore, a SDP that adopts these new perspectives can increase the efficiency and applicability of WSNs. The details of these paradigms are covered in this section.

3.12.1 Context Awareness

> Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application [28].
>
> Context is not simply the state of a predefined environment with a fixed set of interaction resources. It's part of a process of interacting with an ever-changing environment composed of reconfigurable, migratory, distributed, and multi scale resources [26].

Context awareness is the key to provide most appropriate services by exploiting the context information which includes user's preference, location and present environments. A SDP with context awareness can offer more intelligence, which results in efficient SD. In this era of dynamic and heterogeneous WSNs, it is a challenge to provide a service selection mechanism in a SDP. The pool of services offered by different available devices demands an intelligent process of suitable service selection for user tasks. Context information offer the intelligence in decision-making by filtering the unrelated services.

A research effort [53] insists that most of the algorithms and architectures can only work for specific applications, and context awareness enables the adaptability of the dynamically changing environments. Tandem [78], a context-aware clustering method, which re-clusters in case of topological or contextual changes to increase the responsiveness. AlarmNet [135] adapts to individual context and behaviour patterns in assisted residential monitoring to manage the power. CALEEF [107]

(Context Aware Lightweight Energy Efficient framework) has HTTP and XML peer-to-peer communication among sensors, and uses a context model to optimise the polling time of sensors to reduce the energy consumption. A context-aware resource management framework [73] deals with the heterogeneity and mobility of devices in smart homes. Another effort [19] uses context information in a 6LoWPAN-based SD architecture that reduces the traffic overhead considerably by finding and using the closest service. A recent approach [130] has also considered context-awareness for the IoT.

3.12.2 Adaptability

The context awareness provides information, which enables a SDP to become adaptive. Most of the SDPs have not considered to maintain the context information which has potential to provide the features of better service selection and adaptation to dynamic changes and demands. Some SDPs offer context-awareness, but the approaches, mainly concern about service selection without being adaptive to a dynamic aspect. However, the context information can be exploited by a SDP to decide dynamically about different protocol parameters (number of hops an advertisement can travel) and switch between modes of operations (pull or push method). The protocol parameters like advertisement frequency and cache lifetime, etc. are mostly fixed by a SDP, which never changes in the entire operation. Dynamic networks with different types of nodes and changing services, demand for an intelligently diversified protocol, which can adapt to changes and results in increased efficiency.

The PULL and PUSH based discovery approaches can be used alternatively by a protocol depending on the popularity (increasing demand) of services. The protocol can poll the sensors for more demanding services, and can assign a periodic or threshold based push method to nodes with fewer numbers of service queries. The decision equation can also consider the network conditions (e.g., congestion). This dynamic decision-making increases the autonomy of a SDP.

SPIZ (Service Ad/D protocol with independent zone) [96] is a lightweight adaptive cross-layer based on dynamic switching between pull and push-based approaches. Each node in SPIZ has its own zone to facilitate local adaptation to dynamic changes (based on multiple criteria such as call rate, mobility, service popularity, etc.), to reduce the overhead and support scalability. AVERT (Adaptive SerVicE and Route Discovery ProTocol for MANETs) proposed in [128] extends the SPIZ by adapting the rate of proactive messages sent by nodes.

Adaptive middleware for smart home environments [54] ranks all available service providers to choose the appropriate one. This middleware matches the

quality of context (QoC) requirements with the QoS attainable with the sensors. MidFusion [4] discovers and selects the best set of sensors that maximises the potential of reaching application goals. It maps the QoS availability of the sensors to a utility function based on the QoS parameters of the application and the cost of information acquisition.

3.13 Summary and Discussion

SD is instrumental in the provision of services from 6LoWPANs for the IoT paradigm. This chapter has scrutinised the literature to get the insight into the perspectives addressed, and gaps left by existing solutions. The chapter started by discussing the basics of SD and then categorised existing solutions while explaining and analysing them for 6LoWPANs. The SD process is discussed in the context of existing literature and important existing techniques to deal with the issue at each level of SD are explored. In later part, this chapter focused on the general challenges exist for a SDP and new perspectives that can improve the performance by coping with the dynamics of a network.

In summary, there is a plethora of solution exists in literature, which generally can be categorised into two classes depending on their focused technology: IP-based, non-IP based solutions. The existing IP-based solutions are already working and thus become an obvious to be selected. However, these solutions are not designed to target the issues and challenges of 6LoWPANs. Therefore, these solutions can't be used directly in constrained environments. Several efforts have tried to design the compact versions of existing standards, but those need application gateways for translation of messages. Furthermore, their choice of communication is mostly is multicast or broadcast that incurs high control traffic, which is unsuitable for constrained wireless environments with lossy links and limited bandwidth. On the other hand, the non-IP based protocols are mainly performance-centric for a single application, so these solutions from their roots are in contrast to the philosophy of the IoT vision. However, there are many aspects that have been addressed in parts by existing solutions at different levels of the SD process, which are important to be considered for a new solution. New perspectives, including context-awareness and adaptability are the key approaches to deal with the dynamic nature of IoT environments.

The next chapter presents some IoT scenarios to emphasise the challenges and requirements that are important to be considered by a SDP.

Chapter 4

Service Discovery for the IoT: A Requirement Analysis

4.1 Introduction

The IoT vision integrates heterogeneous devices and networks together to form a new collaborative paradigm. This creates new opportunities for co-ordination of end-to-end communication between devices from diverse networks. The idea of wrapping up a functionality into service and providing an interoperable way for invocation revolutionises our perception regarding isolated constrained networks. SD plays a key role to assist the applications and users to locate services. However, there are many challenges that are needed to be considered while devising any solution for IoT environments. The complexity is caused by the inherent diversity, which demands for the trade-off between generality (to deal with range of applications, devices and networks) and efficiency (for constrained domains). This requires a framework that should be evolvable to embrace the new challenges.

This chapter introduces some use-case scenarios in a IoT environment and highlights the involved challenges and requirements.

4.2 Scenario

This section explains different scenarios to emphasise and analyse the role of SD solution for the IoT paradigm. Firstly, this section covers the context of scenarios to give an overview of the situation.

Ipswich university (fictional) has transformed its traditional campus to an intelligent and automated one by installing new sensors in every building and room. The University has also created and distributed an application, to enable its staff and students to use features according to their defined authority. Furthermore,

the application is smart enough to use the sensors on hand-held devices of users to provide sensing information in a region. It also allows other user applications (personal management applications) to get access into the system. The whole system empowers the users to utilise functions ranging from finding the parking space to automate closing down their office windows.

Their system uses different policies to balance the trade-off between the network overhead and the prompt provision of services, depending on the time and demand. During the day in term time, system is scheduled to run on a full mode. This mode is more focused on reducing the delay time of SD and invocation. In the term's night mode, system follows the security mode which only focuses on decreasing the system overhead while providing the security breach alerts. The system also has scheduled off-peak policies for weekends and holidays. Furthermore, it allows a power-user to add/amend policies depending on the current needs, e.g., in the case of some emergency, etc.

Following are few situations where different users utilise the network:

Auto management of personal places with preferences: Michael is a senior lecturer in Ipswich University. He spends time mostly in any of his two offices, and in his house. His mobile usually alerts him on reception of new emails, SMS, and tweets. He has installed the provided application to use features of his University's smart system. He uses it to schedule the heating system of his office, to get the alerts of opened windows, and to actuate set of tasks that are performed on occurrence of some event (he is coming or leaving office). However, he is more interested in controlling his house in the same way.

He installs "My iplaces" application, which allows him to save and view his common places. The system of university authorises his application to log in and acquire various services as a registered user. The application allows him to see different status, e.g., temperature values and doors open/close. He can also set alarms based on various rules specified by him. In addition, he can create sets of commands to execute e.g., When no one is at home close all windows and notify me on security breach, etc. He can automate the whole procedure (e.g., leaving home) as well by allowing application to execute the commands using his context, including current time and his GPS location.

Michael has different demands when he is at home. His home lights are already intelligent enough to switch on when any motion is detected. However, he is more interested to adapt to current available light of a room to adjust their brightness using light sensor and only switching on when he is around. He has also specified different commands comprising a set of tasks, e.g., Set

cinema environment command adjusts the blinds and lights, and changes the TV settings. The application enables him to add any place, e.g., vehicle, garden, study room, bedroom, etc. and to browse through the available services in those areas to create set of commands.

Monitoring and management of places: Mark is a security personnel in Ipswich University. He has installed the University's application on his phone. University's system allows mark to get security related alerts. It recognises him as a power user and allows him to load and change the policies based on different circumstances.

Assisting users to find the best matched services: George is visiting the Ipswich University. He uses his "My iplaces" application to discover and add the Ipswich University into his list. He logs in as a guest and tries to find the nearest parking space by giving his GPS location. Then he enters a building for his meeting and searches for available printers around. He selects the closest printer and issues a command to it.

4.3 User Interactions

The framework of the solution is required to support following three interaction models to materialise the discussed scenarios.

Direct interaction: Applications or users directly access sensors and actuators (in a server/client request-response model). In all scenarios, user applications need to interact with the host devices to actuate a command.

Continuous monitoring: A continuous stream of data is required at regular intervals (e.g., Smart meters). This kind of interaction is required where a user needs to monitor some sensed data, as for security personnel in the scenario.

Conditional monitoring: The updates are only sent when certain thresholds are surpassed or some predefined event happens (e.g., temperature alerts). This is a requirement for Scenarios 1 and 2, where user needs to be notified on occurrence of some event.

4.4 IoT Service Discovery Requirements

The new IoT scenarios (Section 4.2) present a paradigm, where the plethora of applications discovers and invokes services hosted by IP-enabled smart devices.

The SD solution needs to deal with a wide variety of devices ranging from very constrained capabilities (e.g., RFID tags, sensor nodes) to resourceful ones (e.g., smart phones, printers, etc.). This section explains SD requirements extracted from the scenarios.

4.4.1 Heterogeneity and Interoperability

Following are some points in the context of inherent heterogeneity of the IoT and required interoperability:

- The system consists of heterogeneous devices and new devices can be integrated into the system as well. Consequently, the solution should not assume anything about the involved hardware and allow the range of different devices to be a part of it.
- Interoperability becomes crucial because of the underlying diversity. Thus, a SD solution should offer standard interface for services. Furthermore, solution should be routing and MAC protocol independent to allow the underlying networks to select the one which suits them.
- The solution should accept service description ranging from very simple URLs to more concise form of XML or JSON descriptions.
- The devices can be mobile as well, so solution should specify a way to register explicit detail of device's mobility to deal with them in a different manner.

4.4.2 Context-awareness

Context-awareness is an important factor to be considered in the IoT paradigm [100]. The scenarios have emphasised the need of context-awareness, as each user is interested to find services depending on his/her location. The challenge is to allow users to discover more relevant and appropriate services based on the available context information in a transparent manner. Following are the key requirements gathered from the IoT use case scenarios:

- The solution should allow the user to search a service depending on some specified context. There are different perspectives of context, e.g., relatedness depending on location and user authority, battery, demand of service, etc.
- The minimum context information should include location. This can aid the user application to define a query to discover a service in the proximity or in a specific location.

- A service selection mechanism should be offered where in case of discovery of multiple services, a user seek assistance in the selection of an appropriate one. The benefits of service selection are many: it aids the user application to get a best matched service and it also enables the efficient usage of resources e.g., by choosing the closest service in terms of number of hops.

- The solution can make the process even more efficient by saving the user context; for example, usual interest of an individual user to offer related services. This can enable the solution to recognise the multiple requests from users and can adaptively facilitate them.

4.4.3 Adaptability

An adaptive SDP can tackle several challenges by adapting to dynamics of the network to change different parameters, e.g., status maintenance interval, etc. The discussed scenarios introduced the concept of configuration profiles (set of attributes), which can be changed by an adaptive solution dynamically w.r.t. current usage of a network. Following list recognises some of the potential adaptive features that a SDP can provide:

- In SD, status maintenance overhead is a burden on the network that can be marginalised by adapting to the current demand of a network.

- The solution can allow adaptive caching, so service invocation delay can be reduced by using cached values. Constrained networks can be easily overwhelmed by the high number of invocations; therefore, in these networks the energy and bandwidth can also be saved by providing the cached values for services in high demand.

- The solution can allow to schedule and run different policies and configurations based on the dynamics of a network.

4.4.4 Constrained networks

The 6LoWPAN standard enables constrained networks to become an active part of the Internet. However, this also increases the number of challenges involved in designing a solution for constrained networks. This section explains the constraints that should be considered by a SD solution for 6LoWPANs.

Scalability: The solution which can manage and control discovery overhead in a dense network, while keeping track of the availability of service hosts.

Compact size: The complexity of the solution should be implementable on the resource-constrained devices.

Compact communication: The limited bandwidth (20kbps to 250 kbps) and packet size (127 bytes and 60-80 bytes) at the application layer of 6LoWPANs, demands a careful use of bandwidth. Large packets will get fragmented and result in multiple packets, which can overwhelm the entire network.

Sleeping nodes: Constrained nodes usually save their limited power sources by sleeping most of the time, using a RDC. The solution should assign high priority to this issue, by giving some alternatives like proxy which can act on behalf of sleeping nodes.

Interoperability: Different standards for lower layers are being used in 6LoWPANs. The solution should be inter-operable to work between networks with diverse standards. Interoperability aids the application development, as single application can be used across different networks.

Energy efficiency: This is the key to increasing the lifetime of the network. The solution should optimise its operation by minimizing its control overhead. Furthermore, most of the nodes in multi-hop networks waste their energy by passing the message towards the sink.

Resource-awareness: The assumptions of multiple resource-rich devices should not be forced. However, the protocol should be opportunistic enough to exploit the availability of high-capacity devices .

Service composition: Many services are discovered and invoked together, mostly depending on their vicinity. For example, switching lights on and off in a room will require all the actuators of lights to be called collectively. A SDP should consider this aspect to allow grouping of different services in a locality, to offer new services.

4.5 Summary

The analysis of the IoT requirements for SD provide the basis to propose an efficient solution for this emerging domain. This chapter covered these requirements in the context of future use-case scenarios. Furthermore, the application paradigms and different user interactions are also explained to elaborate the points for a user oriented design of a SD solution. The requirement analysis has given more emphasis to the needs of 6LoWPANs by explaining the implicit challenges of this

constrained sub-domain of the IoT. The chapter is concluded by summarising the design choices for a SD solution with the future needs for IoT environments.

The next chapter discusses the first contribution, a context-aware SDP for the IoT, proposed by this research project.

Chapter 5

A Context-aware Service Discovery Protocol (SDP) for the IoT

5.1 Introduction

Chapter 3 analysed existing SD solutions and their shortcomings, including lack of interoperability, verbose formats and heavy bandwidth demands. Chapter 4 presented the requirements analysis of SD for the IoT. Both chapters have given more emphasis to the challenges of constrained sub-domain of the IoT. This chapter describes the design of TRENDY, a new SDP proposed for the IoT. The protocol aims to provide a SD solution to satisfy the identified requirements in Chapter 4. Interoperability is the key to deal with the heterogeneity of involved devices and networks. This chapter explains the architecture of TRENDY, and covers the protocol details related to several aspects of SD, including service description, discovery, selection and invocation.

5.2 Architecture

A registry-based architecture is employed in TRENDY that is more suitable to cover the scalability, reliability and efficiency requirements for a SD solution (Section 3.10.2). Following are the points that become the basis of this choice:

1. The IoT paradigm allows people and things to be connected any-time, any-place, with anything and anyone, ideally using any path/network and any service [126]. Consequently, the queries from the user applications from

outside the network is the norm in such environments. This requires a place to deal with these queries, and certainly a directory-based approach will serve the purpose.

2. This architecture is a better match for 6LoWPANs, where nodes mostly sleep, and broadcasting or multicasting is an expensive way to discover a service in the case of a directory-less solution (Section 3.10.2).

Figure 5.1 presents TRENDY's architecture. It consists of: the DA (Directory Agent), SAs (Service Agents) and UAs (User Agents). This section further covers the detail of these entities.

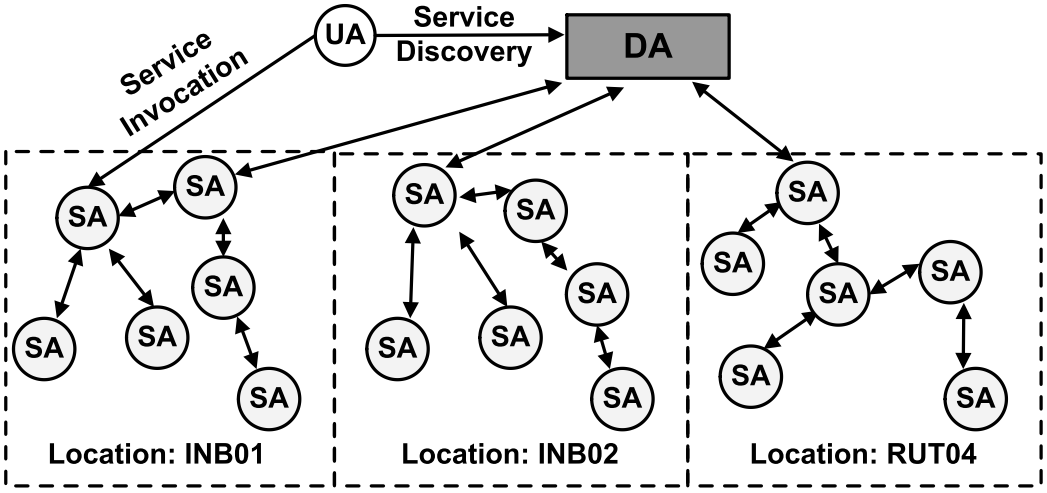


Figure 5.1: Architecture of TRENDY

5.2.1 Directory Agent (DA)

The DA has a backbone role in TRENDY's architecture and serves following responsibilities:

- **Registry:** The DA maintains a registry table to store soft state entries for registered services and context information by the host devices.
- **Status maintenance:** All registry entries bound to a lifetime period specified by the size of the DA's time window. Host devices need to update their status at the DA or their entries are deleted to ensure reliability.
- **Service discovery and selection:** The DA responds to SD requests and uses collected context information to select a better service.

- **Service Invocation:** The DA can act optionally as a proxy, in this case it will respond on behalf of other devices and other benefits like caching can also be exploited.

5.2.2 Service Agent (SA)

A SA is any host device that registers its service descriptions and context information at the DA. It receives a time window length in response to a registration message, which specifies the maximum time to send a status update to the DA. All SAs need to keep the DA updated about their availability by sending status updates within a time window.

5.2.3 User Agent (UA)

A UA is a client interested in discovering services available . It sends discovery requests to the DA, and can be located either within a sensor network or elsewhere in the Internet.

5.3 Communication Protocol

The RESTful web service paradigm is more suitable for embedded constrained networks because of its simplicity and efficiency (Section 2.4.4.2). Furthermore, CoAP (Section 2.5) is a compact and interoperable alternative to HTTP and can be utilised to enable this paradigm in constrained networks [25]. Therefore, CoAP is employed by TRENDY as a default communication protocol between devices. All SAs and UAs can communicate with the DA using either CoAP (default) or HTTP. However, SAs in constraint networks such as 6LoWPANs will require proxies to offer interoperability to HTTP. Moreover, the DA can be extended to present its services in other formats (depending on its capability) and can respond to queries sent using other protocols e.g., DNS-SD.

5.4 Service Description

Service descriptions are important to provide a better SD by giving a more detail for a service. In the IoT, there can be diverse requirements on the level of information needed to be specified by the devices depending on the context of the application and capabilities of the devices. There are several possible service types that can be used by the IoT applications. This suggests the need for such a solution that can accept different service descriptions ranging from very brief to semantically

detailed ones. However, existing versions of service descriptions use verbose XML and other formats (Section 3.10.1) which are unsuitable for 6LoWPANs. The compressed formats like EXI etc. pose the additional requirement for translation to be done on each device (Section 2.6.2).

The solution focuses on the extensibility factor as well as the challenges of constrained networks (Section 4.4.4). Therefore, TRENDY has designed an extensible strategy for service descriptions to balance the trade-off between size of the description (large packets entail fragmentation) and an application's requirements. It defines a default compact format for service descriptions consisting of only semi-colon separated URLs of resources offered by a device. This simple format of resource description is a compact and efficient choice for constrained environments. Additionally, the use of optional IETF Core Link Format (Section 2.5.7) or any other format can also be used to describe extra semantic information by agreement of the end-points. However, the devices initially need to register their service using the default format and if the DA allows other formats, then more description is sent. Figure 5.2 shows the examples of some service description formats being received by the DA.

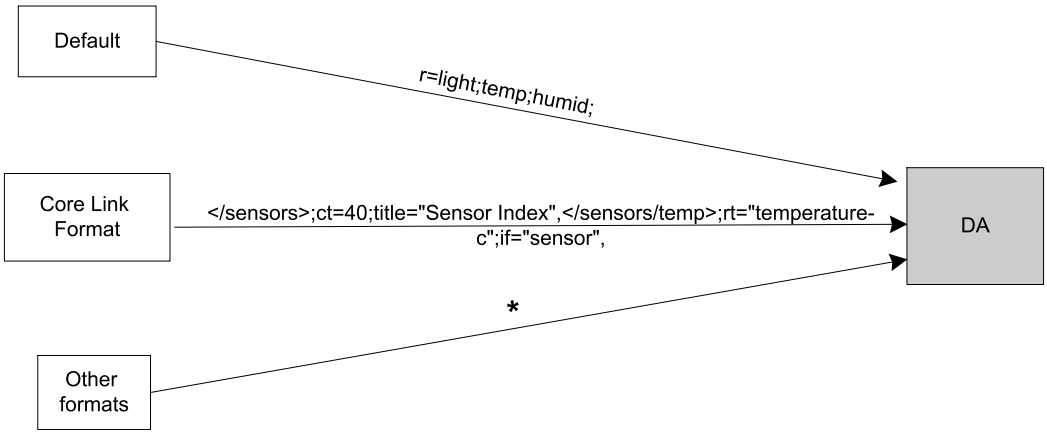


Figure 5.2: Different service descriptions formats are allowed in TRENDY

5.5 Context awareness

The importance of context-awareness is covered in Section 3.12.1 and detail of its importance for IoT environment is emphasised in Section 4.4.2.

Context awareness is a prime requirement for a IoT environment, where a user application usually looks for assistance to get a single relevant service in response

to a query. However, there are plethora of context attributes that can be important for context-aware decision making. Figure 5.3 shows some of the potential context attributes that are useful to be considered in IoT environments.

Any fixed way to define these context attributes can restrict the framework and extensibility required by the IoT. Thus, the solution has only defined the basic simple format for defining context attributes to be used by default, while allowing other formats to be used by using CoAP options. The format of some context attributes is defined, including location, battery consumed, battery source, mobility, hop-count are defined (Figure 5.4), which are important in most of the scenarios. The format detail of these context attributes is covered in Section 5.11.1. This format is evolvable as more context attributes can be easily added by updating the DA to accommodate the application and environment requirements.

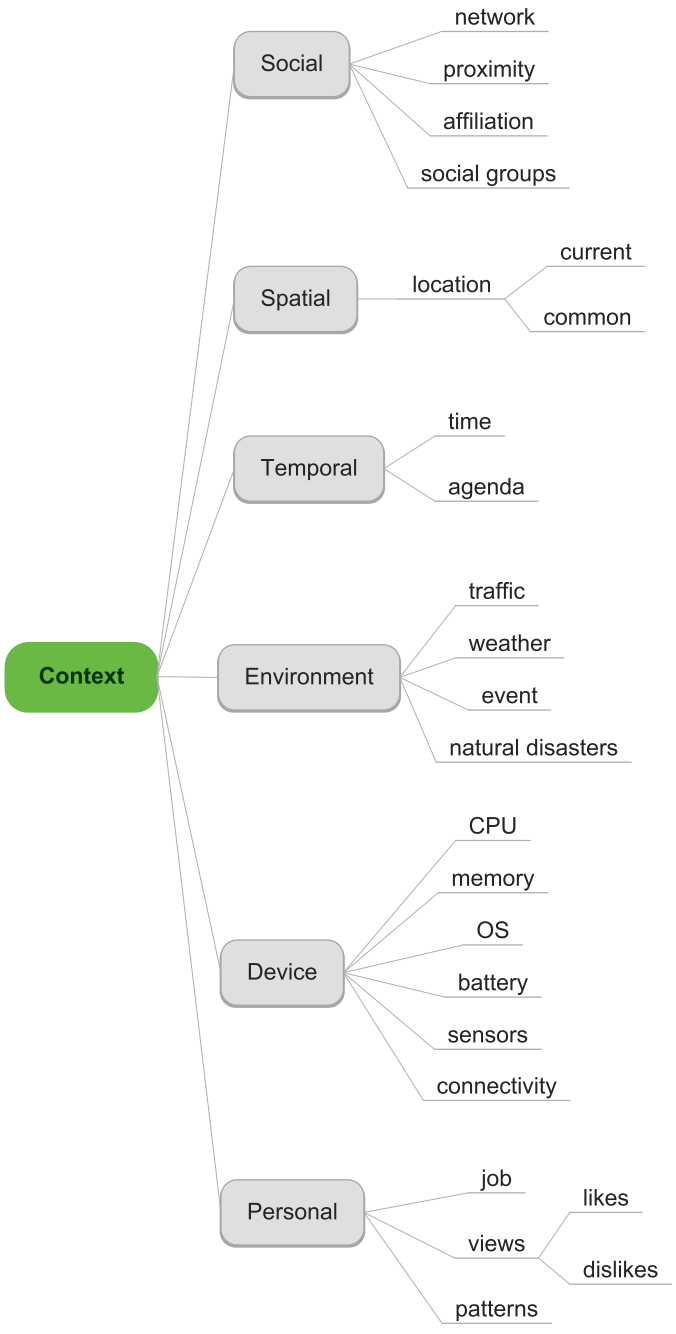


Figure 5.3: Few potential context attributes

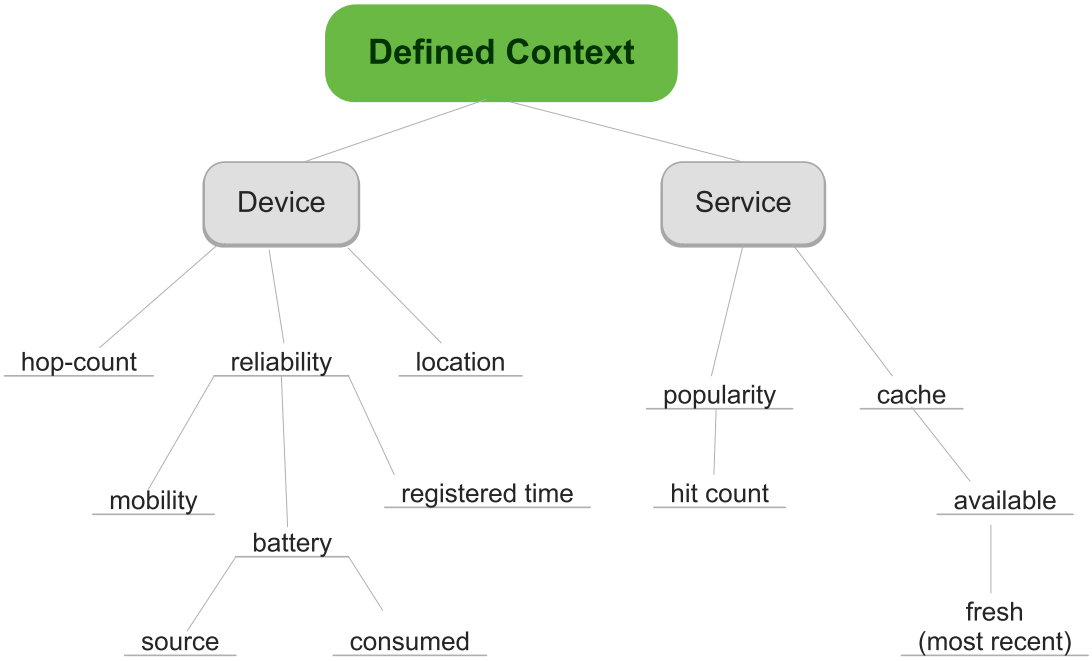


Figure 5.4: Defined context attributes

5.6 Registration and Status maintenance

This section focuses on the challenges related to registration and status maintenance approaches for the selected architecture (Section 5.2). Each SA needs to register with the DA. At first, the SA sends its service information with optional context attributes, e.g., location, battery information, etc. as described in Figure 5.4. The DA responds with an acknowledgement carrying the size of its time window. Figure 5.5 shows the detail about the registration process. After successful registration, every SA informs the DA about its status at least once in a time window. Upon reception of every subsequent status update message, the DA marks the respective SA active in its registry. The registry is checked by the DA at the expiration of a time window, and the records of all inactive SAs are deleted.

5.6.1 Provision of the DA's IP

All SAs need to know the address of the DA to register their services. There are several methods to provide the DA's IP address. It can be either hard coded in a SA (a well known or anycast IP address), or distributed as DHCP parameter, if this protocol is in use. Furthermore, the DA can flood the network to announce its authority, or suitable neighbour discovery messages can be used for this purpose.

5.6.2 Status maintenance

The registry keeps soft states of the entries that require a status update from the host devices within a time window. These status messages ensure reliability of SD, as devices can die or move away from the network after registration. The size of the DA's time window is constant for all SAs that has a potential of causing congestion, when nodes try to send periodic status updates simultaneously in a network [57]. This issue is fixed by defining a rule for each SA to select a random interval between 50% and 100% of the DA's time window. Figure 5.5 presents a scenario where the above approach is being implemented by the SA.

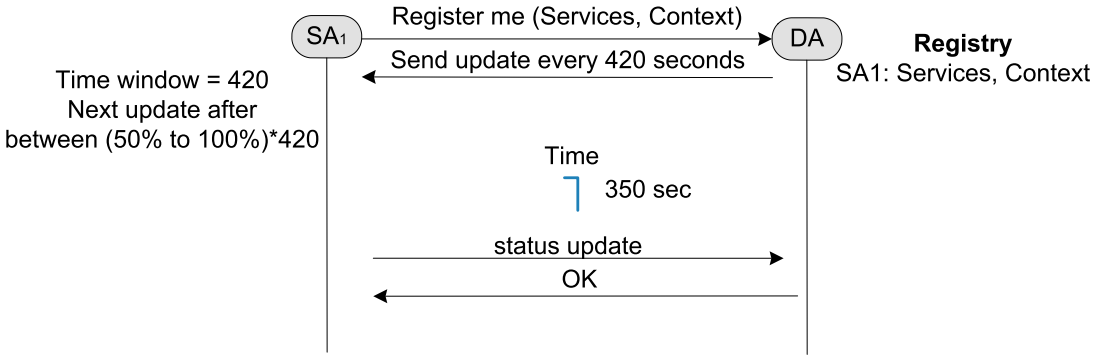


Figure 5.5: SA's registration and status updates

5.7 Service Discovery

The pro-active behaviour of the protocol allows the discovery process with a single unicast message. Figure 5.6 presents an example of a solution's scenario where the UA's query formulation and a glimpse of a SD process at the DA are shown. The UA can specify the context of a service to be discovered including its location, type or some other information (format described in Section 5.11.3). The location information is the most obvious and important criteria for a UA to target the search in a specific area. The DA discovers the matching services using information sent in the query. It searches its registry for services that match query's criteria. Subsequently, the DA responds back to the UA by appending the service information (resource's URL and IP address of the host) of one service or list of services in the payload.

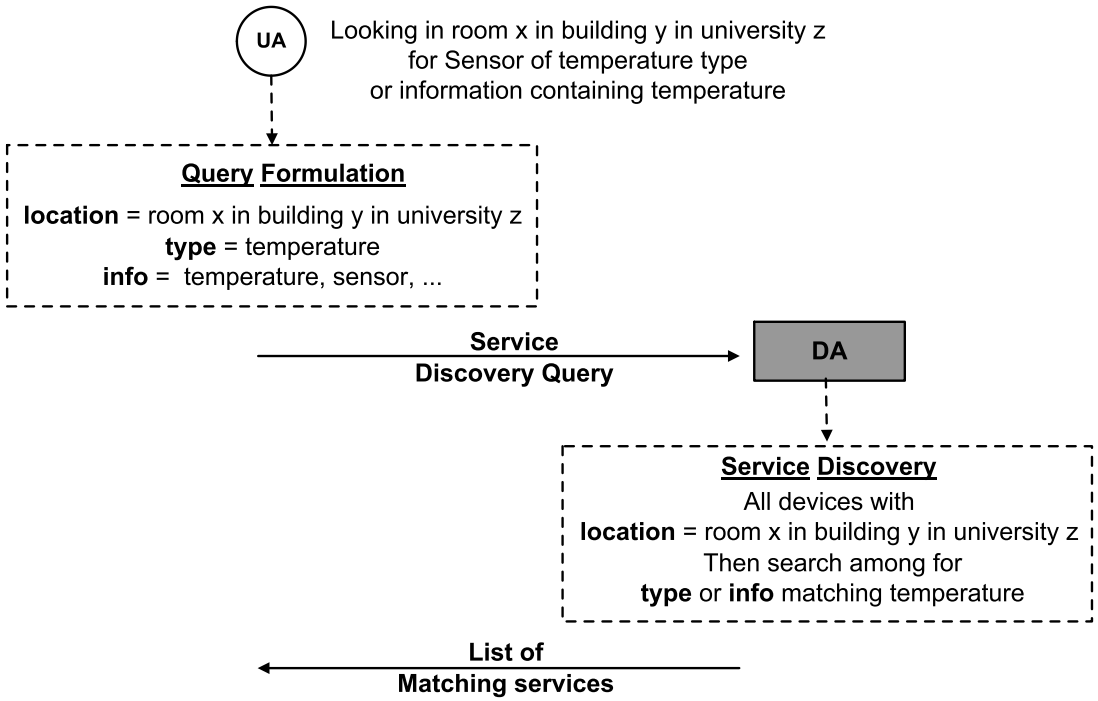


Figure 5.6: A SD Scenario using TRENDY

5.8 Service Selection

The development of a SDP that allows user applications to discover and interact with most appropriate services is crucial for IoT environments [126]. These services are hosted and advertised by heterogeneous devices and software components. An IoT user application will require the assistance in the selection of a best matched service, instead of getting a list of matching services. An example scenario is depicted in Figure 5.7, where the DA has discovered two matched services for the query. In this situation, the DA needs to assist the UA by selecting an appropriate service using available context information.

Figure 5.8 presents an algorithm to show the utilisation of context information by the DA to select an more optimal service from the list. This selection approach applies different checks, on available context information, one by one with the defined priority. It stops if nodes differ for certain context attribute after sorting the list or will carry on with other checks. Either stopping after a successful check or all checks, the first element of the list is returned. This algorithm works even when some of the context attributes are not available, as those checks are skipped. This algorithm is extensible as new context attribute checks can be introduced at the DA.

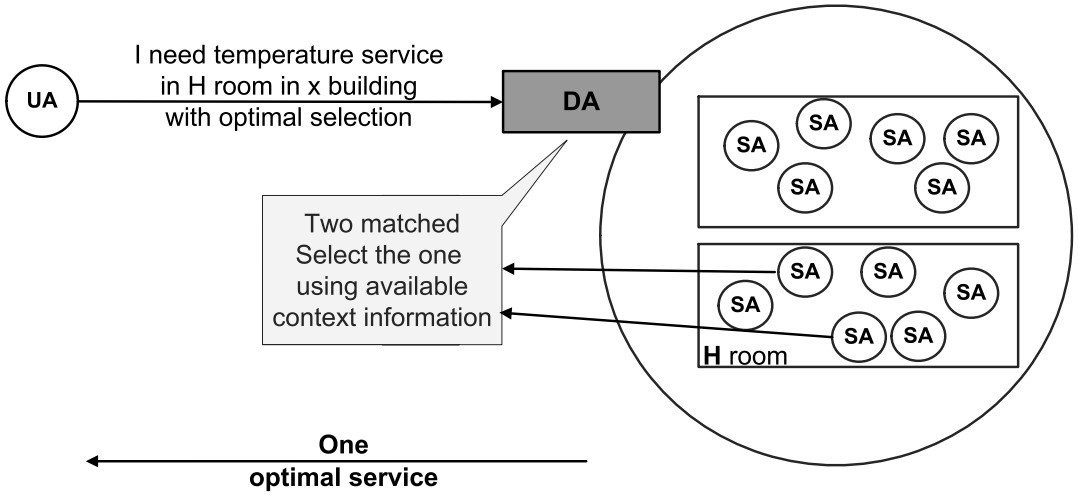


Figure 5.7: TRENDY's Service Selection Scenario

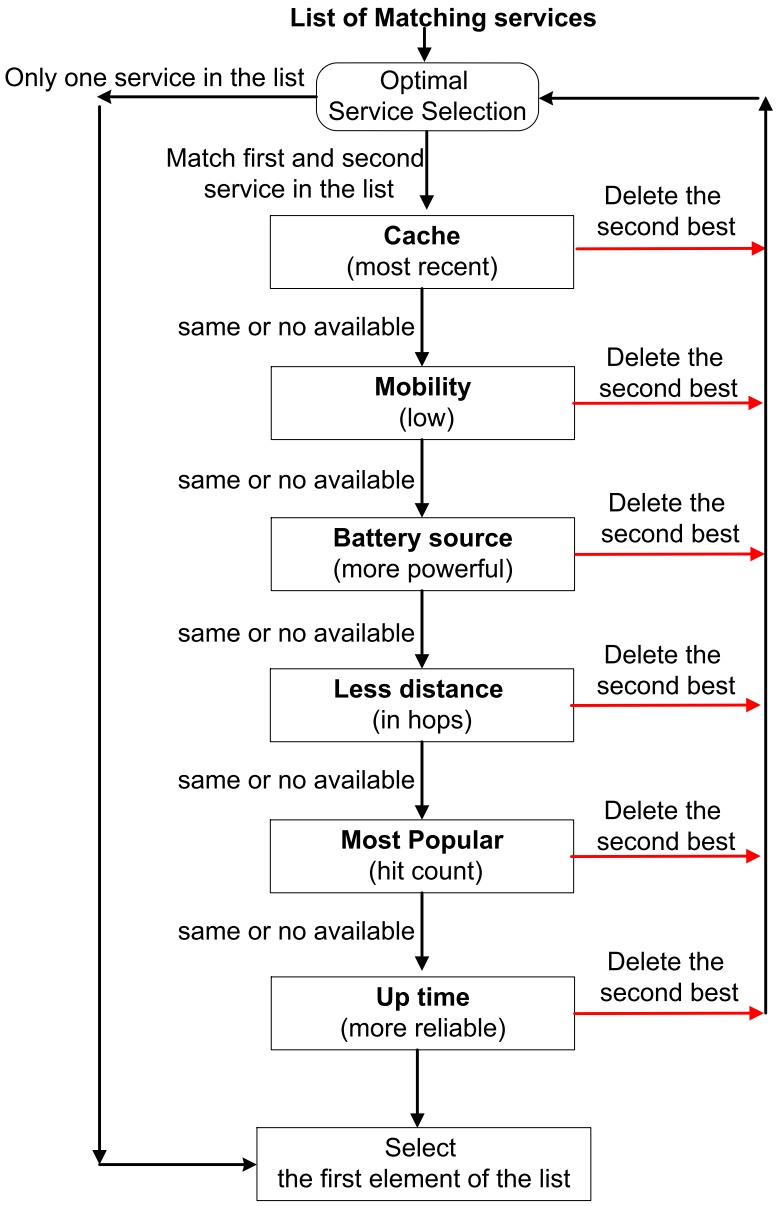


Figure 5.8: Example of TRENDY's Service Selection

5.9 Service Inovcation

SD is completed when an application gets the response with a service identifier and address of its host. In TRENDY, the RESTful web service paradigm is enabled by employing CoAP, so inherent support for service invocation is offered by standard web service interface. CoAP can be used to invoke services at any SA. Furthermore, HTTP can be used as well if a node supports it or a CoAP-to-HTTP proxy is available. TRENDY improves the response of a SA by defining a lifetime period for a resource's value depending on its previous reading. If a SA responds with a value of a resource, it will also append corresponding lifetime period, e.g., temperature sensor can help the SA to decide that life time for the value being sent is 30 minutes. The value is enclosed in parenthesis and lifetime in minutes is placed after that, e.g., a temperature of $22C$ approximately valid for next half an hour is sent as $\{22C\}30$. This enables the UA to get more information about the context and trend of the current value of a resource.

5.10 Overall framework

Figure 5.9 shows the TRENDY's framework with its features. The DA is a central repository that can cause single point of failure issue. Therefore, the DA's functionality is must be placed at a powerful always available node e.g., at edge-router in 6LoWPANs, which is a bridge between IP-based networks and 6LoWPANs.

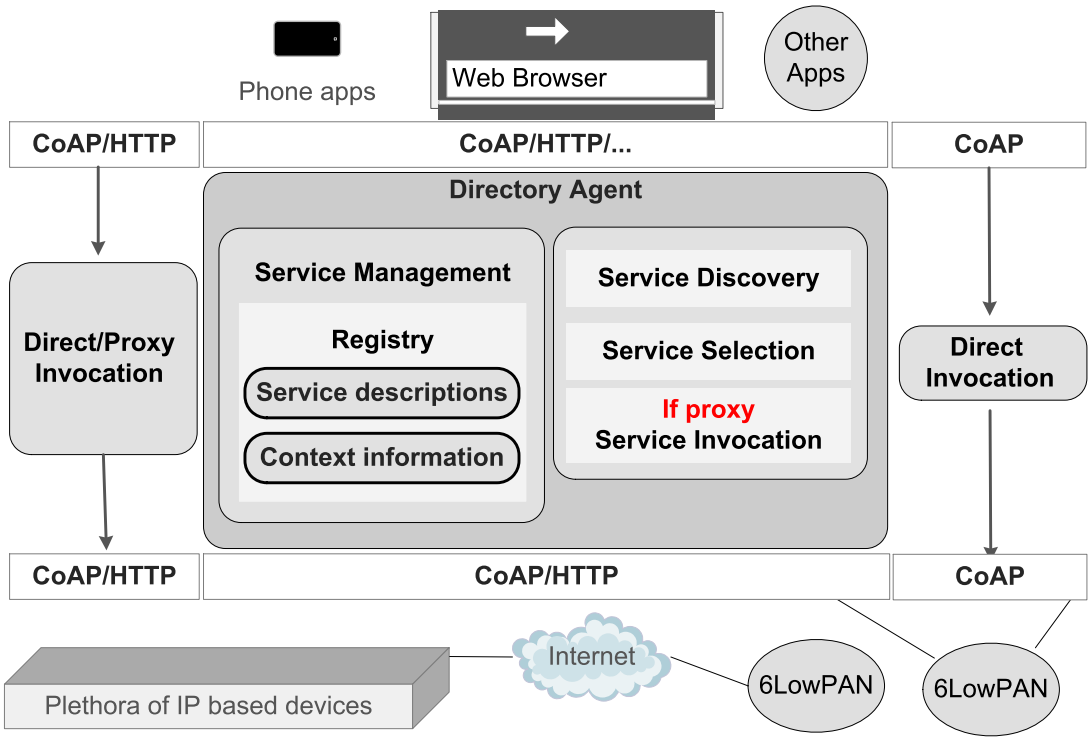


Figure 5.9: Framework of TRENDY with service selection

5.11 Message Formats

The CoAP message format (Section 2.5.4) is used to define messages for TRENDY. All messages are exchanged using unicast and require an Acknowledgement (ACK) message in response. This section covers the detail of TRENDY's messages required for registration, reporting and discovery.

5.11.1 UPD (Update) for registration

When a SA is switched on, it acquires its IP address and schedule time for registration with the DA. It is assumed that the address of the DA is known (Section 5.6.1). Then, it uses UPD message to register its service information with the DA.

Sender: SA

Receiver: DA

Purpose: SA registers its hosted resources at the DA

Format: The format of Update (UPD) messages for a SA is shown in the Figure 5.10.

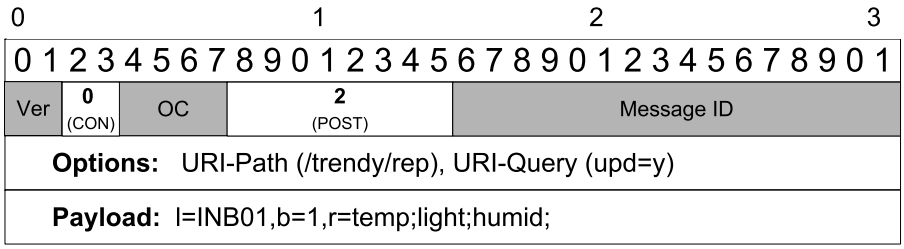


Figure 5.10: UPD message for registration

This message uses the method 2 (POST) to update the registry of DA. The option Uri-Path is set to `trendy/rep` and URI-QUERY is set to `upd=y`.

The minimum information contained in a registration message is a basic service description. Context information can be added, if necessary. The format consists of attribute and value pair (only a subset is defined) is separated by "=" sign whereas "," is used to separate multiple of them. The pairs can be specified in any order. The description of attributes in the payload are defined as follows:

- **Location tag ("l" or "location") = Value (string):** It indicates the physical location information of a SA in a descriptive or encoded format. If this is not given, the registry will register the SA with "Not Specified" location.
- **Battery source ("bs" or "battery source") = Value (8-bit unsigned integer):** This attribute explains that a SA is battery operated or is plugged in directly. If not defined it is set to '0' which implicitly defines that the SA is battery powered. Otherwise, the value '1' is used by a SA to describe it's plugged in.
- **Mobility level ("m" or "mobility") = Value (8-bit unsigned integer):** If specified it indicates that node is mobile. Furthermore, the value ranges from 1 to 9 which specifies the level of mobility. This is used while comparing two services according to their hosts mobility level. If not defined, default value of '0' is assumed.
- **Hop count ("h" or "hops") = Value (8-bit unsigned integer):** This value is used to specify the host's distance in hops from the sink (registry). If not specified then it is set to '0'.
- **Battery consumed ("b" or "battery") = Value (16-bit unsigned integer):** It represents the percentage of available battery consumed by a SA. It is estimated by a SA according to its battery usage. If it is not specified, then '0' is assumed by the registry.

Actions: The DA responds back with an ACK, and carries a two byte value in the payload specifying the number of seconds in a time window. A sample response to registration is shown in Figure 5.11, which carries 600 seconds in payload. The response code 2.01 (equivalent to HTTP 201) is sent back on successful registration to inform about the creation of a new record in the registry. In case of any problem, a corresponding error code is sent in the response.

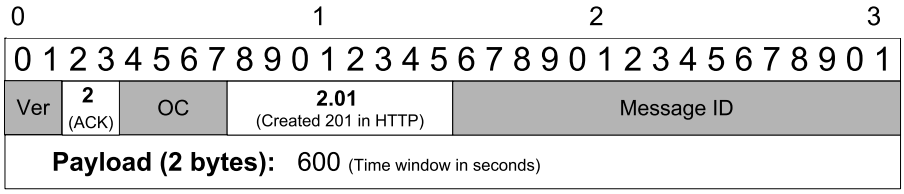


Figure 5.11: The DA's response for a registration request

5.11.2 UPD (Update) message for Status maintenance

This is a variation of registration message (Section 5.11.1) and used by SAs to update their status at the DA.

Sender: SA

Receiver: DA

Purpose: A SA updating the DA about current status.

Format: The format of UPD messages for status maintenance is shown in the Figure 5.12.

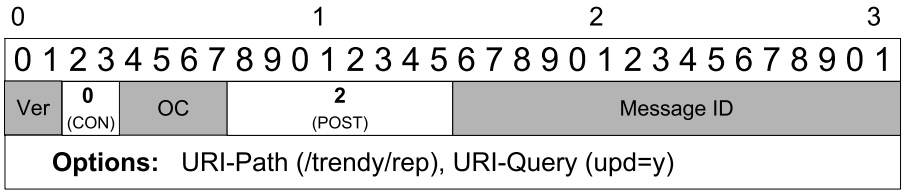


Figure 5.12: UPD message for a status update

This message uses POST method, Uri-Path option is set to `trendy/rep` and URI-QUERY is set to `upd=y`. If a SA also appends its service description or context attributes, then the DA will overwrite all its previously advertised information with the newly received one, which can be used by a SA whenever it detects change in any of its hosted services.

Actions: Figure 5.11 shows the format of an ACK. In the case of a successful update, a response code 2.04 (equivalent to HTTP 204) is sent back. In case of a failure, the DA will send a reset message with corresponding error code, as shown in Figure 5.13. After receiving such a reset message the SA will send another UPD message for registration of its services at the DA.

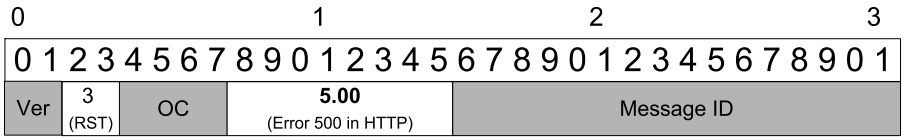


Figure 5.13: Reset response from the DA

5.11.3 SDR (Service Discovery Request)

This section covers the detail of Service Discovery Request (SDR) message which is used by a UA to discover a service.

Sender: UA

Receiver: DA

Purpose: Services are maintained at the DA to enable a unicast discovery. The DA accepts SDR messages in both CoAP (default) and HTTP protocol. The UA uses URL query options to explain context of a service or multiple services to discover. Services are searched based on attached query attributes and responded accordingly.

Format: A UA specifies the `trendy/server` URL with CoAP's GET method and URL queries (examples shown in Table 5.1) for SD. The discovery query can be specified in one of the following three different ways:

1. **Attribute-based:** The UA can define type, info (information) and location attributes using URI-QUERY to describe a query. Any or all attributes can be used to discover a service. The format of this type is shown in Figure 5.14.
2. **Best search based:** Any attribute based discovery message can be amended by adding `best` in the payload to seek assistance from the DA for appropriate service selection.
3. **Other format request:** The UA can request for the WSDL or WADL file of service information, by mentioning a URI-QUERY option with the format name. The message format is shown in Figure 5.15.

Table 5.1: TRENDY's discovery queries

Discovery queries	Matching Criteria
`?location=INB01`	Location based
`?location=INB01&type=temperature`	Location and type based
`?location=INB01&type=temperature&info=sensor`	Location, type and relevant information based

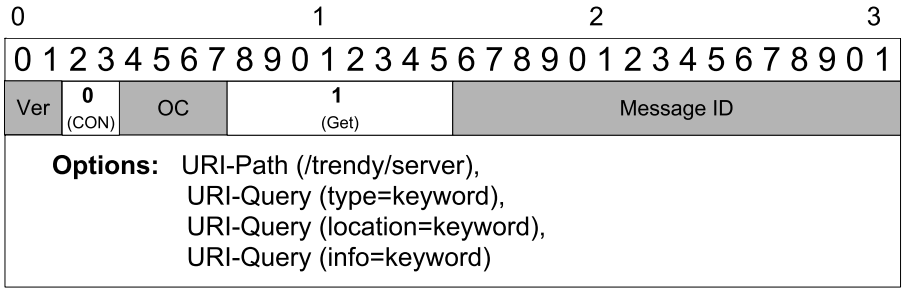


Figure 5.14: Service Discovery request (SDR) for services

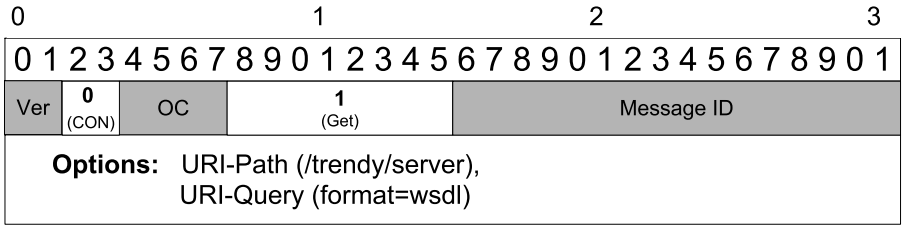


Figure 5.15: Service Discovery request (SDR) for format

Actions: The DA responds to a SDR query, depending upon its type.

1. **Keywords based search:** Matching service(s) with their URL(s) and SA's IP address(es) is returned in the payload.
2. **Best search:** The DA sends service information of the most appropriate service matched with its URL and host's IP address.
3. **Requested format file:** A HTTP link to the format file in the payload.

The general format of a response to SDR message is shown in figure 5.16. In case of success, the response code 2.05 (equivalent to 205 in HTTP) is used or, respective error code is sent to the UA.

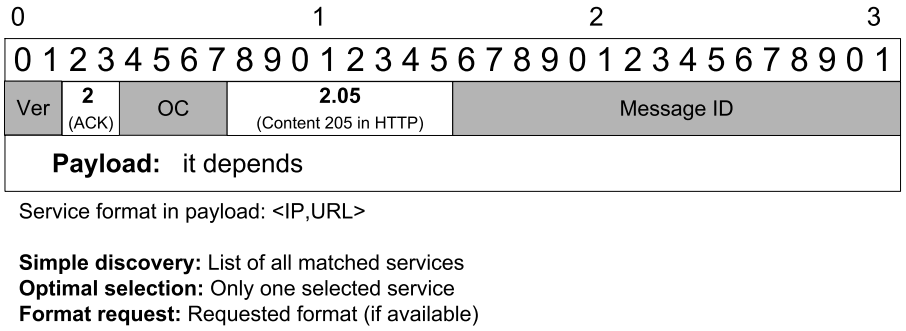


Figure 5.16: SREP: response carried in payload as requested

5.12 Summary and Discussion

This chapter has described the proposed TRENDY protocol that provides a context aware SD and selection for IoT environments. This protocol focused on constrained environments (more specifically 6LoWPANs) and consider the challenges and requirements identified in Chapters 3 and 4. The discussion started with the TRENDY's architecture and then different aspects of protocol are covered including, communication protocol, service description, context-awareness, registration, status maintenance, SD, selection and invocation. In the end, its overall framework is described and message formats are specified.

The solution uses light weight state of the art web services to enable RESTful paradigm. This enables the host devices to wrap up their offered services in web services with standard interoperable format. The design is focused on interoperability and extensibility, which are the main requirement for IoT environment. All specifications including, service descriptions, context attributes, SD, service invocation are defined while considering the future needs and extensibility. Overall,

TRENDY offers a SD solution that satisfies new requirements of the IoT and more specifically deals with the challenges posed by its constrained domains.

The next chapter explains the experimental design with the detail of selected tools for experiments. Afterwards, Chapters 7, 8, 9 describe additional techniques for TRENDY including, adaptive timer, context-aware grouping and adaptive caching.

Chapter 6

Experimental Design and Performance Metric

6.1 Introduction

This chapter starts by discussing the important performance metrics for SD from the context of the IoT. Then, the details of experimental set-up used is defined by describing configurations, including protocols used at different layers of IP stack, and employed physical topologies with generated RPL topologies. Furthermore, the experimental design is explained, which covers testing, debugging, validation, compilation of statistics, and graph generation. In the last part, the chapter presents the generated results from experiments performed to analyse the benefit of the service selection mechanism of TRENDY.

6.2 Performance Metrics

This section describes SD performance metrics, which are used to analyse the performance of the proposed solution.

6.2.1 Control packet overhead

A protocol's control packet overhead is defined by the number of control packets used by a protocol to serve its job. These packets are generated by a protocol to complete its operational and management tasks; therefore, it's a burden on the network. The control overhead affects the energy and bandwidth consumption of a network, and thus impacts the scalability factor of a protocol. In a dense network, the traffic generated by SD becomes an extra burden, because it has a considerable effect in a multi-hop network. The frequent packet forwarding reduces the sleep time of the nodes by keeping them awake for passing messages, so this consumes

more energy. Therefore, an efficient and scalable SD solution generates less control traffic.

In the case of TRENDY, an application layer protocol, the control overhead is the sum of all registration, and reporting messages. The number of service invocation messages are technically not considered as a part of overhead; however, if a SDP employs a caching mechanism, then this number becomes important to analyse the impact of the scheme.

TRENDY works at application layer and is independent of lower layer protocols that's why only application-layer level packets are counted to calculate its overhead. However, in reality, the application layer messages initiate a round of several efforts at the bottom layers, and overhead actually costs several times more than the count at the top layer. This effect becomes much larger for 6LoWPANs with lossy radio environments consist of nodes with sleeping cycles, as one packet might be sent several times before it reaches at the destination. Therefore, the total network packets are also recorded in experiments to quantify the impact of such properties. Subsequently, further analysis is done on recorded packets using Wireshark [98] to segregate and classify SD messages from routing and other messages.

6.2.2 Service Discovery Delay

The service discovery delay is the interval between initiation of a UA query message and receipt of a response from the DA. It includes the processing time at the DA to match and select a service with query parameters. TRENDY has a registry-based architecture and has proactive behaviour, so this factor does not have much significance because only a single uni-cast message required between two entities (UA and DA) for SD. However, it is an important SD metric that matters more in directory-less and distributed directory based solutions where a query has to pass through different nodes either by multicast or unicast in case of multiple local directories.

6.2.3 Service Invocation Delay and cache hits

Service Invocation (SI) delay is defined by the time taken from issuing an invocation request until the receipt of a response. This is an important parameter to analyse the efficiency of a SD solution, because lower SI delay means better system performance and response to a user application. The network traffic load, average path length in multi-hop network and message processing time are the factors that affect the SI delay. Usually, a service discovery solution has no impact on this metric, unless it is either supporting service selection, service invocation or using some caching mechanism to facilitate user applications. If caching is enabled, then

the number of cache hits defines the number of times the maintained cache is used by the registry to serve a request. This number measures the benefit of maintaining the cached values. TRENDY's employed communication protocol; CoAP, enables service invocation using RESTful web services. Thus, service invocation delay becomes an important metric to be measured in TRENDY experiments.

6.2.4 Energy Efficiency and Network lifetime

Energy is considered a precious resource in 6LoWPANs with mostly battery-operated nodes. A node consumes energy while processing, sleeping, and radio listening, receiving and transmitting. Energy consumption is also highly dependent on the routing overhead, and selected MAC and RDC mechanisms [6].

In experiments, Energest [33] is used to evaluate the energy consumption of individual nodes. Energest is a software-based mechanism to estimate the energy consumption of a sensor node, which runs directly on a node to provide real-time estimates of current energy consumption. It keeps track of the time spent by individual modules, including, CPU, radio transceiver (listen, receiving and transmitting), and low power or sleep mode. The time is then processed using the data-sheet of emulated devices to calculate the energy consumption. COOJA performs the simulations at the hardware-level by using the real hardware profiles of the employed nodes; therefore, devices can be emulated with actual hardware configurations. Tmote Sky motes are emulated in all experiments as 6LoWPAN nodes in COOJA. Table 6.1 shows the data-sheet values of current consumed by a Tmote Sky while performing different operations. These values along with CONTIKI's fixed operating voltage of `3V` are used in experiments to calculate the total energy consumption of each node. Different statistics, including, individual node energy consumption and average node energy consumption are used to compare the energy efficiency of proposed techniques in the performed experiments.

Table 6.1: Current consumed by a Tmote Sky while performing different operations

Operation	**Current** (`mAmp`)
`Radio listening` and `Receiving`	`19.7`
`Radio transmit`	`17.4`
`CPU ON`	`1.95`
`CPU LPM` (Low Power Mode)	`0.5`

Energy consumption and network lifetime are interrelated as lower energy consumption suggests that network will last longer. The definition of the network

lifetime varies according to a specific application or topology [20]. In TRENDY, location tags are considered as a key criterion to search and differentiate services according to their placement within a network. Therefore, the network is segregated into different locations with a dynamic topology with the RPL [133] protocol. RPL maintains its topology by forming new routes dynamically during the simulation period. Consequently, the different topologies are used in experiments to check the impact of the diversity of generated RPL routing trees. In reality, the definition of the network lifetime depends upon the way nodes are placed in a network. This research project considers the most widely used definition of the network lifetime where the network lasts until first node dies in a network.

Emulated Tmote Sky motes can be powered by two AA batteries with operating voltage range between $2.1V$ to $3.6V$. Normally, two Alkaline batteries provide on average capacity of `2500mAh` with operating voltage of 3 volts. Following formula is used to compute the capacity of batteries in Joules.

$$E_{Capacity} = V \times I \times 3600$$

Where $E_{Capacity}$ represents the available energy in Joules, V is the electromotive force expressed in volts and I is the current expressed in amperes. The equation $V \times I$ is multiplied by 3600 to get the Watt-seconds. Therefore, the total available energy capacity, by applying the formula is:

$$E_{Capacity} = 2.5 \times 3 \times 3600$$

$$E_{Capacity} = 27000 \text{ Joules}$$

The individual energy consumption of nodes during a simulation period is used to linearly approximate the energy required by them in a day, which is used further to approximate the total lifetime of nodes for available energy capacity provided by two AA batteries. The assumption taken here is that all batteries will provide `2500mAh` capacity while their voltage will never drop from 3 Volts. Finally, these estimations are used to determine the time when first node dies.

The assumption taken in energy consumption approximations is that the same events occurred during the simulation period (number of UA queries and control overhead) will keep on repeating in the similar way over the lifetime of the network. However, the nodes will not get registered again neither the equivalent number of queries will be repeated in reality. For example, in the discussed scenarios (Section 4.2), the network is supposed to have longer idle periods of low demand during the nights. Therefore, this calculation is only an assumption and represents an approximation of the network lifetime for a rather worst-case scenario.

6.2.5 Scalability factor: Packets to the DA

In a centralised SD solution, all services are stored and maintained by a DA. This requires messages for registration and status maintenance to be sent to the registry by all host devices. Consequently, the number of messages generated by nodes for the DA can overwhelm a network, as mostly these messages need to pass through multiple hops to reach at the DA (sink). Thus, the total number of these messages determines the scalability factor of a SD solution.

TRENDY's DA counts the number of packets received from all SAs as a measure to determine the scalability of the solution. Each TRENDY message consists of a confirmation and an acknowledgement, so the number of received messages at the DA actually results in two packets; however, the messages are just counted once when they arrived at the DA. In a 6LoWPAN with mostly sleeping nodes, which usually more than one hop away from the DA, the anticipated generated traffic by these application-level messages is far higher. To study this impact in different network densities and RPL routing configurations, the network is sniffed for the total number of packets sent and received by the DA to analyse the overall impact of a mechanism to reduce the actual all-inclusive traffic.

6.2.6 Reliability and Accuracy

The reliability covers different aspects: the reliable message delivery in a lossy wireless environment, and the prevention of single point of failure in the case of a centralised directory. Whereas, accuracy is determined by the correctness of the information provided by a SDP in response to the UA queries. Following is the brief detail of different factors which determine the reliability and accuracy of a SDP:

Reliable communication: Constrained networks operate with a low bandwidth, small packet size, and lossy asynchronous links (Section 2.2.2). The challenge for a SDP is to offer reliability in these constrained environments. TRENDY addresses the reliability by using CoAP based communication that offers reliability using back-off mechanism and acknowledgements (Section 2.5.5). Moreover, TRENDY's messages can easily fit into a single 6LoWPAN packet, which alleviates the need of packet fragmentation.

Reliable directories: A SDP solution with centralised architectures can cause a single point of failure problem, where a solution stops functioning after its central directory fails. A SDP solution with central directory needs to provide some way to ensure reliability. TRENDY deals with this problem

by implementing a centralised directory at the resource-rich edge-router (Section 5.10).

Accurate Service Information: The accuracy of a SD solution determined by the accuracy of discovered service information and cached values in the case of caching. In a SD solution, service information is advertised by nodes either among neighbour nodes (directory-less approaches) or to directories (directory-based architecture). This advertised information is cached at different nodes to reduce service discovery delay, as other nodes can also serve queries. This cached service information maintained at other nodes is needed to be refreshed by status maintenance to guarantee the accuracy of a SD response.

In TRENDY, the DA stores the soft values of service entries, which require status updates from the nodes in a specified time period (Section 5.6). If no status update is received from a SA, the corresponding records are considered obsolete and removed from the DA's registry. This ensures the accuracy of SD responses from the DA. Furthermore, the UA sends a service invocation query after receiving a SD response, so a successful service invocation certifies the accuracy of the service information.

Accurate cached values: If a caching mechanism is employed, there is a risk of expired cached values being sent to a UA by intermediate proxy. Therefore, this issue of outdated cached values need a consideration in such cases.

6.3 Experimental Setup

This project employed the CONTIKI operating system with RPL as a routing protocol and used COOJA to simulate a 6LoWPAN network. COOJA allows hardware-level simulations, so it can emulate different hardware nodes with respective drivers. Therefore, all the SAs in the 6LoWPAN are emulated as Tmote Sky [115] motes, so the same implemented code can work directly on real hardware. ContikiMAC [32] is used as the RDC scheme and compared to NullRDC (where radio never switches off), and CSMA is used as MAC protocol. Two implementations of CoAP are used in experiments. Erbium [70] is a CONTIKI based CoAP implementation, which is used inside the 6LoWPAN for SAs; and Java based Californium[1] is used to implement the DA and UA.

All simulations consist of a different number of nodes where one node acts as a border router to connect the COOJA-based 6LoWPAN to the DA running as Linux

[1] http://people.inf.ethz.ch/mkovatsc/californium.php

process via the Serial Line Internet Protocol (SLIP). All nodes are placed according to one of the topologies (explained in Section 6.3.1) and given 5 unique location tags. This is done to send queries in different areas of a network and to analyse the effect of context-awareness of TRENDY. Each node hosts three resources: temperature, humidity and light. Furthermore, this has been assured that a SA can register its all resources in one 6LoWPAN packet. Thus, the TRENDY message and packet become synonymous and are used interchangeably. CONTIKI's ENERGEST [33] module is used to measure the energy consumed at each node. All UA queries are stateless (each is sent as if from a new UA) and randomly selected to discover a resource at the DA from one of the five locations after a random interval between 0 and 10 seconds. After receiving a response for a SD query, the UA sends a *GET* request to invoke the service at the SA. Consequently, the number of Service Invocation requests are equal to SD queries. All the experiments are repeated with 10 different random seed values, and overall traffic is sniffed for validation in one iteration to analyse the actual number of packets produced in a network. Furthermore, the analysis classifies and segregates different types of packets to see the real burden of SD and the impact of different topologies.

The simulation configurations are summarised in Table 6.2.

Table 6.2: General configurations for experiments

Total nodes	Variable + 1 border router
Number of locations	5
Node Type	Tmote Sky
Routing protocol	RPL
Radio duty cycling	ContikiMAC and/or NullRDC
MAC	CSMA
Energy Measurement	ENERGEST (CONTIKI)
Radio Environment	Unit Disk Graph Medium (UDGM)
Transmission range	TX: 50m, Interference: 50m
Physical topology	Section 6.3.1
Area	Depends on the selected topology
Number of cases	Variable
Number of Iterations	10
Total Service Discovery queries	Variable
Total Service Invocation queries	Variable
First UA Query	Variable
Interval between queries	Random between 0 and 10 seconds
DA time window period	Variable
Number of time windows	Variable
Simulation Duration	Variable

6.3.1 Topology

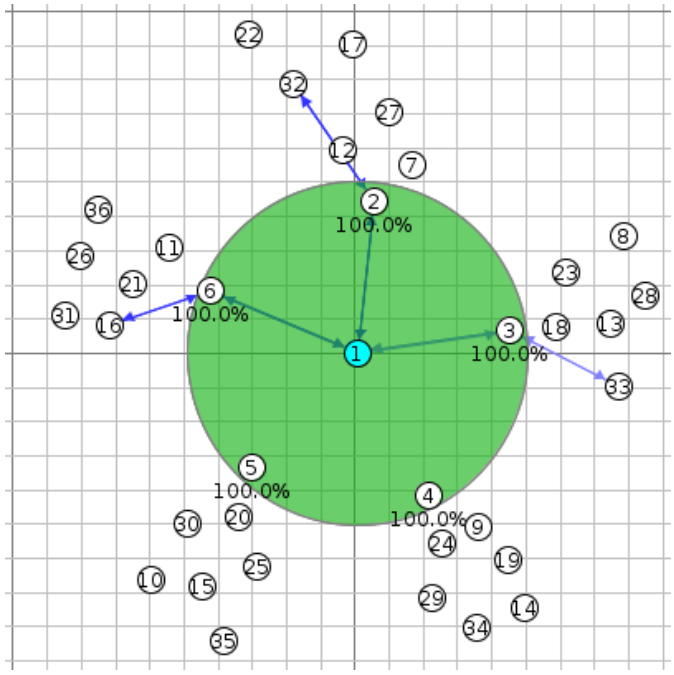
The experiments are done using three different tree topologies. In these topologies, nodes are placed from completely random to a fully controlled way to analyse the effect of RPL's generated tree topologies when different TRENDY mechanisms are used. The placement of a node at a certain level of RPL tree is instrumental in its energy consumption. In experiments, each iteration uses a different random seed value for a COOJA simulation. This value is responsible for the packet order, node start-up time that changes the generation of interrupts. Therefore, the RPL tree construction is also affected by this value and results in a slightly unique tree for each seed values. Following is the description of employed topologies:

Topology#1 - Fully Controlled topology: The RPL topology construction mechanism (Section 2.3.1) can be controlled by using different RPL's OFs and ranks for a DODAG. This topology fully exploits this fact and places the nodes in different areas while strategically placing only one node in each group one-hop away from the DODAG's root (where the DA is also placed). However, other nodes in each group are placed randomly, but in a way that no node from any group is in the radio range of another group's node. The whole topology covers an area of $190m \times 180m$. The nodes organisation in distant groups is analogous to discussed scenarios in Section 4.2 where nodes can be placed in different building and one resource-rich node that has large transmission range in each building becomes the parent (in RPL tree) of all local nodes. The DODAG's root (border router) acts as a sink. Figure 6.1a shows the physical topology of 6LoWPAN defined in COOJA simulator. Furthermore, Figure 6.1b presents the RPL tree generated for one COOJA simulation iteration, which is, in reality, is incongruent to physical topology; however, it matches closely to the physical topology for this topology because of its strategic configurations.

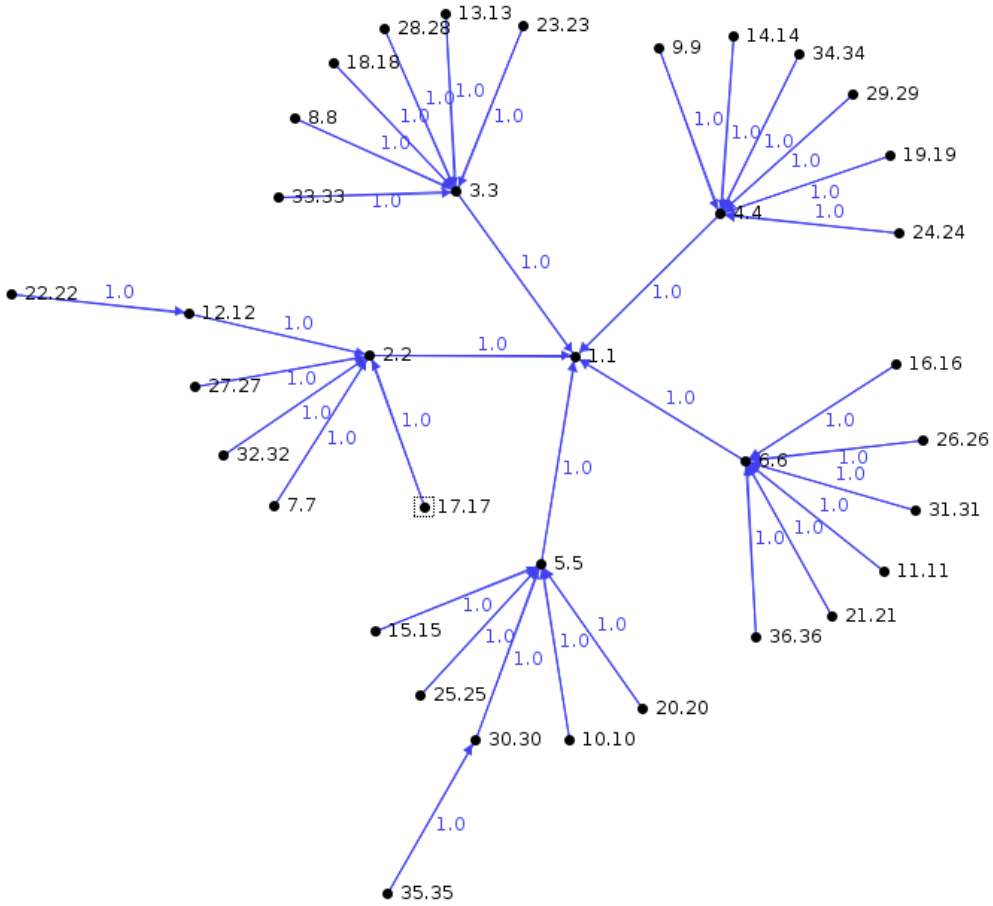
Topology#2 - Partially Controlled topology: This topology is similar to topology 1, but in each area, nodes are allowed to have overlapping radio access to nodes in other areas. Therefore, the topology becomes more dense and occupies an area of $180m \times 140m$. This topology demonstrates a physical topology of nodes in a single building where areas represent different floors. Figure 6.2a shows the physical topology in COOJA and Figure 6.2b presents corresponding RPL routing topology for one random seed value. In contrast to topology No. 1, it can be noticed that the routing tree is not even for each area, and different parent nodes have a different number of branches. This is the result of RPL parent selection mechanism (described in Section 2.3.1)

that has preferred the parents from other areas and thus creating dissimilar routing paths compared to topology 1. This topology is a combination of routing tree tweaking (forced top parent nodes in the tree) and more common overlapping areas in the IoT.

Topology#3 - Completely random topology: In this topology, nodes are placed in a fully random order with overlapping radio ranges. It is more dense compared to other two topologies and covers an area of $150m \times 170m$, as shown in Figure 6.3a. Figure 6.3b shows the generated RPL tree, which reflects how RPL can behave when nodes are homogeneous and thus selects any node to become a parent of others. It can be notice that more nodes are situated at tree's level one and have a direct link to the DODAG root. Furthermore, only two nodes get more than three branches. This will cause more contentions towards the root, and will increases the delays that will multiply when some RDC is used by nodes for sleep cycles. This topology represents a real-life deployment where all nodes are homogeneous and are meant to work in a complete ad-hoc manner without any RPL configurations.

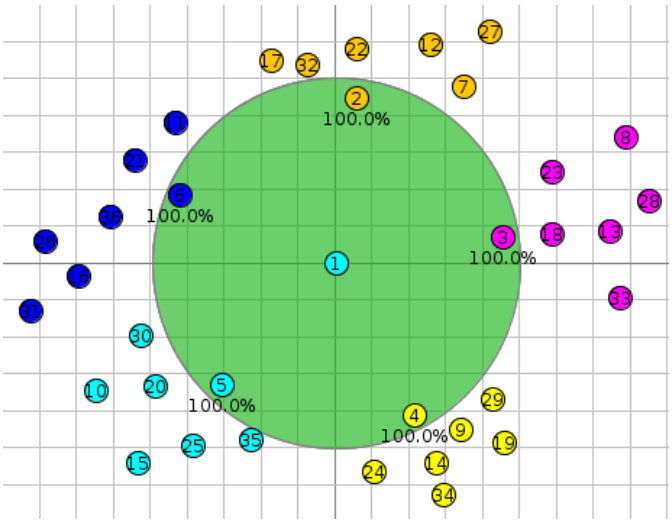


(a) Physical Topology No.1

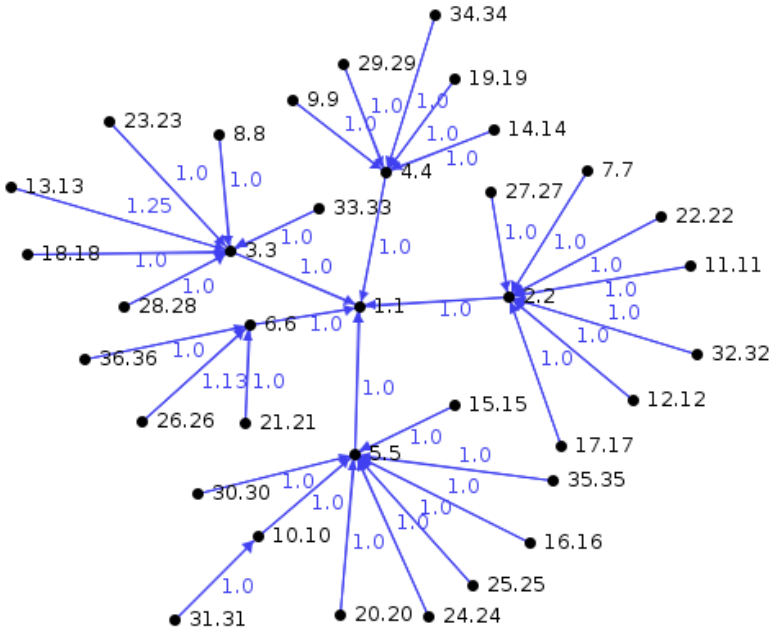


(b) RPL tree for Topology No.1

Figure 6.1: Different views of Topology 1

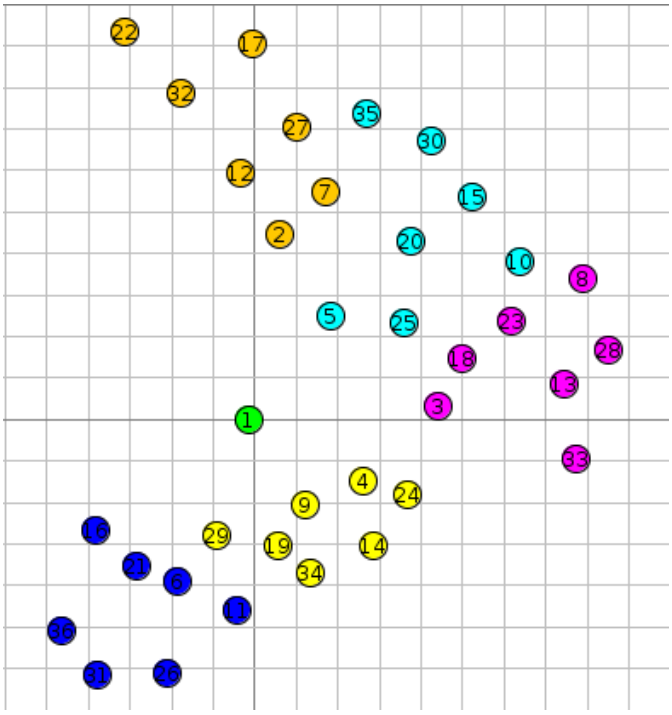


(a) Physical Topology No.2

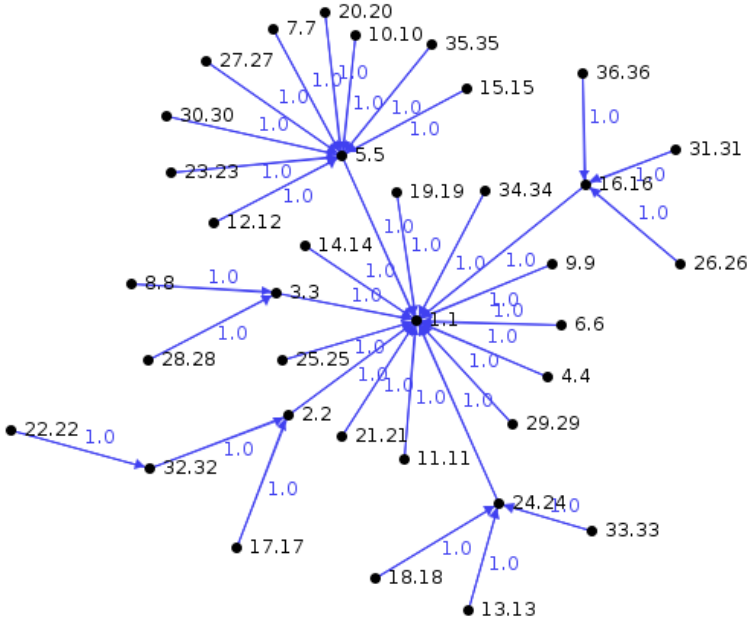


(b) RPL tree for Topology No.2

Figure 6.2: Different views of Topology 2



(a) Physical Topology No.3



(b) RPL tree for Topology No.3

Figure 6.3: Different views of Topology 3

6.4 Experimental Design

The DA and UA run as Java's processes on Linux machine and the 6LoWPAN network runs in a COOJA Simulator (which is also a Java process). The communication between IPv6 and 6LoWPAN is managed by using a Tunslip via an edge-router located in COOJA environment. The maximum number of 35 nodes are used in the experiments, because the simulation running in COOJA performs slower than real-time when emulated nodes interact with the other Java processes (DA and UA) as it uses only one CPU thread for execution. The clocks of the DA and UA (which normally run at real-time), and the COOJA simulation speeds were synchronised to run the experiments with 0.85 of real time. The simulation architecture is shown in Figure 6.4.

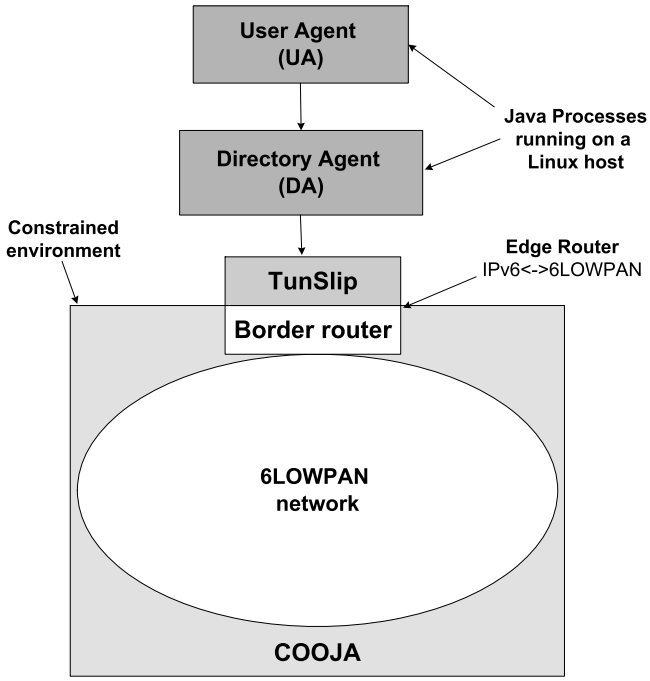


Figure 6.4: Experiment setup for evaluation of TRENDY

The design of performing experiments and data collection is shown in Figure 6.5. Eight VMs (Virtual Machines) are used in the shown way to execute simulations and synchronise the generated data using Dropbox web service[2]. All validation and statistic data are collected at a central machine, where it is synced via Dropbox. The received raw statistics are further processed by applying bash scripts to generate the suitable format for plotting graphs using gnuplot [58] scripts. The details of automation of simulations, COOJA's JavaScript, and data processing bash scripts, validation logs and statistics, and gnuplot scripts can be found in Appendices A, B and C.

[2]http://www.dropbox.com/

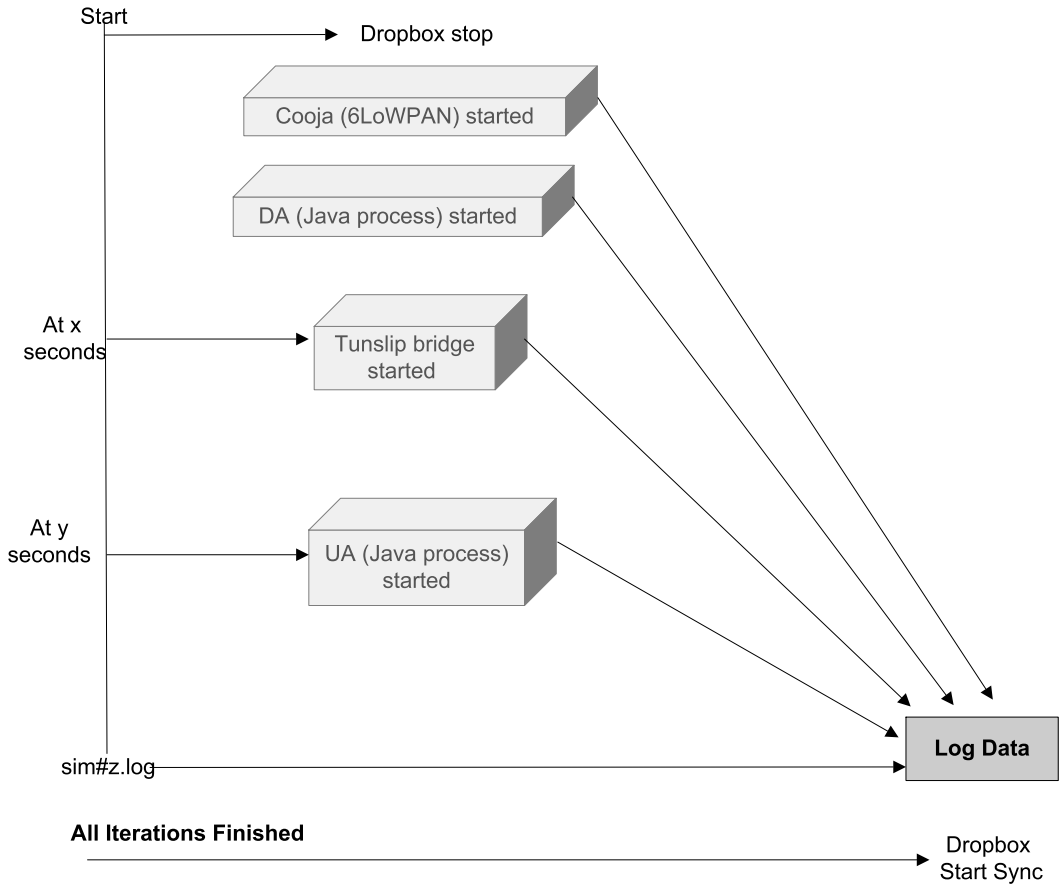


Figure 6.5: Experimental design: Simulation setup and data collection for experiments

6.5 TRENDY's Service Selection Experiments

This section covers the detail about experiments conducted to emphasise the role of TRENDY's service selection mechanism (Section 5.8), including scenarios, experimental setup and discussion of the results.

6.5.1 Introduction

Following scenarios are used to analyse the impact of service selection for a network of 35 Nodes:

Case 1 - Basic TRENDY: This scenario only enables the basic functionality of TRENDY, so the UA will get a list of all matching services.

Case 2 - Basic TRENDY with service selection: This scenario enables TRENDY's service selection mechanism on top of case 1. Thus, the UA gets a best matched resource's URL and IP address of a SA hosting the matching service.

The simulations are configured with the same settings defined in Section 6.3 with the changes given in Table 6.3.

Table 6.3: Configurations for the experiments of adaptive timer

`Total nodes`	35 nodes + 1 border router
`Physical topologies`	1, 2 and 3 (Section 6.3.1)
`Number of locations`	5 with 7 SAs in each location
`RDC`	Both ContikiMAC and nullRDC
`Number of cases`	2
`Total Service Discovery queries`	1000
`Total Service Invocation queries`	1000
`First UA Query`	At 600 seconds
`DA time window`	5 minutes = 300 seconds
`Number of time windows`	30
`Simulation Duration`	$300 \times 30 = 9000$ seconds

This section continues with the generated results of simulations to analyse the effect of TRENDY's service selection mechanism.

6.5.2 Control packet overhead

The service selection mechanism has not affected the application-level control packet overhead, because it has not introduced any extra packets nor is it's target to reduce the number of packets. However, Figure 6.6 shows that the service selection mechanism actually improves the total network traffic, because of the selection of

closer nodes in a multi-hop network. Furthermore, all the scenarios with NullRDC have performed similarly with the significantly low traffic compares to ContikiMAC cases, because it never switches off the radio, and thus it avoids re-transmissions. On the other hand, topology 3 has the highest number of packets below routing layer in ContikiMAC scenarios, because of its high density. In the simulations, RPL assumes the availability of IP stack for the provision of bi-directional links and Neighbour Un-reachability Detection (NUD). Consequently, more neighbour discovery packets are communicated between nodes in a dense network to update their neighbour table entries. Figure 6.7 supports this phenomenon by categorising the traffic below RPL, which explains that neighbour discovery traffic is much higher in topology 3 for ContikiMAC cases. Similar trend is quite evident in NullRDC cases, but the increment in the number of neighbour discovery messages is very small because of always available radio.

Figure 6.8 segregates the total CoAP traffic into the number of SD, and invocation packets. The benefit of using NullRDC is apparent, but it can also be noticed that the service selection mechanism has decreased the number of service invocation messages. Same fact is further elaborated in more detail in Figure 6.9 which classifies service invocation messages into the number of requests and responses. The packet analysis shows that the number of packets for service invocation requests and responses decreased almost equally. This effect is the result of the nearest node selection by the service selection mechanism that reduces the number of hops to reach the destination. Overall, ContikiMAC's retransmission is the reason of the high number of packets of its cases compared to NullRDC scenarios.

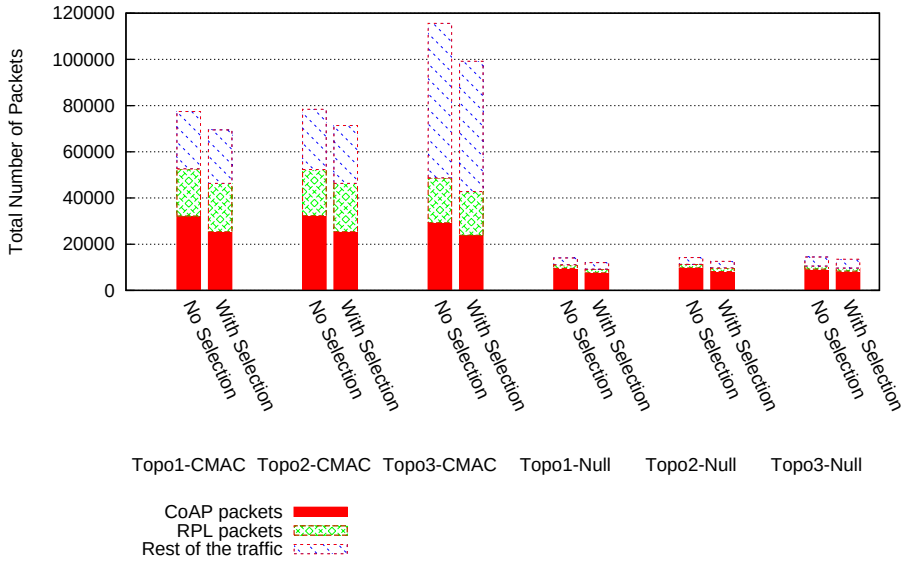


Figure 6.6: Total network traffic in ContikiMAC and NullRDC cases for topologies 1, 2 and 3

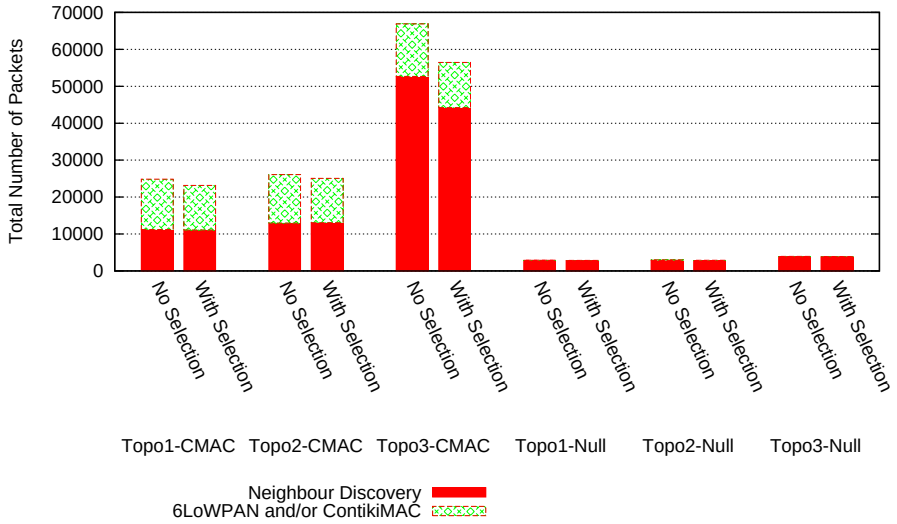


Figure 6.7: Details of traffic below routing layer in ContikiMAC and NullRDC cases for topologies 1, 2 and 3

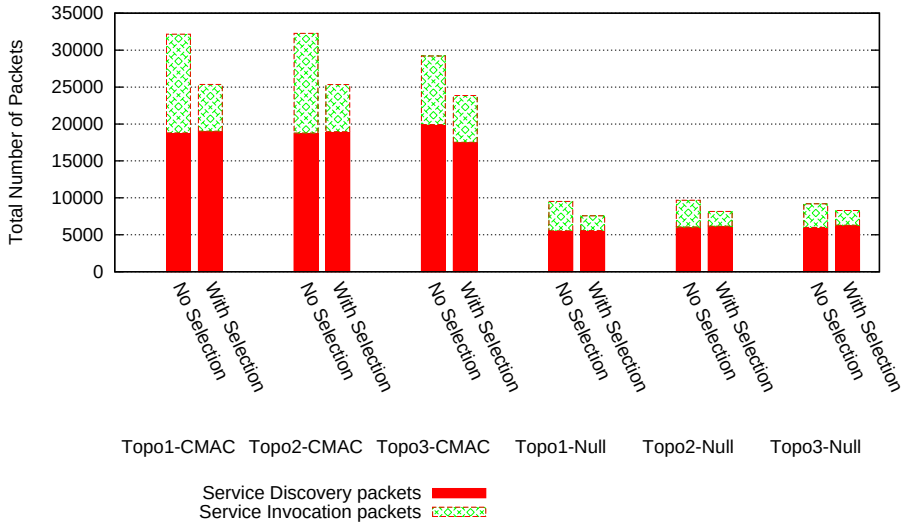


Figure 6.8: CoAP's traffic details in ContikiMAC and NullRDC cases for topologies 1, 2 and 3

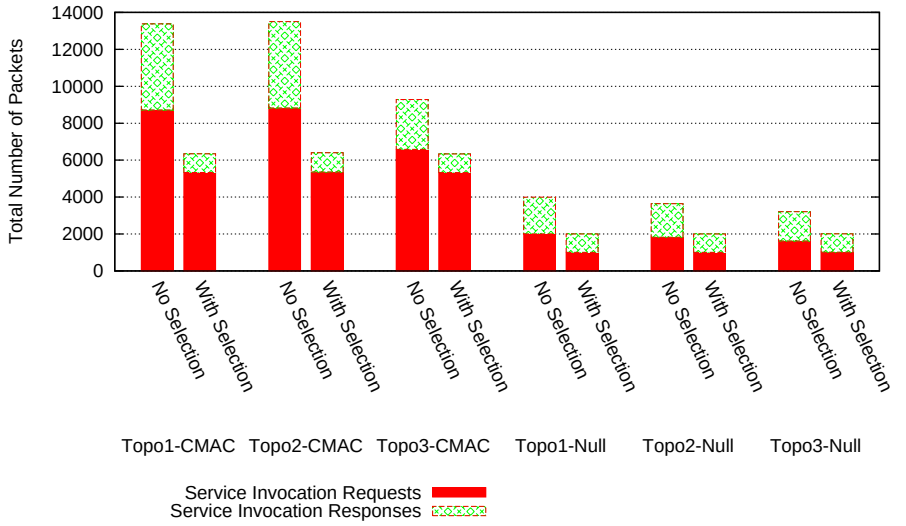


Figure 6.9: Service Invocation traffic details in ContikiMAC and NullRDC cases for topologies 1, 2 and 3

6.5.3 Service Invocation Delay

Figure 6.10 shows that the service selection mechanism has reduced the average service invocation delay in ContikiMAC scenarios for all topologies. Both configured topologies 1 and 2 have performed nearly similarly where delay reduced to almost half; however, topology 3 still incurred high invocation delay. The density of topology 3 has played a major role in increasing service invocation delay, because more nodes are at the first level of the RPL tree, which increases the time of response from the corresponding node because of the extra communication done at lower layers, as shown in Figure 6.7. The efficiency in service selection scenarios is the result of the context-awareness of TRENDY, as it selects the most appropriate service from all the matching ones (explained in Section 5.8).

On the other hand, the NullRDC cases don't show a significant improvement compared to ContikiMAC cases, because in these cases, the radio remains always on which decreases the query response time significantly, as no extra time is spent by the messages in waiting for radio to be switched on. However, the service selection mechanism has even then improved the service invocation delay and topology 3 gained the maximum efficiency compared to other topologies, because the benefit of better node selection becomes more visible in NullRDC. This fact is further elaborated by Figure 6.11 that shows the role of the service selection mechanism in ensuring the low delay for most of the UA queries. It can also be noticed that the employment of service selection improved the service invocation delay for almost 80% of the UA queries in topology 3, but still the overall high delay for 20% of queries affects the average service invocation delay. The analysis of Figure 6.12 shows that the high delay happens after few queries in all topology 3 scenarios, which is the effect of high density that increases the waiting time using both ContikiMAC and NullRDC.

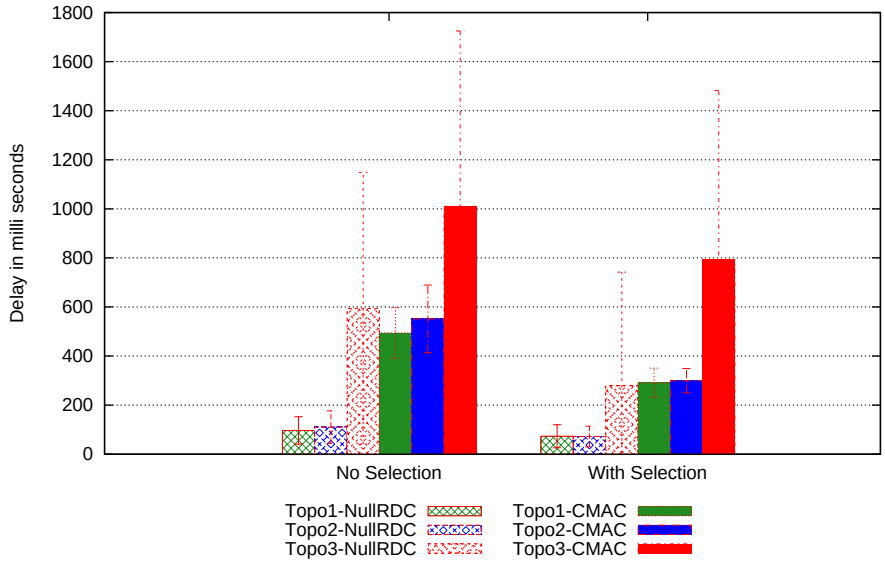


Figure 6.10: Average Service Invocation delay in ContikiMAC and NullRDC cases for topologies 1, 2 and 3

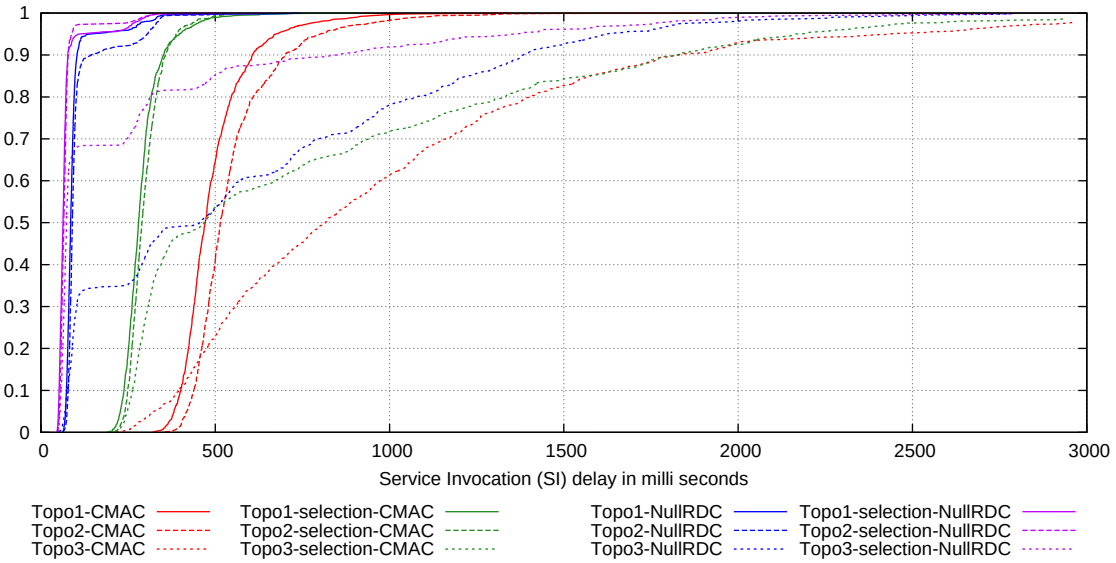
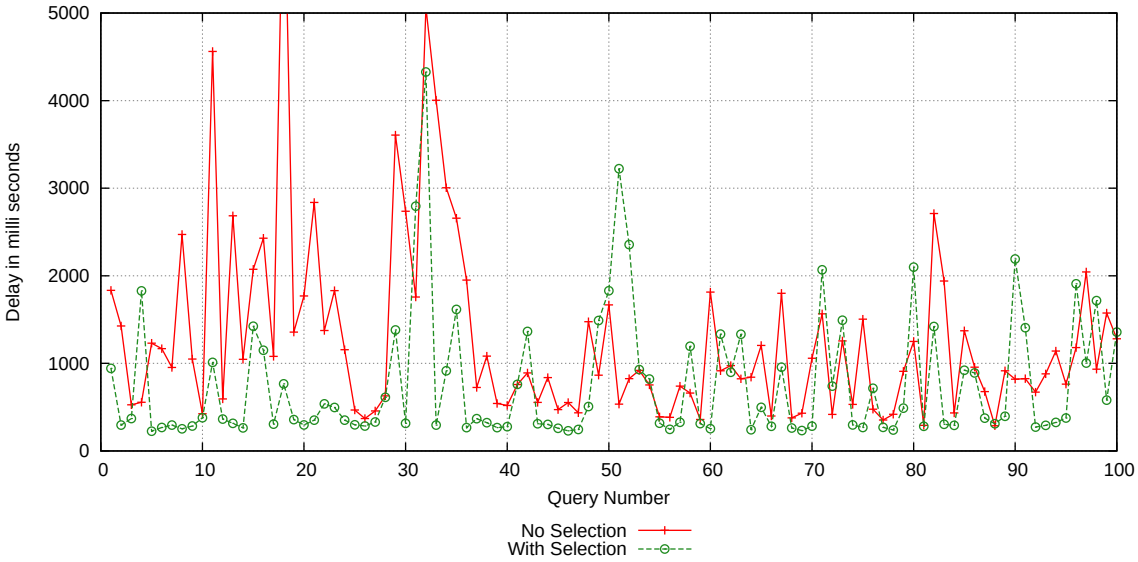
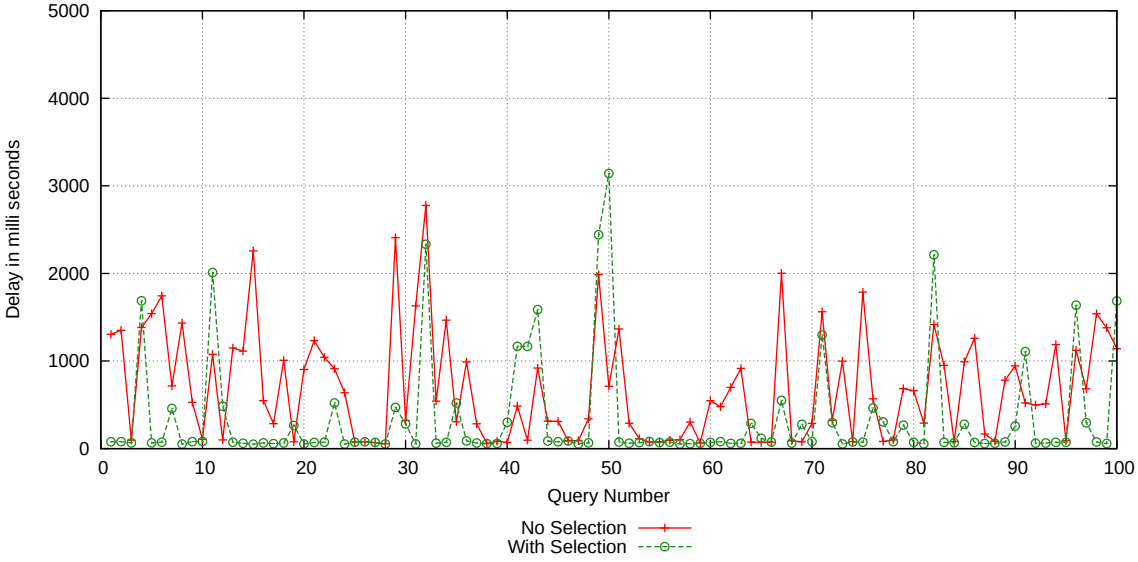


Figure 6.11: CDF graph of Service Invocation delay in ContikiMAC and NullRDC cases for topologies 1, 2 and 3



(a) ContikiMAC scenarios



(b) NullRDC scenarios

Figure 6.12: Service Invocation delay for first 100 queries for topology 3

6.5.4 Energy Consumption and network lifetime

This section covers the detail of energy consumption and network lifetime only for ContikiMAC scenarios, because NullRDC keeps the radio always on and consumes a tremendous amount of energy, which is significantly higher to be compared. Figure 6.13 shows that TRENDY's service selection improves the energy consumption of nodes. It also explains that most of the nodes are benefited in terms of energy efficiency in all scenarios. This efficiency is the result of preference of closer and more reliable nodes by the service selection mechanism. Topology 1 is more efficient in terms of energy efficiency for all nodes, because of its routing configurations. Furthermore, topologies 1 and 2 have the same trend for energy efficiency when service selection is employed. However, the high density of topology 3 results in more energy consumption for all nodes in the network compared to other topologies. Figure 6.14 shows that topologies 2 and 3 have a similar trend for the top five nodes in energy consumption, because of their un-even DODAG trees (shown in Figures 6.2b and 6.3b). Table 6.4 estimates the network lifetime in the number of days and concludes that service selection increases the lifetime of the network as well.

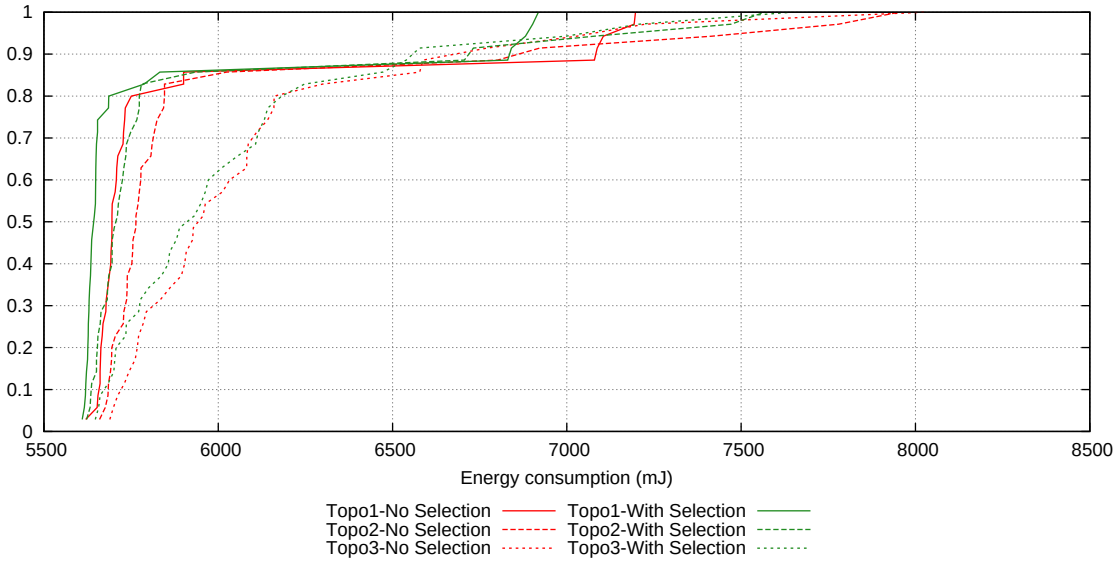


Figure 6.13: CDF graph of energy consumption per node in ContikiMAC scenarios for topologies 1, 2 and 3

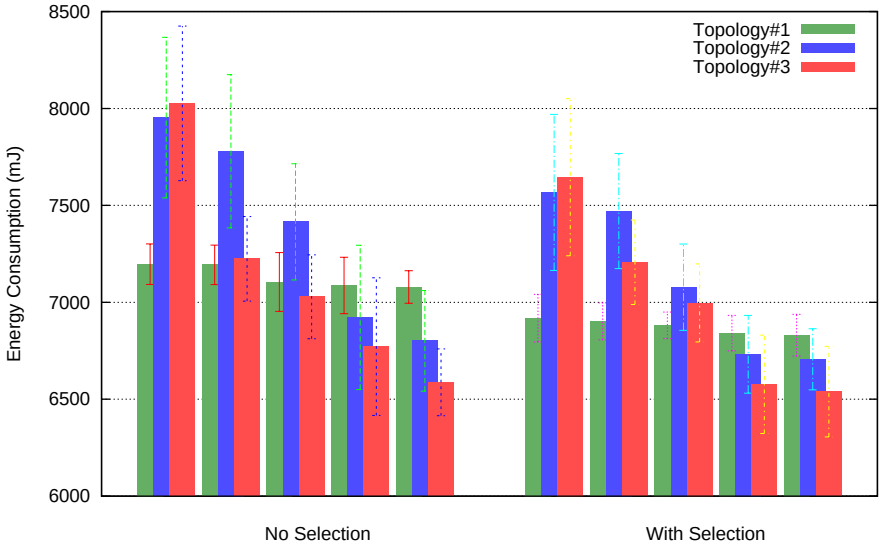


Figure 6.14: Energy consumption of top five nodes in ContikiMAC scenarios for topologies 1, 2 and 3

Table 6.4: Service Selection: network lifetime in days estimation for 35 nodes

Topology	**No Service Selection**	**With Service Selection**
1	390	405
2	350	370
3	350	365

6.5.5 Packets at the DA: a scalability factor

TRENDY's service selection is not aimed to decrease the number of packets at the DA, so all cases have the similar number of application-level messages received at the DA (excluding service invocation messages). However, Figure 6.15 shows that there is a slight benefit of using the service selection mechanism in topology 3 in terms of total traffic to or from the DA. This gain comes from the reduction of the number of neighbour discovery messages, because the DA sits on tops of DODAG's root and once the closest node is selected to serve a query its entry in the neighbour's table (maintained by neighbour discovery) is also updated. Consequently, the network traffic in topology 3 that's decreased around 10%.

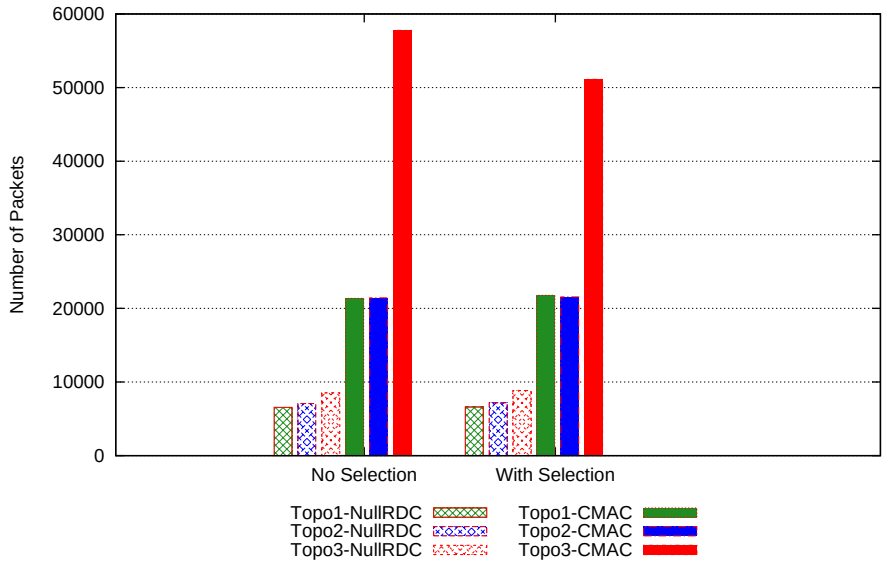


Figure 6.15: The DA traffic in ContikiMAC and NullRDC cases for topologies 1, 2 and 3

6.5.6 Reliability and Accuracy

The reliability and accuracy of TRENDY are explained in Section 6.2.6 holds true for service selection scenarios. However, the context-aware selection of service hosts increases the reliability, because the selection criteria (Section 5.8) chooses the most reliable SA. All service invocation responses are verified and validated, using logs and sniffed packets (Appendix B), and are found accurate. Consequently, it is concluded from the validation that the DA has always sent reliable SD information to the DA.

6.6 Summary and Discussion

This chapter has explained the SD performance metrics, experimental setup and design of generating the results from the performed experiments. Furthermore, the results of the experiments done to analyse the service selection mechanism are presented and discussed. The analysis of the results of experiments explains the impact of the service selection mechanism from different perspectives. The total network traffic analysis highlights the different behaviour of a network depending on its physical topology's density and generated RPL tree. Service selection only improves the traffic by reducing the number of service invocation requests. However, the rest of the traffic still depends on the physical topology and organisation of the RPL tree. The more dense network incurs extra neighbour discovery traffic and causes extra delays for packets. Therefore, the density of the network diminishes the service invocation delay efficiency introduced by service selection, because the high delay of small percentage of service invocation queries aggregates and increases the overall average delay. Similarly, the energy consumption of nodes also found higher in a dense network because the nodes need to be awake more often to update their neighbour discovery table's entries. In summary, the service selection mechanism improves the control overhead, service invocation delay and energy consumption; however, its level of efficiency depends on the density of the network and formed routing topology.

The next chapter discusses the proposed adaptive timer mechanism.

Chapter 7

Adaptive Reporting Timer

7.1 Introduction

Chapter 5 proposes a context-aware SD solution for the IoT. The solution provides an interoperable and open framework to work with different data formats accessible using compact CoAP-based RESTful web services. However, all services are registered at the DA and then status maintenance messages required to be sent to the DA as well. The generated traffic increases the energy consumption of nodes, as other nodes usually need to pass the control traffic towards the DA in a multi-hop network. This chapter proposes an adaptive timer algorithm that reduces the control traffic. Consequently, nodes become energy efficient and scalability is also improved, as control traffic not remains linearly proportional to the increasing number of nodes in a network. The detail of the mechanism's aims and challenges, design, message format and experiments with the results are covered in this chapter.

7.2 Aims

The main goal of the adaptive timer algorithm is to decrease the number of control packets by adaptively increasing the interval between status updates. This has multiple benefits: it will decrease the control packet overhead and will improve the energy consumption. However, this will consequently, sacrifice the inherent reliability by increasing the probability that the DA will forward stale (outdated) information to a UA. Therefore, the challenge for timer is to reduce this risk by sensing the requirements of a network and its dynamics.

Following is the summary of the timer mechanism's objectives:

1. The number of status updates should be reduced to improve the energy consumption of constrained nodes and network.

2. The decision of increasing the interval should be based on some network dynamics and demand of services. Furthermore, it should be adaptive enough to decrease the interval when it is required.

7.3 Adaptive Timer Process

The TRENDY timer mechanism increases the interval between status updates w.r.t. the number of times a SA sends an update. Therefore, all SAs use the basic time window period of the DA (Section 5.6) to calculate the status interval. Later on, the DA sends a TRENDY counter value in the response payload of every status update, which is used by the corresponding SA to increase the status update interval. The SA multiplies the initial status interval, derived from the DA's time window, with the received counter value to schedule the next status update message. This adaptively decreases the number of control packets from a SA, and consequently other nodes will also benefited with the reduced traffic in a multi-hop network.

The DA is configured with a general upper bound for the maximum TRENDY counter value, and each SA record, in the DA's registry, has its individual maximum limit as well that is adaptively changed. Furthermore, a hit count attribute for each service is also maintained by the DA, which is incremented whenever a service is discovered and selected. This hit count or popularity or demand of a service is used as a basic criterion for adaptation. Timer adapts to the current popularity of services hosted at a SA to increase or decrease the corresponding counter limit. Therefore, any SA hosting more popular services is needed to send status update more frequently. The process is adaptive, as the individual counter limit value for a node is increased by the DA for prolonging the status update interval only when hosted services of a SA not remain popular.

7.3.1 Design

Following are the parameters maintained by the DA to enable the adaptive timer algorithm:

- `max_trendy_counter_limit`: This is global maximum value for adaptive timer that is used as default upper bound of the TRENDY counter.
- `node_trendy_max_limit`: The individual upper bound for the TRENDY counter maintained for each registered SA. This bound can't exceeded then global maximum value.

- `hit_count_threshold`: The threshold value to determine the decision point when hosted services of a SA become enough popular. This attribute is used by the DA before increasing or decreasing `node_trendy_max_limit` of a SA. This threshold can be set at any value up to the application requirements, and its effect will only become apparent when same service is selected multiple times for SD queries.

- `retain_threshold`: It determines the mimimum number of hit counts required to keep the decremented `node_trendy_max_limit` unchanged.

- `timer_step`: Whenever the `hit_count_threshold` is exceeded, the `node_trendy_max_limit` is decremented by this value for the corresponding SA. This step value is also used to increment the SA's maximum limit when hit count remains less than `retain_threshold`.

- `node_trendy_counter`: It represents the TRENDY counter value for each registered SA, and its value never surpasses `node_trendy_max_limit`.

- `node_trendy_current`: This attribute is updated with `node_trendy_counter` value each time a status update is received from a SA. This value is decremented after every time window until reaches to 1.

- `node_trendy_counter_changed`: This is a flag used to ensure that multiple changes are not made to a `node_trendy_counter` in one time window.

All attributes starting with `node` prefix are maintained for each SA, whereas others have global scope. Each SA only requires `trendy_counter` attribute that has a default value set at 1, which is updated when a new value is received from the DA

Following is the procedure of the adaptive timer:

Step 1 - Initiation: The DA starts with the configured parameters, and it follows the Algorithm 1 when it receives a registration message from a SA. This algorithm initialises the TRENDY counter variables for the node and sets the flag to avoid multiple changes in its value with in a time window. The SA gets a basic time window period in response to its successful registration at the DA.

Step 2 - Evaluation on receiving a Status update message: The DA follows Algorithm 2 at receipt of every status update message. This algorithm checks and increment the `node_trendy_counter`, which subsequently enables the SA to increase the time period for its next status update message. This

Algorithm 1 DA at SA's registration time

```
1: node_trendy_max_limit ← max_trendy_counter_limit
2: node_trendy_counter ← 1
3: node_trendy_current ← 1
4: node_trendy_counter_changed ← false
```

algorithm ensures that the counter value will only gets changed once in a time window. The status maintenance requests are responded by the DA with respective `node_trendy_counter` in the payload.

Algorithm 2 DA on receiving an status update message

```
1: if !trendy_counter_changed and trendy_current == 1 then
2:     if node_trendy_counter < node_trendy_max_limit then
3:         node_trendy_counter + +
4:         node_trendy_counter_changed ← true
5:     end if
6: end if
7: node_trendy_current ← node_trendy_counter
```

Step 3 - Report time interval calculation of a SA: The SA follows the Algorithm 3 on receiving response for the status update. This algorithm follows the basic interval calculation (Section 5.6.2) and then multiplies it with the received `trendy_counter` value to schedule the next status update.

Algorithm 3 SA report time interval calculation after getting response for a status update

```
1: trendy_counter ← received_trendy_counter
2: report_time ← 0.5time_period
3: report_time += random(between 0 and 0.5)time_period
4: if trendy_counter == 1 then
5:     default_report_time ← report_time    ▷ time saved at registration time
6: end if
7: report_time ← report_time * trendy_counter
```

Step 4 - Hit count evaluation at discovery time: The timer is adaptive to the hit count (popularity) of a service. The DA increments the hit count attribute of each service whenever its service description is passed to a UA. Then a evaluation of hit count is done by the DA as shown in Algorithm 4. This evaluation set the appropriate interval between status updates by changing the trendy counter of the SA.

Step 5 - Hit count evaluation at time window expiration: The DA follows Algorithm 5 at the expiration of each time window. This algorithm evaluate the demand of all services hosted by each device to update `node_trendy_max_limit`.

Algorithm 4 Hit count evaluation at discovery time

```
if node_hit_count > hit_count_threshold then
    node_trendy_max_limit ← node_trendy_max_limit − timer_step
    trendy_counter_changed ← true
    node_hit_count ← 0
end if
```

Algorithm 5 Hit count evaluation at end of a time window

```
if trendy_counter_changed == false then
    if hit_count < retain_threshold then
        node_trendy_max_limit ← node_trendy_max_limit + timer_step
        trendy_counter_changed ← true
        if node_trendy_max_limit > max_trendy_counter_limit then
            node_trendy_max_limit ← max_trendy_counter_limit
        end if
    end if
end if
```

7.3.2 Example Scenario

This section demonstrates the working of the timer mechanism by explaining it with some scenarios expressed in figures, which depict adaptive timer mechanism in action at different stages.

Figure 7.1 describes the adaptive timer in a scenario, where the hit count threshold was set at 2. The counter value kept on increasing over the time and reached its maximum. It keeps on updating the DA with the same large interval until its services become popular. The algorithm assesses this change of demand, and adaptively changes the maximum counter limit value to reduce the status update interval for the particular SA. Furthermore, Figure 7.2 shows the same scenario in the long run and describes the behaviour of timer when a service is not in demand for a SA. Moreover, Figure 7.3 expresses a possible state of the network while using the adaptive timer mechanism, where the nodes have different intervals as a result of the diversity of SD queries.

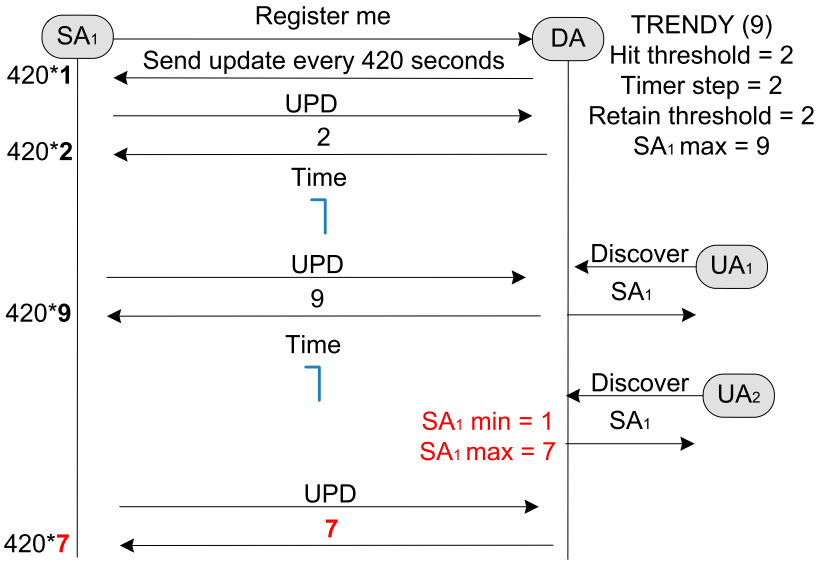


Figure 7.1: Adaptive timer's in a scenario

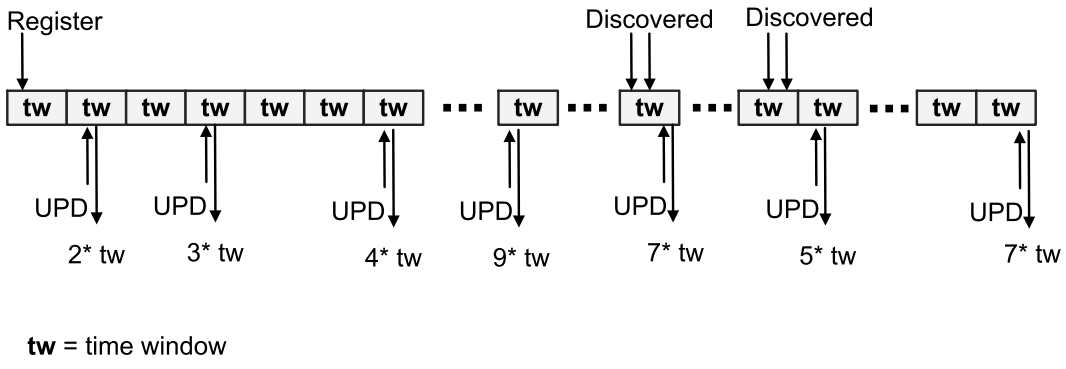


Figure 7.2: Timer's adaptability in the long run

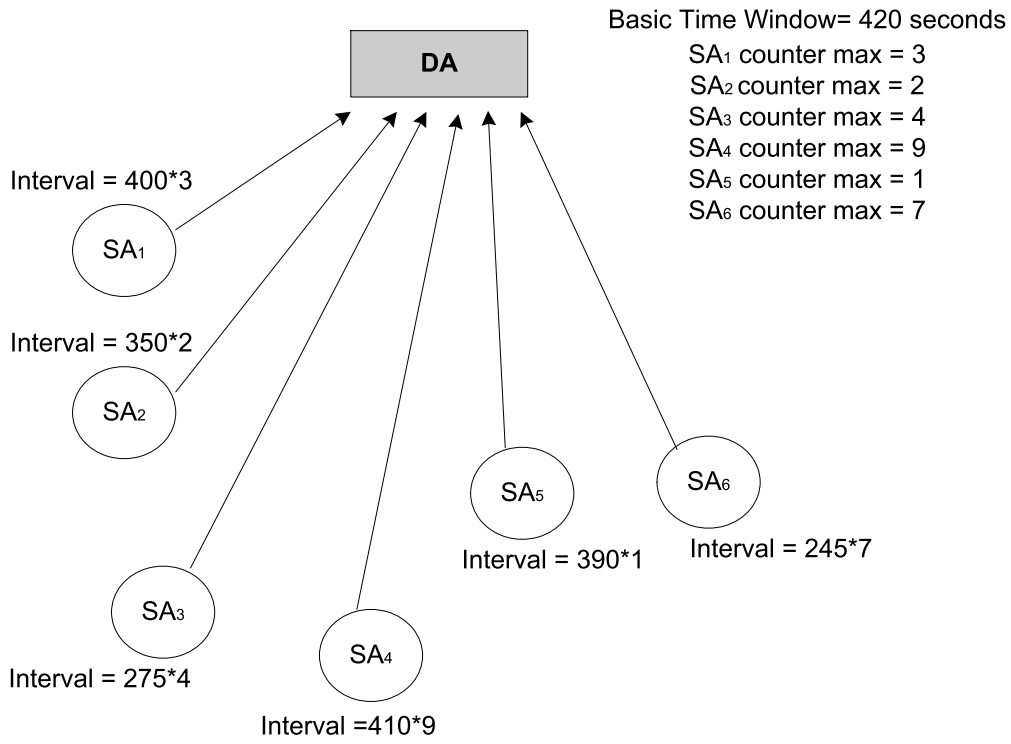


Figure 7.3: Behaviour of adaptive timer in a network

7.4 Message Formats

7.4.1 Updated TRENDY reporting message

The adaptive timer mechanism requires a small update in the response of the DA for status maintenance updates, as explained in Section 5.11.2. The DA now sends a `node_trendy_counter` in the payload of response of a status update to change the update interval of a SA. Figure 7.4 shows the updated message format.

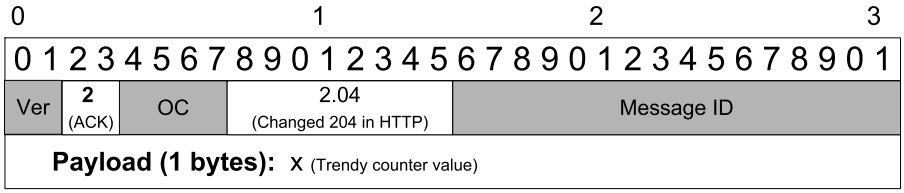


Figure 7.4: Adaptive timer: TRENDY message change for status maintenance

7.5 Experiments and Results

This section covers the detail about experiments, including scenarios, experimental setup and discussion of the results.

7.5.1 Introduction

Following scenarios are repeated for different number of nodes (15, 25 and 35 nodes).

Case 1 - Basic TRENDY without timer: This scenario only enables the basic functionality of TRENDY with service selection. The UA gets a best matched resource's URL and IP address of a SA hosting the matching service.

Case 2 - Basic TRENDY with timer threshold 1: This scenario has implemented adaptive timer with `trendy_counter_maximum` fixed at 9, `hit_count_threshold` at 1 and `timer_step` at 2 on top of case 1's functionality.

Case 3 - Basic TRENDY with timer threshold 2: This case is exactly like case 2 with `hit_count_threshold` fixed at 2.

The hit count threshold of timer is varied to check the effect of adaptive timer in scenarios of the different number of nodes. The reaction of timer mechanism in response to the change in a hit count threshold depends on multiple factors, including, the number of specific queries for a location, and the number of times when services of the same SA are selected by the service selection mechanism. In experiments, the UA queries are stateless, generated randomly and sent after a random interval. Therefore, only two threshold values (1 and 2) are selected for above scenarios after testing different values for 1000 queries. The selected values produced the maximum impact to be analysed with the query set, and other higher values produce almost the same result.

The simulations are configured with the same settings defined in Section 6.3 with the changes given in Table 7.1.

Table 7.1: Configurations for the experiments of adaptive timer

Total nodes	35 nodes + 1 border router
Physical topologies	1, 2 and 3 (Section 6.3.1)
RDC	ContikiMAC
Number of locations	5 with 7 SAs in each location
Number of cases	3 (Cases) × 3 (total nodes variations) = 9
Total Service Discovery queries	1000
Total Service Invocation queries	1000
First UA Query	At 600 seconds
DA time window	5 minutes = 300 seconds
Number of time windows	30
Simulation Duration	300 × 30 = 9000 seconds

This section continues with the generated results of simulations to analyse the effect of adaptive timer mechanism. The analysis is done for two sets of experiments: topology 1 with 15, 25 and 35 nodes, and topologies 1, 2 and 3 with 35 nodes. Both sets provide different perspectives of timer's impact in various sizes of networks with unique topologies.

7.5.2 Control packet overhead

The high control packet overhead is unsuitable for constrained networks, because its effect is amplified in a multi-hop network consists of sleeping nodes, which, consequently, consume more energy. The main aim of adaptive timer is to decrease the number of control packets generated by TRENDY.

Cases for different number of nodes: Figure 7.5 shows the significance of using adaptive timer for scenarios (Section 7.5.1) with 15, 25 and 35 nodes within a network with topology 1. It is evident that timer reduces the number of per node status update messages to one-fourth for about $70 - 85\%$ of the nodes in all scenarios.

The overhead is reduced further for more percentage of nodes with the increasing size of the network. This is the effect of adaptability of the timer algorithm that increases the status update interval for nodes with unpopular services. Therefore, those nodes consume fewer numbers of packets over the time. Thus, scenarios with 25 and 35 nodes have more nodes with the low application-level control packet overhead.

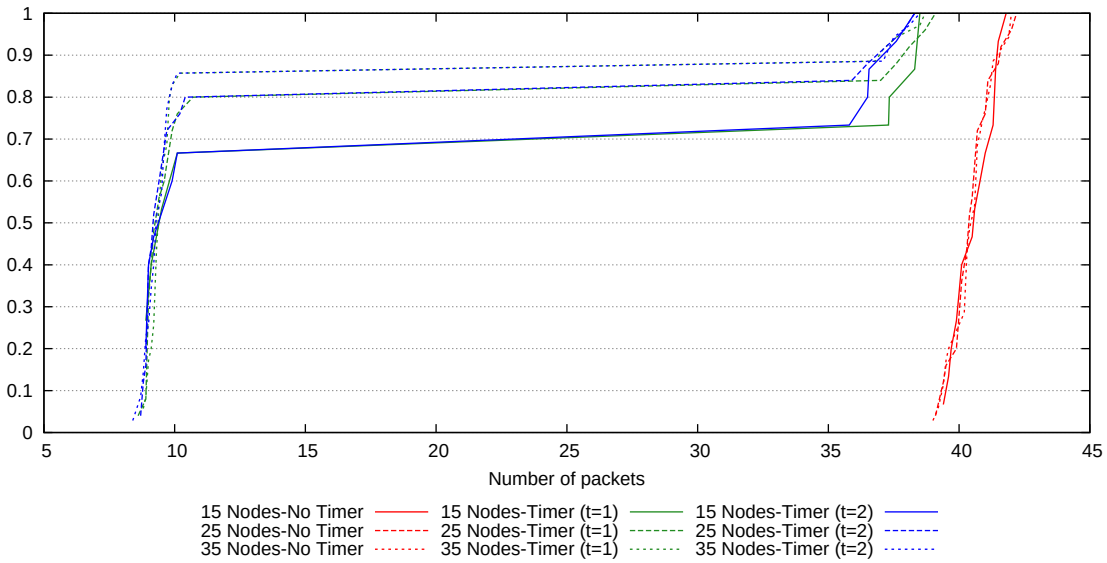


Figure 7.5: CDF graph of application-level control packet overhead per node with different timer threshold values for topology 1

Figure 7.6 shows the aggregated control overhead for all nodes in the network. It is clearly evident in the figure that employment of adaptive timer makes the solution scalable as increasing number of nodes only marginally increases the control overhead. The scenarios with 35 nodes have performed the best, where the number of packets reduced to more than one-third with the usage of adaptive timer because of the aggregated benefit of all nodes. In addition, Figure 7.7 shows the same trend in terms of total network traffic overhead. It can also be noticed that the decreased number of CoAP packets has also reduced the number of packets at layers below RPL in all scenarios. Moreover, Figure 7.8 presents the detail of CoAP packets by categorising those in SD and invocation packets. The number of SD packets in scenarios with timer demonstrate the same scalable behaviour. The cases with the hit count threshold 1 and 2 performed almost similarly because of the randomness of the used query set and the effect of service selection that chooses a better SA each time and hence reduced the number of hit counts for services. In summary, the rate of control packet overhead is not proportional to the increasing number of nodes, which indicates that timer has made the solution scalable.

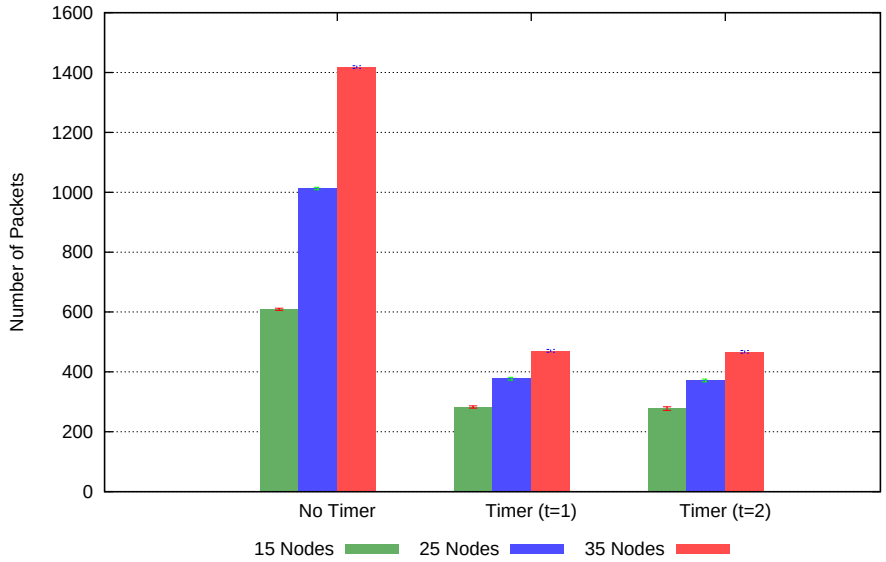


Figure 7.6: Aggregated Control packet overhead of the network for topology 1

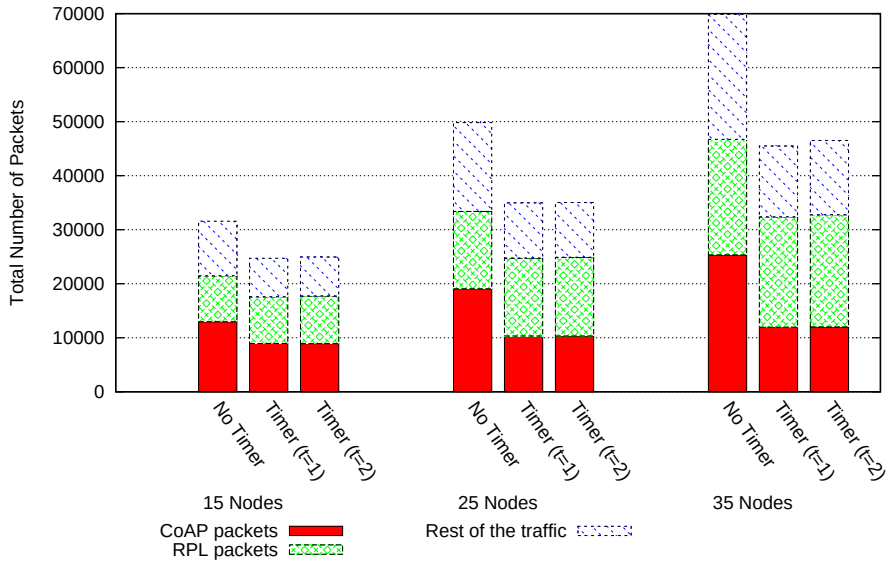


Figure 7.7: Total traffic of the network for topology 1

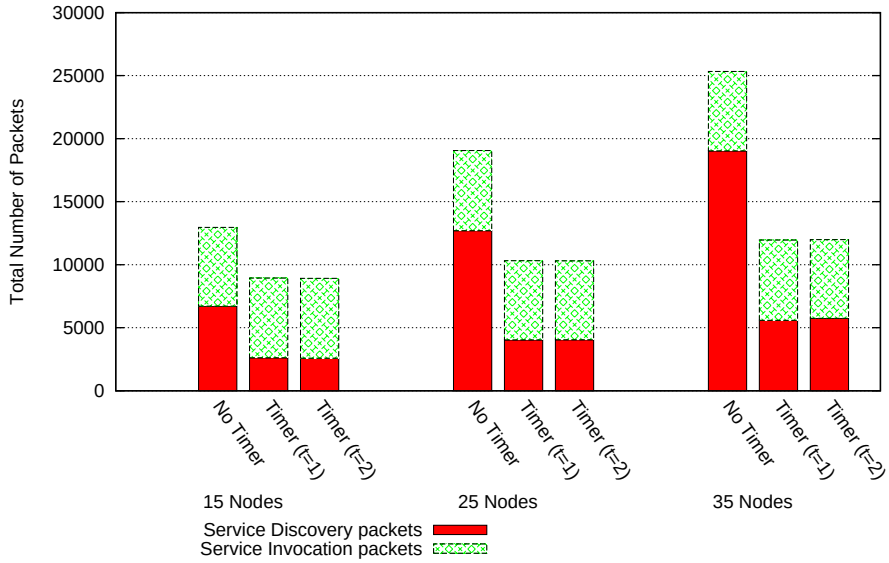


Figure 7.8: Total CoAP traffic of the network for topology 1

Cases for different topologies: This section discusses the timer's effect in three topologies with different densities (explained in Section 6.3.1). Figure 7.9 shows that all the topologies have shown a similar trend like topology 1 when the timer mechanism is employed. This validates the application-level packet overhead that should be similar for different topologies. However, the total packet overhead at all layers, presented in Figure 7.11, shows that the performance for topology 3 is not equivalent to other topologies, because it has more nodes one-hop away from the root. Therefore, the service selection mechanism selects different SAs each time to serve a query. The difference between the performance of both timer scenarios becomes visible for topology 3, where the more sensitive timer (with threshold at 1) produced extra traffic because of its quick adaptability.

The aggregated application-level control overhead is shown in Figure 7.10, which points out the scalability introduced by timer. Furthermore, Figure 7.11 presents the overhead detail of the whole network and explains that the reduction of the application-level messages has resulted in the decline of overall traffic. The trend is same for all topologies, but topology 3 gained the maximum benefit because of its density with the 40% reduction in overall traffic. Figure 7.12 elaborates that SD packets (updates, etc.) are reduced to around $50 - 60\%$ in all scenarios. However, topology 3 has slightly fewer SD packets than other topologies, which indicates its low average hop count distance between nodes and the DA because of high density.

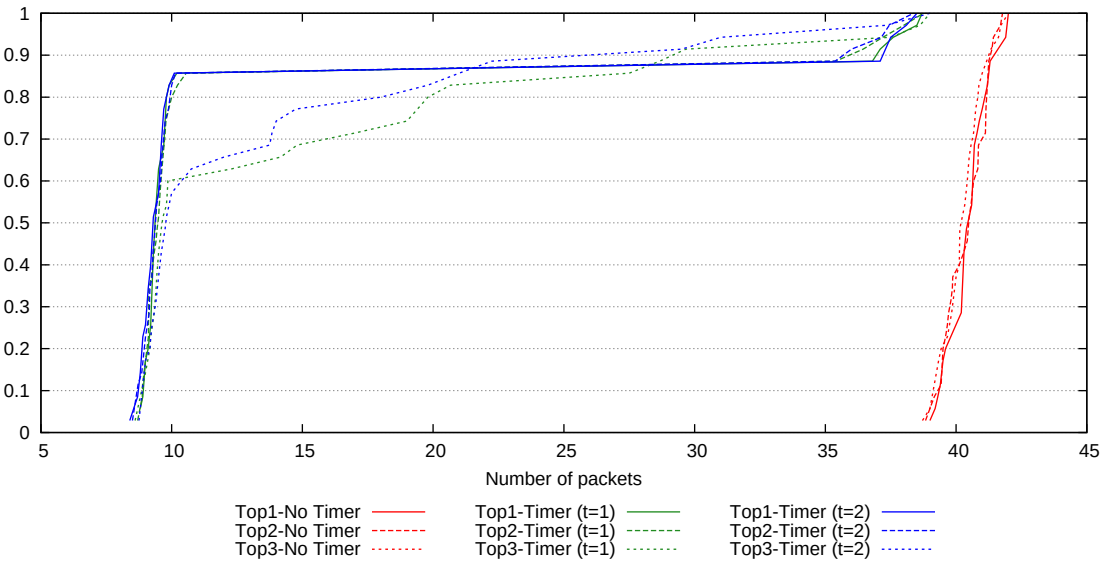


Figure 7.9: CDF graph of individual control packet overhead per node with different timer threshold values for topologies 1-3

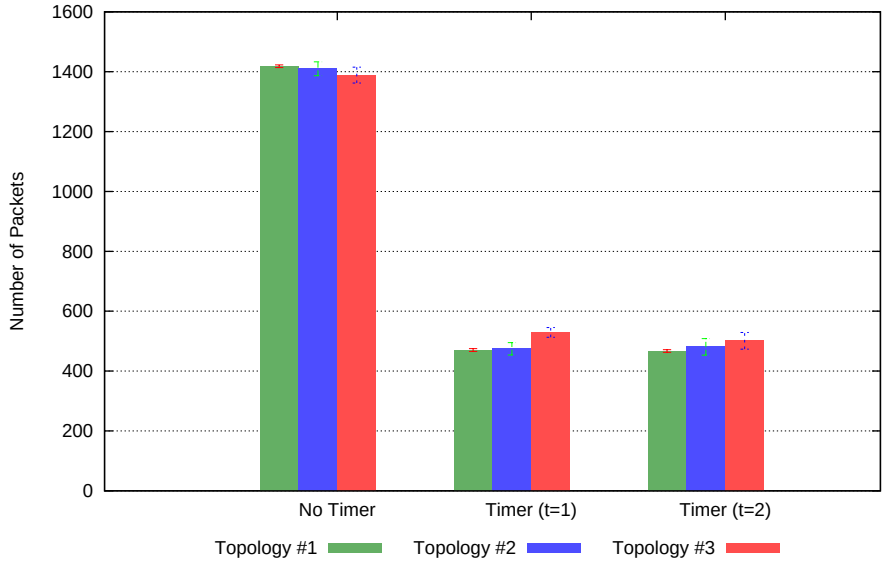


Figure 7.10: Aggregated application-level control packet overhead with different timer threshold values for topologies 1, 2 and 3

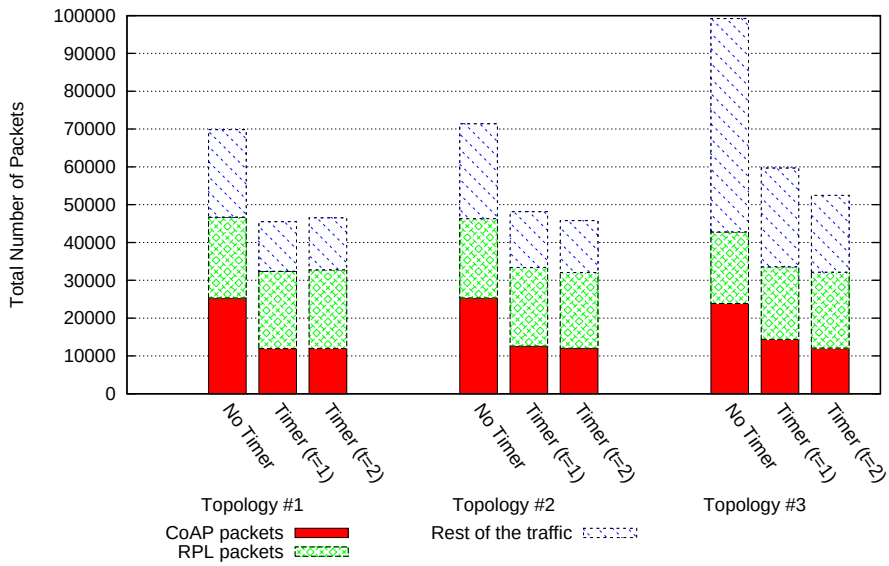


Figure 7.11: Total traffic of the network for topologies 1-3

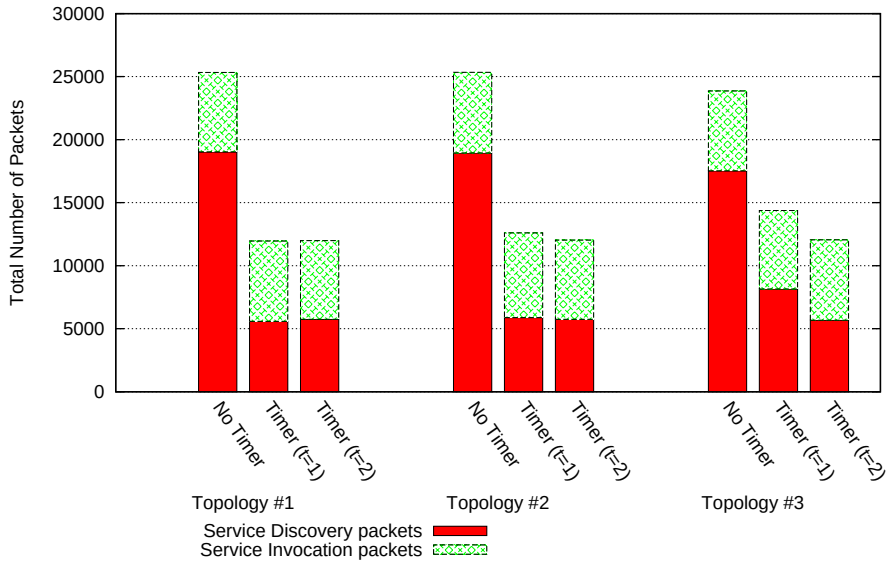


Figure 7.12: Total CoAP traffic of the network for topologies 1-3

7.5.3 Energy Consumption and network lifetime

Cases for different number of nodes: Figure 7.13 presents the individual energy consumption for scenarios with 15, 25 and 35 nodes using topology 1. The overall energy consumption rose with the increasing number of nodes in the network; however, the energy efficiency for the timer scenarios also increases from around 200mJ to 800mJ. This behaviour supports the argument that adaptive timer increases the scalability of the solution from

energy consumption's perspective as well. Both the scenarios of timer have performed almost similarly in these simulations with 1000 queries.

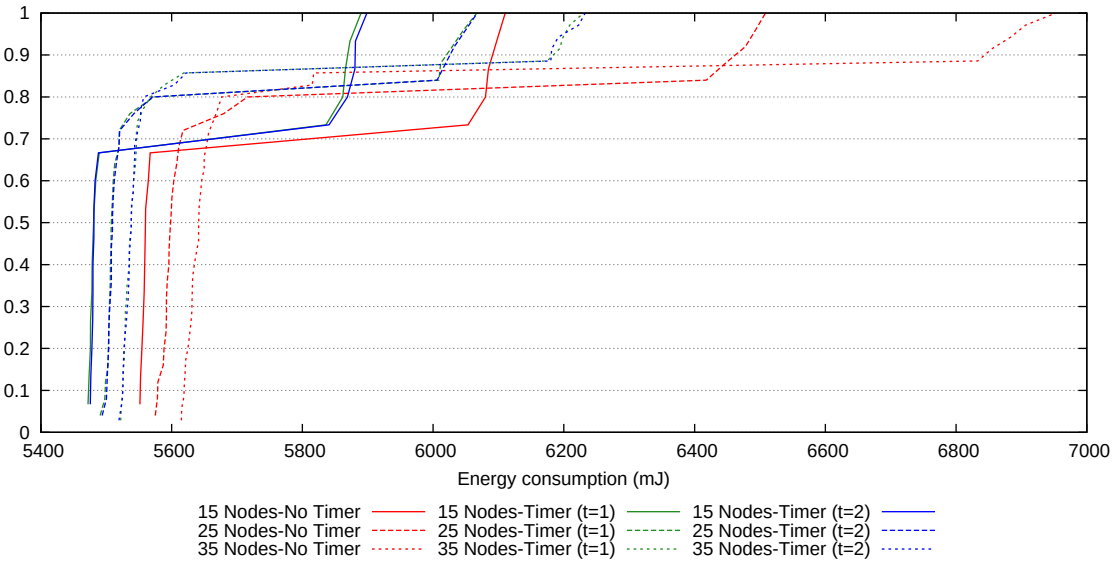


Figure 7.13: CDF graph of energy consumption per node for scenarios for topology 1

Figure 7.14 shows the topmost five nodes in energy consumption. In addition, the simulation time (9000 seconds) and energy consumption of the top-most node are projected linearly in Table 7.2 w.r.t. the available energy provided by 2 AA batteries (Section 6.2.4). It is evident from both figure and table that network will last longer in adaptive timer scenarios, because the timer has reduced the energy consumption for the nodes. The network lifetime approximation in Table 7.2 shows that the network will last for around 10% extra time by employing the timer mechanism.

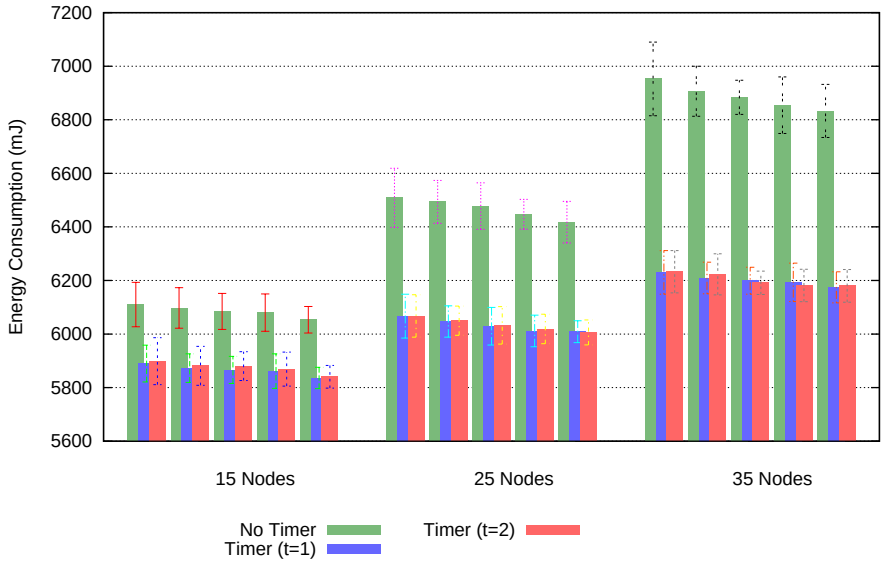


Figure 7.14: Top five nodes in energy consumption for topology 1

Table 7.2: Adaptive timer: network lifetime estimation in days for 35 nodes

Technique	**Topology#1**
No timer	400
With timer	445

Cases for different topologies: Figure 7.15 shows the energy efficiency gained in all topologies when timer is employed for 35 nodes. All topologies 1-3 experienced a gain in energy efficiency for most of the nodes. However, it is evident that topology 3, because of its density, still has higher energy consumption for individual nodes than other topologies. Anyhow, it also shows that topologies 3 has gained more energy efficiency compared to other topologies. The extra gain was actually comes from the reduction of packets generated at lower layers, as presented in Figure 7.11. Moreover, Figure 7.16 presents the top five nodes of each topology in energy consumption. The approximated network lifetimes for all topologies are given in Table 7.3, which further support the benefit of using timer mechanism.

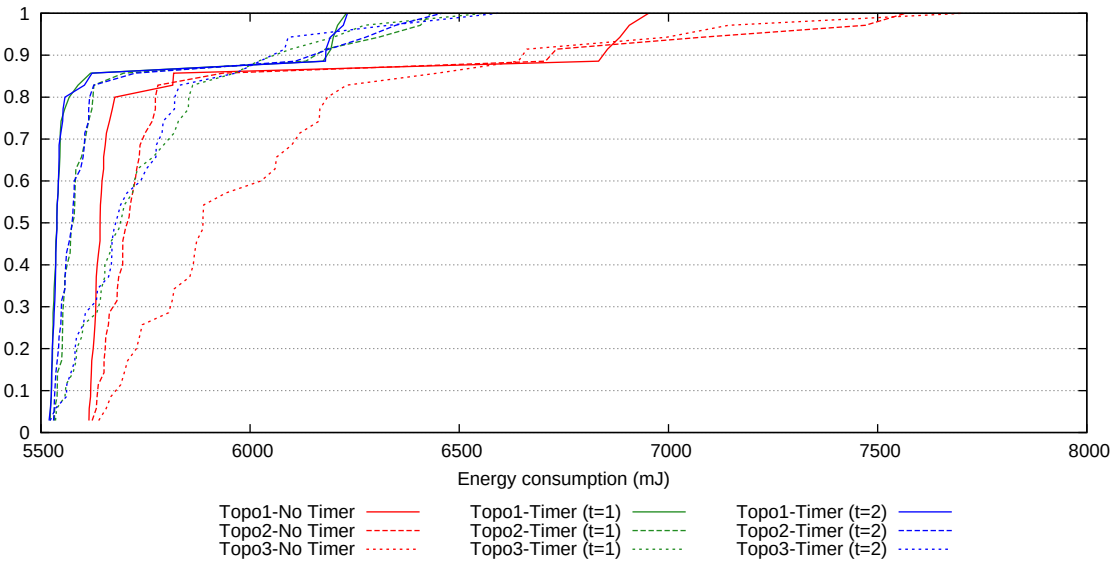


Figure 7.15: CDF graph of energy consumption per node for scenarios for Topology 1

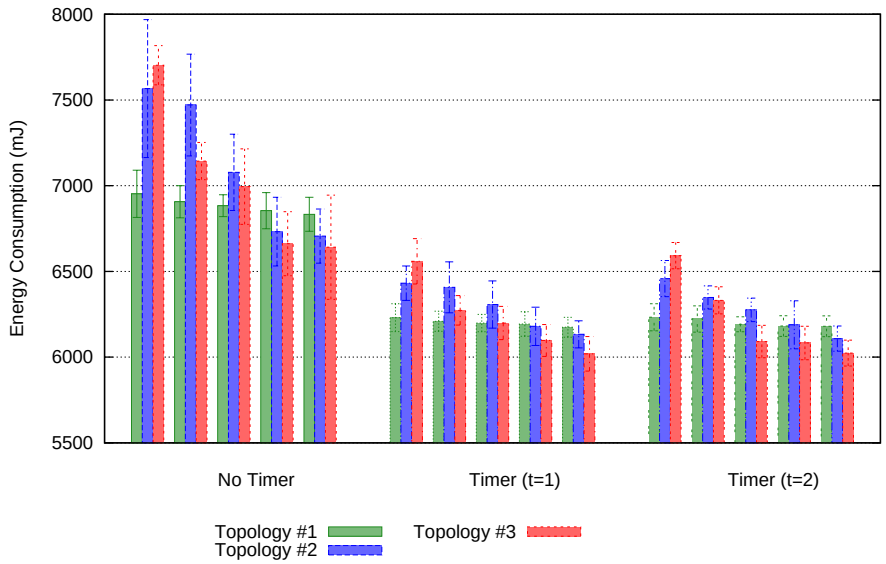


Figure 7.16: Top five nodes in energy consumption for topology 1

Table 7.3: Adaptive timer: network lifetime estimation in days for all topologies (with 35 nodes)

Technique	**Topology #1**	**Topology #2**	**Topology #3**
No timer	405	370	365
With timer	445	435	425

7.5.4 Scalability factor: Packets to the DA

Cases for different number of nodes: Figure 7.17 shows the number of packets received by the DA from 15, 25 and 35 nodes networks using topology 1. The scenarios with the adaptive timer mechanism have generated fewer control overhead packets, consequently; fewer numbers of application-level packets are received at the DA. Moreover, the total packets received and sent by the DA at all layers are shown in Figure 7.18. It can be noticed that the number of packets at the DA is not linearly proportional to the size of the network, which concludes that using a timer makes TRENDY more scalable.

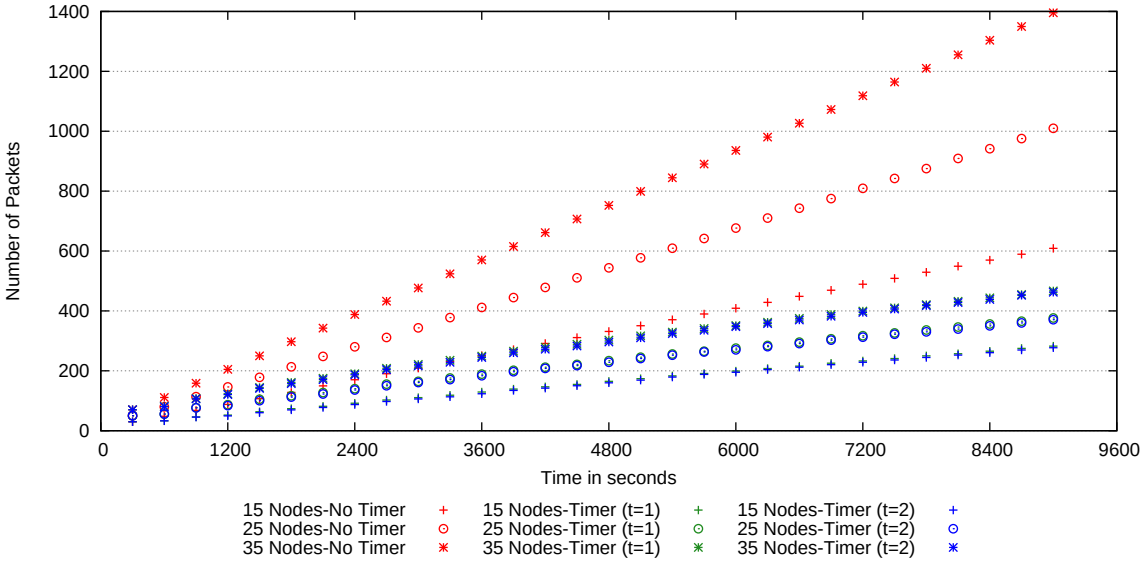


Figure 7.17: Packets received at the DA for topology 1

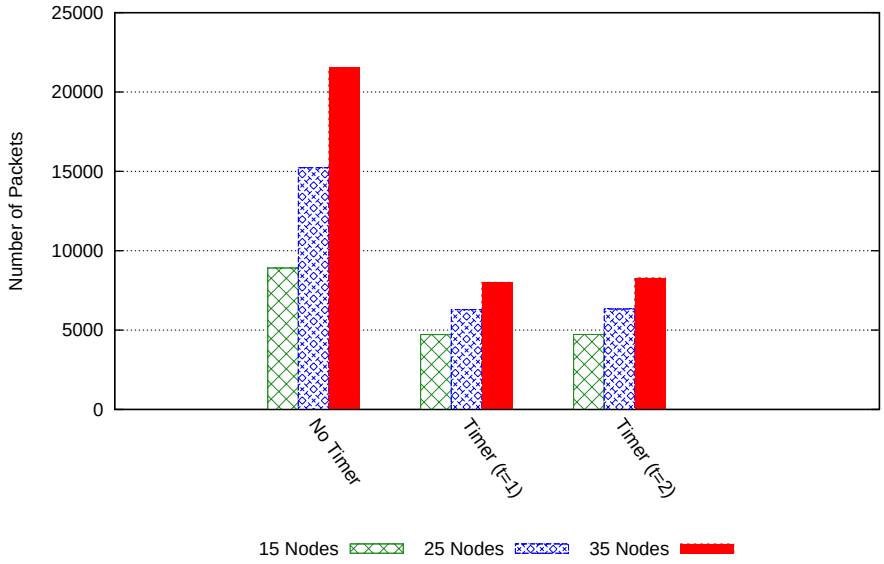


Figure 7.18: Total traffic at the DA for topology 1

Cases for different topologies: Figure 7.19 presents the application-level messages received at the DA. Fewer packets in scenarios with timer show that the traffic flow at the DA is decreased considerably. Moreover, Figure 7.20 presents the overall traffic flow at the DA in terms of all packets sent or received by the DA at all layers. It can be noticed that timer has reduced the number of packets to less than half in all topologies. However, topology 3 has gained the highest efficiency, but still has almost twice the number of packets compared to other topologies, because it has more direct neighbours to the DODAG root where the DA is placed.

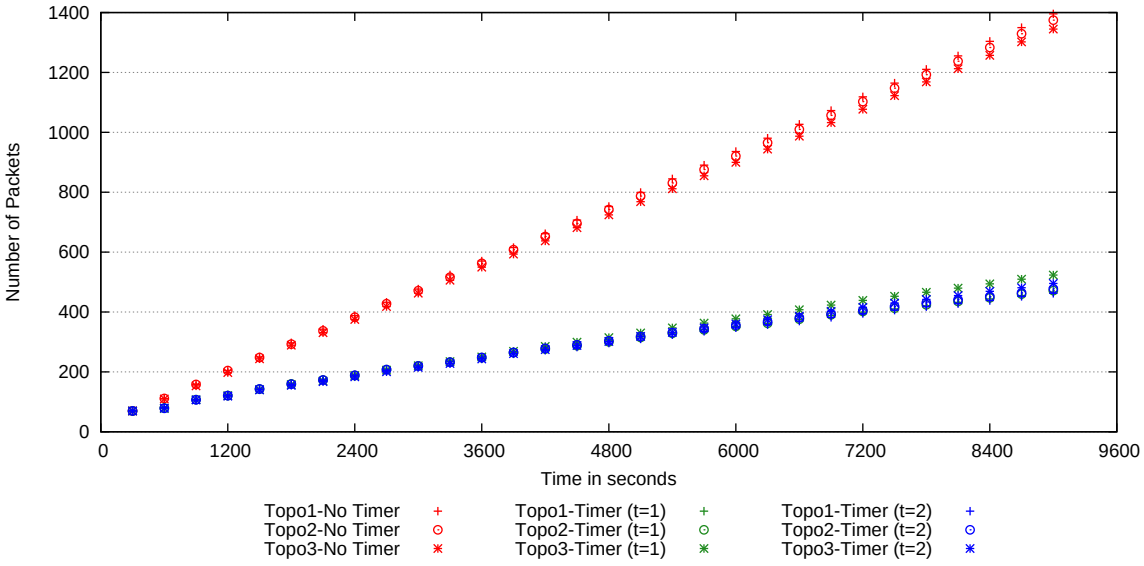


Figure 7.19: Packets received at the DA from 6LoWPAN for topologies 1-3

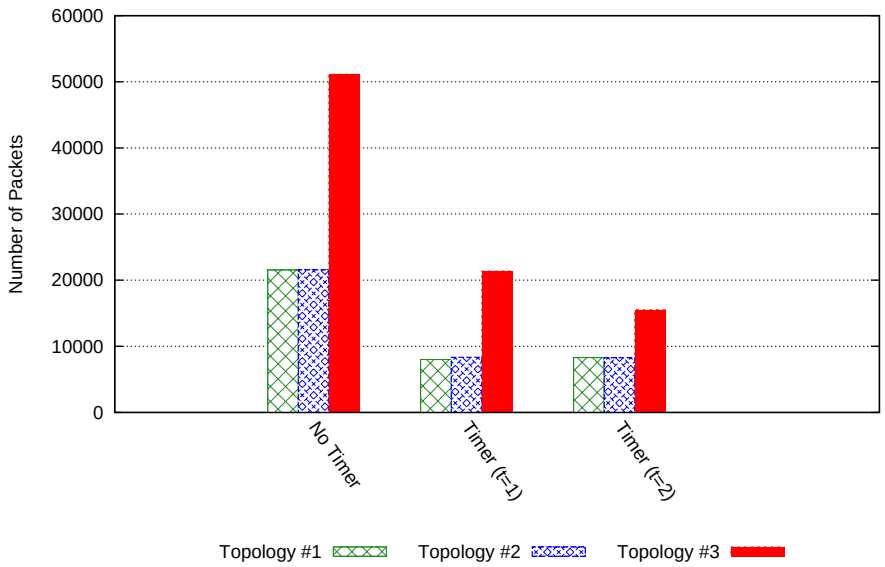


Figure 7.20: Total traffic at the DA for topologies 1-3

7.5.5 Service Invocation Delay

Figure 7.21 shows that adaptive timer had almost no impact on the service invocation delay in scenarios with topology 1. On the other hand, Figure 7.22 presents a unique phenomenon for topology 3, where timer has improved the average service invocation delay. This efficiency is the result of considerable reduction of total traffic in the network for topology 3, as shown in Figure 7.11. Figure 7.23 provides another perspective, which explains that the reduction in traffic actually improves the delay for around 30-40% of queries in scenarios with topology 3, and consequently, improves the average service invocation delay.

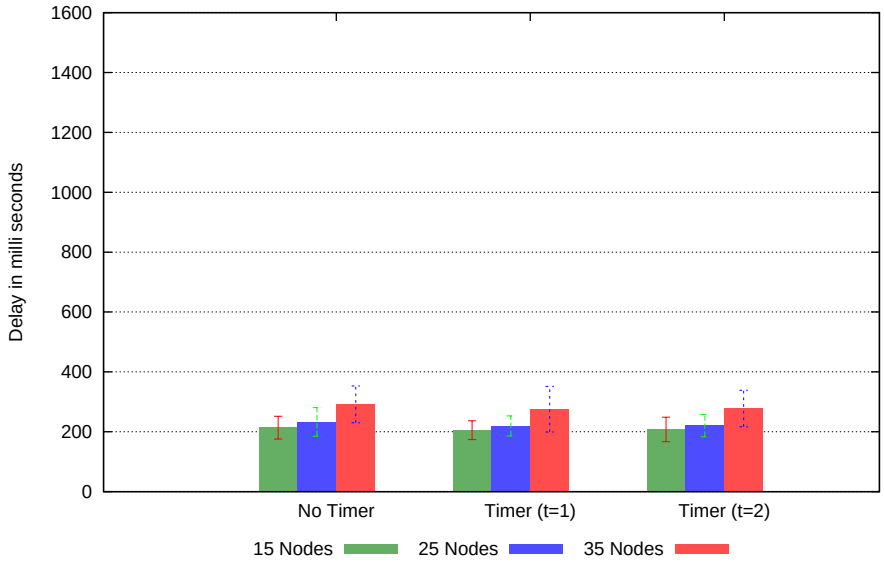


Figure 7.21: Average Service Invocation delay for topology 1

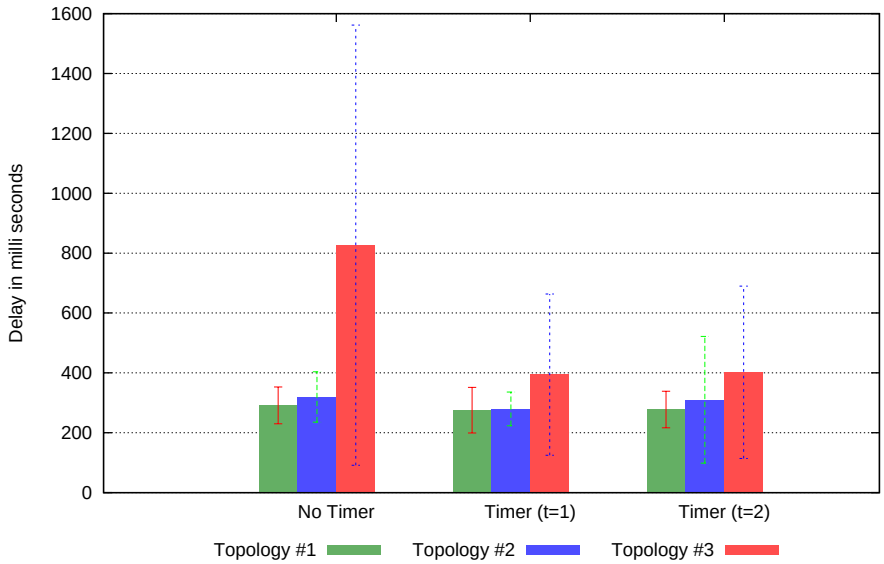


Figure 7.22: Average Service Invocation delay for topologies 1-3

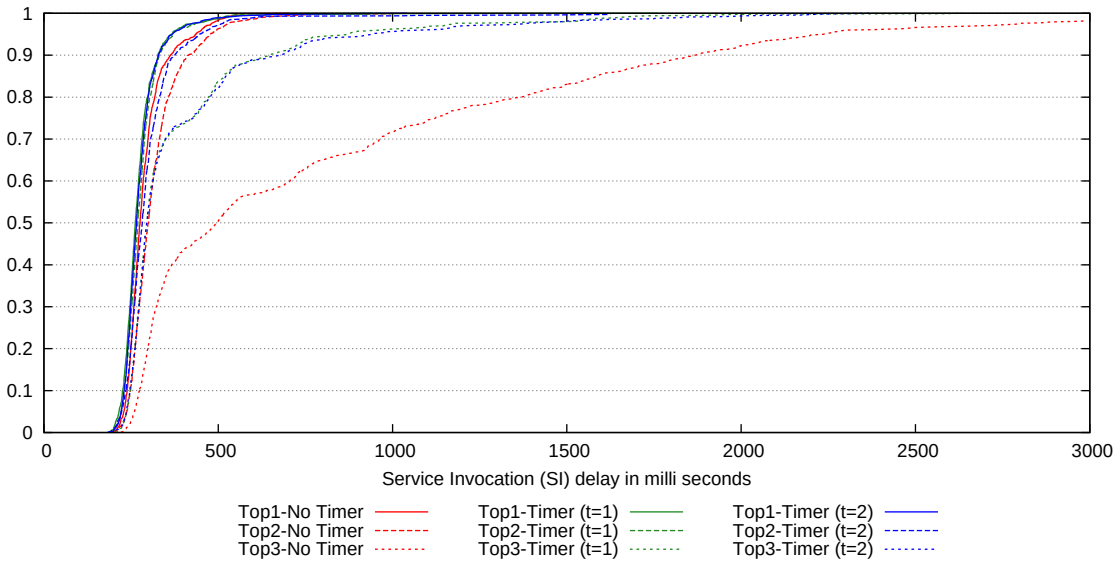


Figure 7.23: CDF graph of Service Invocation delay for topologies 1-3

7.5.6 Reliability and Accuracy

The adaptive timer mechanism increases the interval between status updates to decrease the control overhead. This results a the more energy-efficient and scalable solution. However, this also risks the validity of service information at the DA, as a SA hosting services might fail or replaced before the next status update is due. The timer mechanism deals this by gradually decreasing the status update's interval for the nodes with popular services (Section 7.3.1). Nevertheless, there is still a chance that the DA can send stale service information to the UA.

In all experiments, no node failed during simulations, and all service invocations were successful. In addition, the validation of all service invocation queries (explained in Appendix B) concludes that all responses were correct. This describes the success of the DA to provide accurate service information to all SD queries, as each SD query is immediately followed by a service invocation request. Overall, the risk of unreliable information from the DA still exists; however, the DA can be configured dynamically to increase or decrease the counter's maximum or minimum values to control the frequency of SD messages.

7.6 Summary and Discussion

The node density plays an important role in the overall overhead of a network. The number of neighbour discovery messages for a node increases in a dense vicinity. This also raises the delay for passing the messages over a multi-hop network. On top of that, the SD mechanism adds its burden, which aggregates to the cost in terms

of longer delays, overheads and energy consumption. Consequently, constrained environments with battery-operated nodes need more frequent status updates. The status maintenance of TRENDY is a major overhead; however, a large time window for maintaining registrations of constrained and unpredictable nodes is also not a worthy idea. Therefore, the adaptive timer mechanism is designed to trade off the involved risk of obsolete information. This mechanism adaptively increases the status interval over the time depending on the number of times a node has sent updates. However, the interval is limited by a bound, which adapts to the popularity of a node's hosted services. As a result, a node is asked to send updates more frequently if its services are becoming popular. There are several other aspects, which can be included into the equation used for adaptation, e.g., node's energy and its number of hosted services, etc. However, the hit count translates to the actual popularity and demand of node's services and hence selected as a criterion for adaptation.

This chapter has covered the detail of the adaptive timer mechanism that makes TRENDY energy-efficient and scalable by reducing the number of control packets. The results generated by various experiments using the different number of nodes and topologies support the benefit of using the timer algorithm in providing the energy efficiency and scalability. The gain in energy efficiency increases the network lifetime, and scalability allows more nodes to be part of the network without creating too many overheads. Furthermore, it is also concluded that the timer improves the performance further when the density of a network increases, because it reduces the overall network traffic. In a highly dense network, timer can even improve the average service invocation delay as well, because the reduced traffic allows the messages to propagate more quickly. All these benefits are achieved by introducing a mechanism that only requires one attribute at each SA and one message change in TRENDY (Chapter 5). However, the DA needs changes in its logic and new attributes for each node entry in its registry. The experiments validate the benefits of using the adaptive timer in terms of scalability and longer network lifetime.

The next chapter introduces a context-aware grouping mechanism proposed by this research project.

Chapter 8

Context-aware Grouping

8.1 Introduction

The centralised architecture of the proposed solution (Chapter 5) requires the registration and status maintenance updates to be received at the DA. However, this can lead to a single point of failure and also has a potential for bottleneck problem for scalability. The problem is partially addressed by embedding the DA role in an edge router. This can tackle the single point of failure issue, as an edge router is a bridge between a 6LoWPAN and the Internet. Furthermore, adaptive timer (Chapter 7) decreases the control traffic, but the inherent problem of scalability with centralised architecture still exists as all control traffic flows towards a single point. This chapter covers the detail of context-aware grouping technique, which solves the problem by localising the status maintenance task and provides the basis to enable service composition paradigm.

8.2 Aims

Following are the aims of context-aware grouping:

1. **Localise status maintenance**: The main goal of the grouping scheme is to localise the traffic generated by status updates.

2. **Efficient**: It should not overwhelm the network by extra burden of grouping messages, so grouping overhead should be minimum.

3. **Resource awareness**: The mechanism should not pose more requirements to constrained devices. On the other hand, it should be able to intelligently use the resources of highly capable devices.

4. **Service Composition**: It should address the demand of service composition for the IoT scenarios.

8.3 Architecture

The location-based grouping classifies a SA into Group Member (GM) or Group Leader (GL). Following is the detail of these new entities.

8.3.1 Group Member (GM)

A GM is the extended version of SA, as it needs to register its resources and send status updates. However, it sends the status updates to allocated GL after groups are formed. Each GM needs to maintain a RESTful reporting resource identified by the "`trendy/rep`" URL to support the grouping messages from the GL.

8.3.2 Group Leader (GL)

A GL maintains a group table to keep the record of its associated GMs. Each GM entry in the table has attributes: GM's IP address, a flag to confirm that a status is updated and a synchronisation flag to represent whether GM has been successfully notified about its grouping. The size of a grouping table depends on the capability of a GL and can vary.

Context-aware grouping also defines different GL roles to introduce a modular design. Therefore, a GL can choose a feature-set to implement according to its available resources. Following list shows different types of RESTful resources for GLs that should be understood by a DA.

GL status maintainer: This resource is slightly updated version of GM's reporting resource and identified by the same "`trendy/rep`" URL. This updated resource is a mandatory for a GL. On top of basic GM functionality, it also handles the UPD (Update) message (Section 5.11.2). If some GM failed to send a status update within an agreed interval period to respective GL, then a NRP (Not responded) message (Section 8.6.3.2) is used by the GL to inform the DA.

GL grouping resource: It is identified by the "`trendy/gl`" URL and a mandatory resource that should be offered by each GL. This resource allows a GL to receive grouping related messages, including YGM (Your Group Member) and RGM (Remove Group Member) messages (Section 8.6.2).

Forwarder: This optional resource is identified by "`trendy/glfwd`" URL. This resource tells the DA that the GL can also forward a PUT command to its GMs e.g., Switching light or heating actuators. Section 8.6.4.1 covers the detail of the related FWD (Forward Query) message.

Aggregator: This resource enables the GL to execute a composed service by collecting the local values to return an aggregated result to the enquirer, and has "`glagg`" URL.

GL as Local Proxy: This resource is identified by "`trendy/glproxy`" URL. Implementing this resource conveys to the DA that the GL can act as a local proxy and can respond to the querier on behalf of other SAs.

GL as Local DA: A GL implementing this resource can maintain a local DA for its group to serve SD queries only for a certain location. This resource uses "`trendy/glda`" URL.

Figure 8.1 presents new architecture of TRENDY after enabling context-aware grouping, which is behave in a distributed way compared to earlier one (Figure 5.1). This architecture is distinct from clustering as a GL is no obliged to provide the functionality of a distributed directory, nor it is elected by the surrounding nodes.

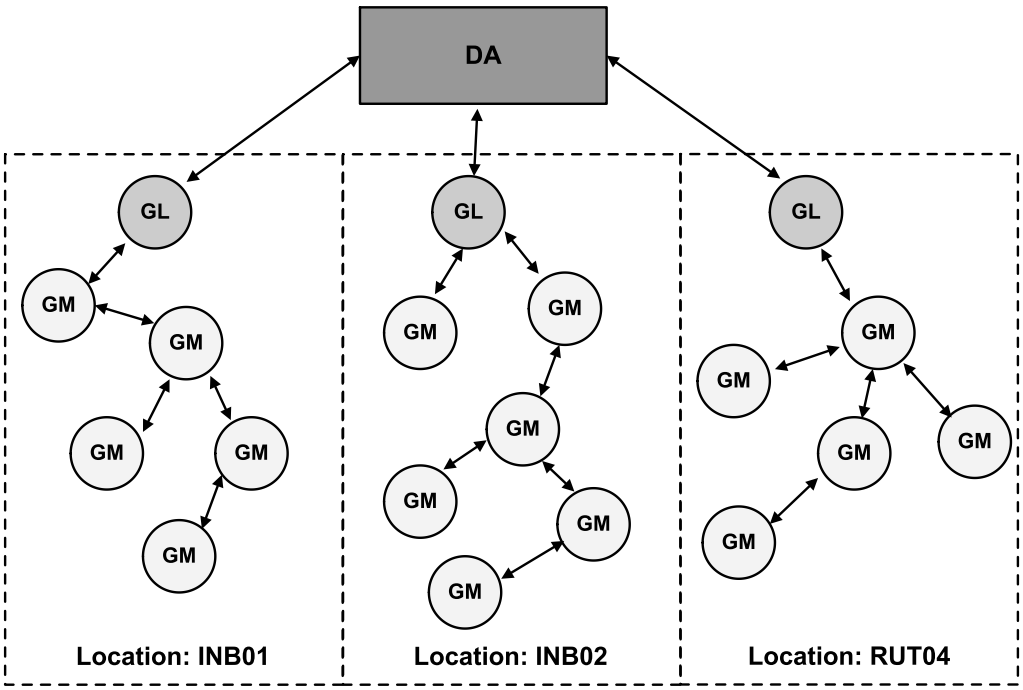


Figure 8.1: New architecture of TRENDY with grouping mechanism

8.4 Grouping process

Context-aware grouping creates an application-level grouping overlay, to keep the load of status maintenance in the specific area of a network and to deal with

group-based requests. Location tags of the nodes are exploited by this mechanism to form groups of nodes. It requires some SAs to offer the functionality of GL by taking additional responsibility to correspond to group formation requests and manage a group. Following is the process of grouping:

1. While registering the nodes, the DA checks whether any node is offering the basic GL functionality (Section 8.3.2). It keeps all identified GLs in a separate location-based GL list. Figure 8.2 presents a sample scenario of this step.

2. The DA periodically analyses its registry for grouping. It selects the location-based GLs for each un-grouped GM. If multiple GLs are available for the same location, the DA will select the appropriate one (Section 8.5), as shown in Figure 8.3. Any ungrouped GM with some level of mobility (Section 5.5) is not considered for grouping.

3. For each newly grouped GM, the DA sends a YGM message with the GM's IP address to the respective GL.

4. The GL confirms the addition of the new GM and sends back the confirmation to the DA, with appended GM's IP address (for validation).

5. The GL periodically checks the newly added GMs, and synchronises them by sending YGL message. This needs an acknowledgement from the GM to finalise the registration.

6. The registered GM then starts sending UPD messages to GL instead of the DA. Figure 8.4 shows the grouping process at this stage.

7. The GL periodically checks the status of its GMs after time window interval, and deletes the record before sending a report of the inactive ones to the DA.

8. The GL keeps track of their remaining battery and after reaching a threshold value can inform the DA to select a new GL. It uses the GLD (Group Leader Done) message (Section 8.6.3.4) for this purpose.

9. If any GL provides extra functionality by implementing more GL resources (Section 8.3.2) then the DA can also use the capabilities of GLs to further optimise the network or to offer new mash-up services.

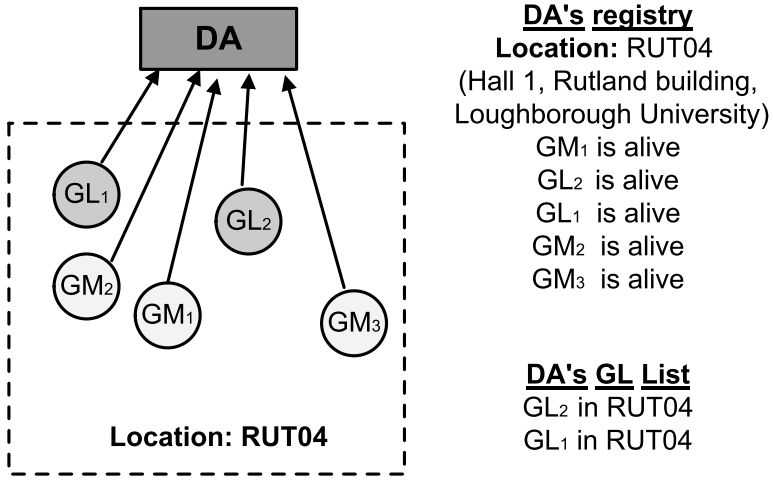


Figure 8.2: Step 1 - Registration and recognition of location and profiles

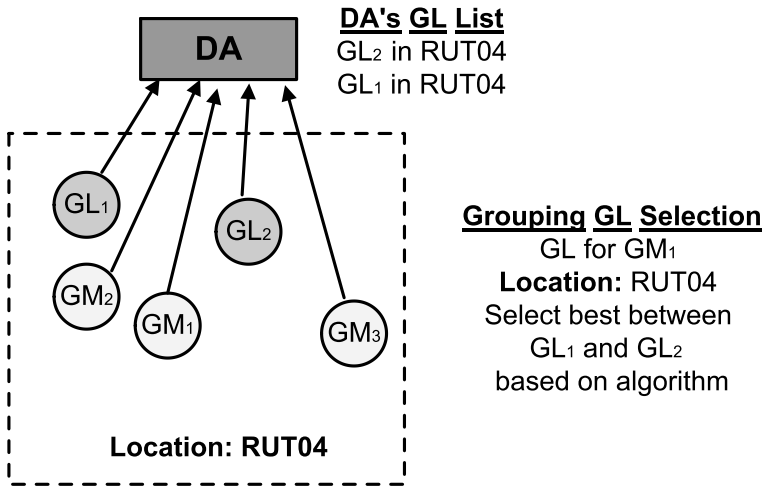


Figure 8.3: Step 2 - The DA's analysis for grouping and GL selection

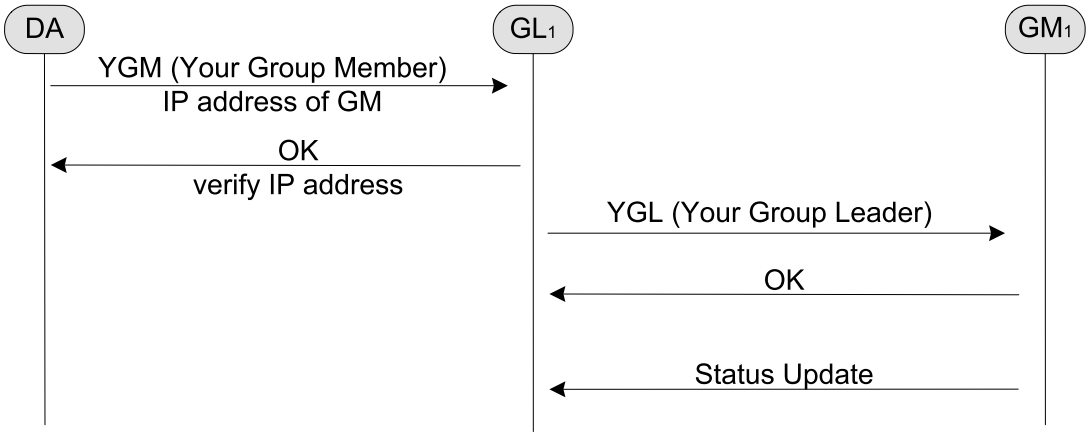


Figure 8.4: Step 3 - Grouping completed by exchanging messages

8.5 Best GL Selection

Context-aware grouping mechanism devises a best GL selection mechanism to select among multiple potential GLs in a location. The DA selects a GL with more GL resources (Section 8.3.2) whenever multiple GLs are available in one location. If in case all GLs have implemented equal number of GL resources, then the DA will prefer the one with better battery source (Section 5.5). If still decision is not made then the DA goes through the list and compares their ranks. These ranks are maintained by the DA and updated every time the registry is analysed for grouping. The equation to calculate rank is:

$$rank = S_T + N_{GM} - f - b$$

Where S_T is serving time of a GL in hours, N_{GM} is the number of GMs supported by a GL, f is the number of failures in response to YGM messages, b is battery consumed. It is important to select a best GL that can serve the job for longer duration, because re-grouping can cause the adverse effect on the performance.

8.6 Message Formats

This section covers the detail of message formats required for context-aware grouping. All messages require an acknowledgement in response.

8.6.1 UPD (Update) for GL registration

The registration of a GM is exactly like a SA's registration (Section 5.11.1). However, the registration of a GL differs with the addition of two new context

variables. The format of UPD messages for a GL is shown in the Figure 8.5.

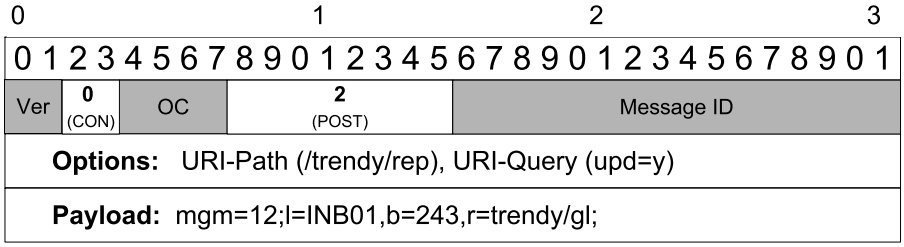


Figure 8.5: GL Registration request

Following is the description of new context attributes for group leaders:

- **Maximum Group Member ("mgm" or "maxgm")= Value (8-bit unsigned integer):** The GL uses this attribute in registration to inform the DA about its capacity in terms of the number of GMs supported.
- **Group Member ("gm" or "currentgm") = Value (8-bit unsigned integer):** This attribute is used by the GL to inform the DA about the number of its registered GMs. This value is used by the DA for validation of GL's record.

8.6.2 Grouping Messages

The context-aware grouping introduces Your Group Member (YGM), Remove Group Member (RGM) and Your Group Leader (YGL) messages to enable the grouping technique.

8.6.2.1 YGM (Your Group member)

Sender: DA

Receiver: GL

Purpose: The DA uses this message to allocate a group members to a GL.

Format: The format of a YGM message consists of: `trendy/gl` URI-path and `ygm=y` URI-QUERY. The IP address of the GM is also piggybacked, and the POST method is used. Figure 8.6 shows the format of a YGM request message.

Actions: When the GL receives a YGM message, it adds a new record of GM in its group list, by using the IP address from the payload. The ACK message from GL, as specified in Figure 8.7, carries the IP address of the newly added

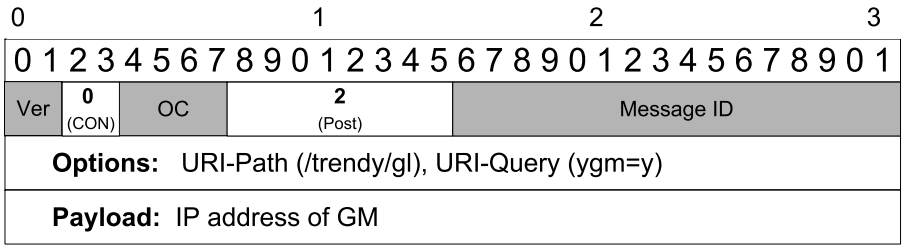


Figure 8.6: YGM: Your Group member request

GM. The success code 2.04 (equivalent to HTTP 204) that specifies that the GM has been added by the GL.

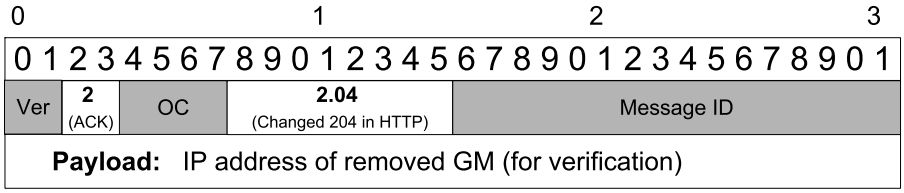


Figure 8.7: YGM-ACK: Your Group member response

8.6.2.2 RGM (Remove Group member)

Sender: DA

Receiver: GL

Purpose: The DA uses RGM messages to remove group members from GL's list.

Format: To remove a single GM, the DA uses `trendy/gl` URI-path and `rgm=1` as a URI-QUERY, 4 (Delete) method code and IP address of GM in the payload. The message format is shown in Figure 8.8. However, the DA can ask a GL to reset and remove all GMs by setting URI-QUERY to `rgm=2`, as shown in Figure 8.9).

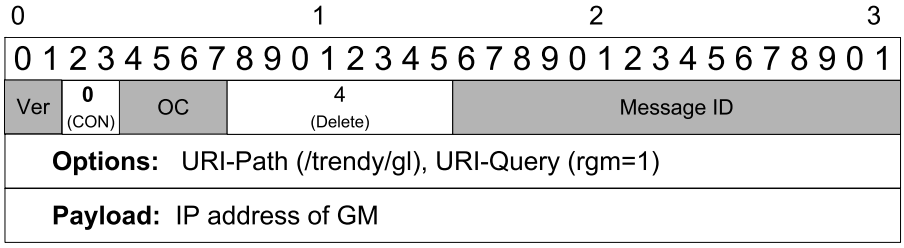


Figure 8.8: RGM request: Remove Group member request

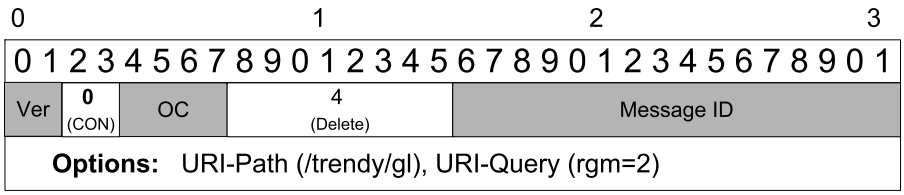


Figure 8.9: RGM request: Remove all Group member request

Actions: If the query variable is 1, the GL uses the IP address to remove the GM. It then sends the ACK response, as specified in figure 8.10 that carries the IP address of the removed GM. In case of query variable's value more than 1, the GL reset its GM's list and format acknowledgement as specified in Figure 8.11. In both cases, the success code 2.02 (equivalent to HTTP 202) or respective error code is set in acknowledgement.

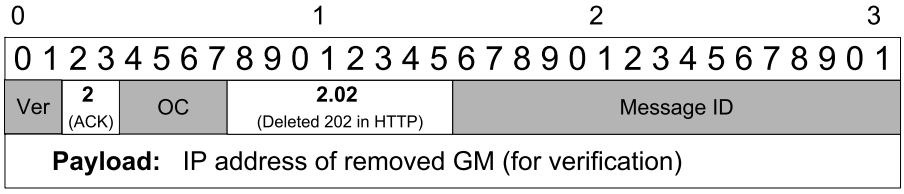


Figure 8.10: RGM Response after removal of one GM

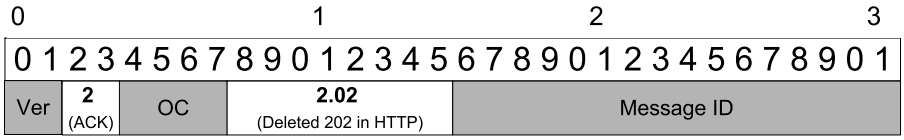


Figure 8.11: RGM Response after removal of all GMs

8.6.2.3 YGL (Your Group Leader)

Sender: GL

Receiver: GM

Purpose: The GL announces its role to a GM. This message orders a GM to start sending the status updates to the GL.

Format: The format uses `trendy/rep` URI-path and `ygl=y` as a URI-QUERY, POST method code. Figure 8.12 shows the format of a YGL message.

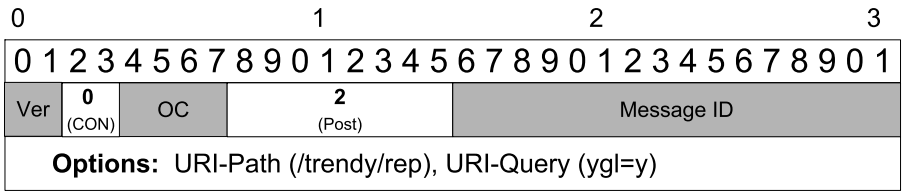


Figure 8.12: YGL: Your Group Leader request

Actions: The GM saves the sender's IP address of the message to send future status updates, and sends a simple ACK message with the corresponding status code. In case of success, 2.04 response code (equivalent to HTTP 204) is sent, as shown in Figure 8.13.

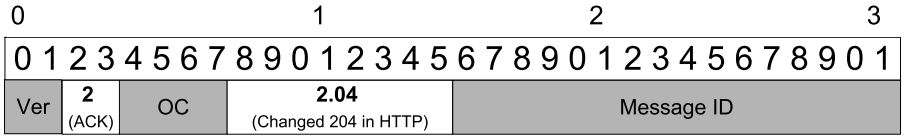


Figure 8.13: YGL: Your Group Leader response

8.6.3 Reporting Messages

All GLs and GMs use UPD (Section 5.11.2) and Some Service Changed (SSC) messages for status maintenance. GLs also use Not Reported (NRP) message to send report of their unresponsive GMs and Group Leader Done (GLD) for load balancing request to the DA.

8.6.3.1 UPD (Update) message for Status maintenance

It follows the same format and actions explained in Section 5.11.2. However, the GL resets its GM list when it receives a reset response from the DA.

8.6.3.2 NRP (Not reported)

Sender: GL

Receiver: DA

Purpose: The GL informs the DA that a GM is unavailable.

Format: The format of a NRP request message is shown in Figure 8.14, where method code 3 is specified for a DELETE request. NRP message specifies the `trendy/server` in URI-PATH option and unavailable GM's IP address in the

payload, to request the registry resource to update the service information against the IP address of a node.

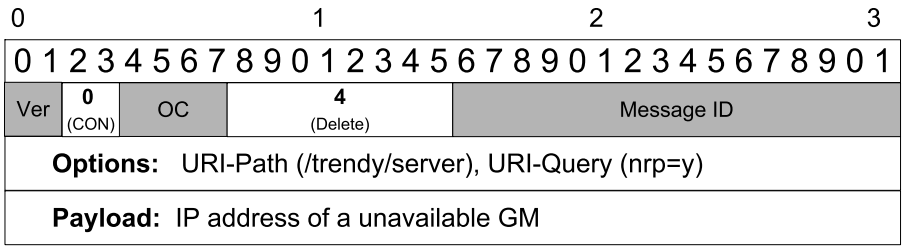


Figure 8.14: NRP (Not reported Message) request

Actions: The DA deletes the entry for a SA by using the IP address received in request's payload. It then sends back an ACK in response with 2.02 (HTTP 202) response code or corresponding error message, as shown in Figure 8.15.

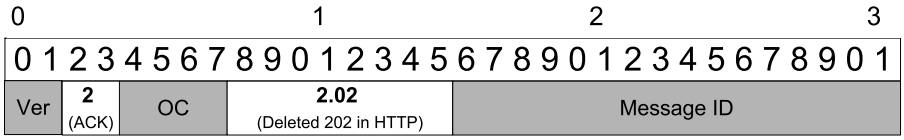


Figure 8.15: NRP (Not reported Message) response

8.6.3.3 SSC (Some Service Changed)

Sender: SA (GM or GL)

Receiver: DA or GL (if GL is also acting as a local DA)

Purpose: A SA informs its DA or GL (if grouped) that some resource availability is changed.

Format: This message contains URI-PATH option that is set to `trendy/server` and `ssc=1` URI-QUERY with DELETE (to delete a service) or POST (to add a service) method. The URL of a service, which is needed to be added or deleted, is included in the payload. The message format is given in Figure 8.16.

If a SA wants to update the DA about multiple services, it uses format shown in Figure 8.17. In this case, the URI-QUERY option is set to `ssc=2` and service descriptions of all resources is appended to the payload.

Actions: If the DA receives the `ssc` URI-QUERY set at 1, it reads the payload for resource's URL. The request's method code is used to perform the requested

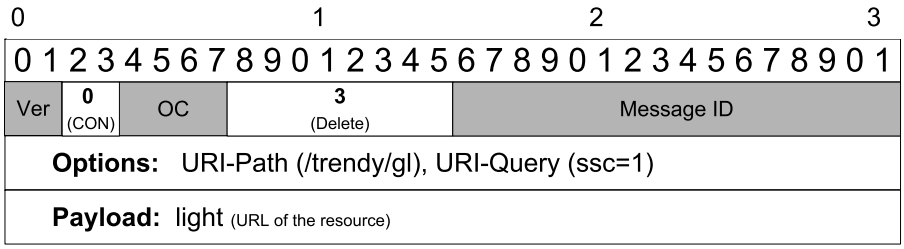


Figure 8.16: SSC Some Service Changed request for one resource change

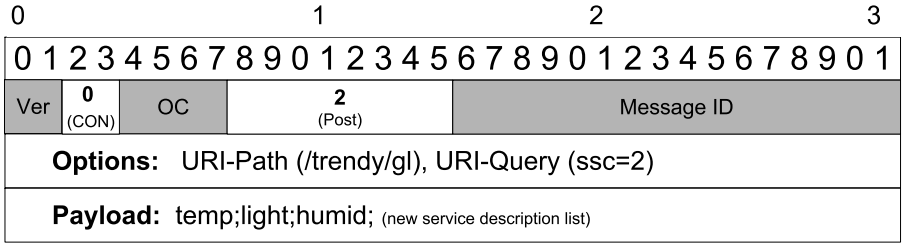


Figure 8.17: SSC Some Service Changed request for all resources

operation. Otherwise, if it is more than one, then the DA overwrites the node's service description attached to the payload. In both cases, the response shown in Figure 8.18 on success. In case of failure, the respective error code is used.

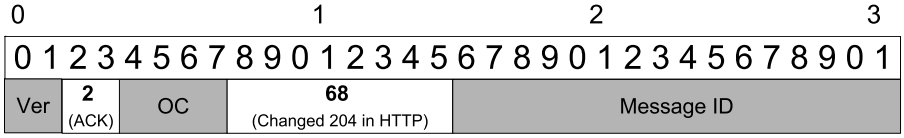


Figure 8.18: SSC Some Service Changed response

8.6.3.4 GLD (Group Leader Done)

Sender: GL

Receiver: DA

Purpose: The GL informs the DA that its battery is depleting. The DA can remove GMs from a GL to regroup them; however grouping also offers the load balancing on the GL's demand. The constrained GL can specify a threshold value for battery level. When a battery is depleted and crosses the threshold value, an event triggers a GLD message to notify the DA for load balancing.

Format: The format of a GLD message is shown in Figure 8.19. The request is specified by addressing the reporting resource at DA by `trendy/server` URI path and `gld=x` URI query, where x defines the number of times a GLD message is already sent by this GL to the DA.

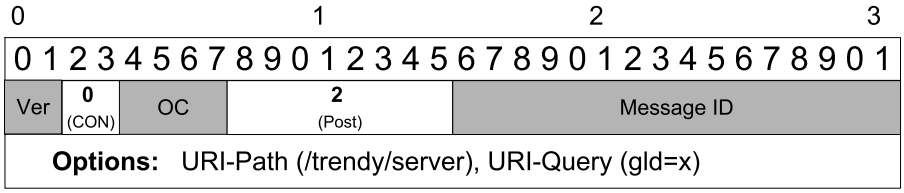


Figure 8.19: GLD (Group Leader Done) request

Actions: After receiving a GLD message, DA sends an acknowledgement message. Figure 8.20 shows the acknowledgement where the DA asks the GL to wait for reset command. The response code is set to 2.03 (HTTP 203) that means request was valid but nothing is modified yet. The DA then starts the re-selection procedure for a new GL. The previous GL will keep performing the role of a GL, until it receives a RGM message (Section 8.6.2.2). However, the DA can ask GL to reset immediately by sending the acknowledgement shown in Figure 8.21. The message type set to 3 which means 'RESET', and the GL immediately resets.

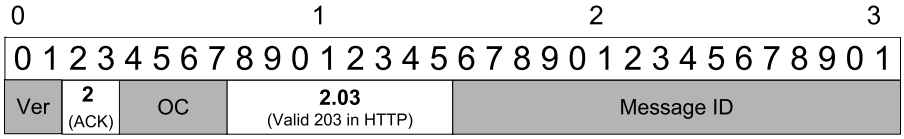


Figure 8.20: GLD (Group Leader Done) response: DA asked to wait

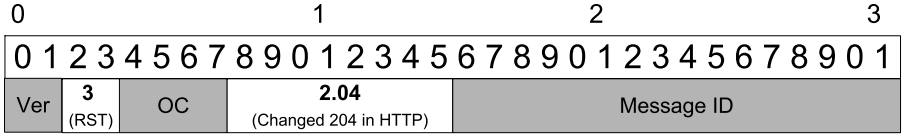


Figure 8.21: Reset response: DA asked to reset

8.6.4 Discovery Messages

The grouping mechanism introduces a base for service composition, where a GL can pass a query to its GMs and can process responses as well. This section defines

a query forwarding process using Forward Query (FWD) message for GLs with `Forwarder` resource (Section 8.3.2).

8.6.4.1 FWD (Forward Query)

Sender: DA

Receiver: GL

Purpose: The DA can use the group information to offer higher level services. This message is used by the DA to forward a UA query to all GLs in a location, when it receives a query for an offered higher level location-based service. The UA can request different operations, e.g., PUT method to switch the lights ON or OFF in location X. The GLs then disseminate the command to all their GMs.

Format: The DA formulate a request by specifying corresponding method (GET, POST, PUT or DELETE) and specify the rest of the message as shown in Figure 8.22.

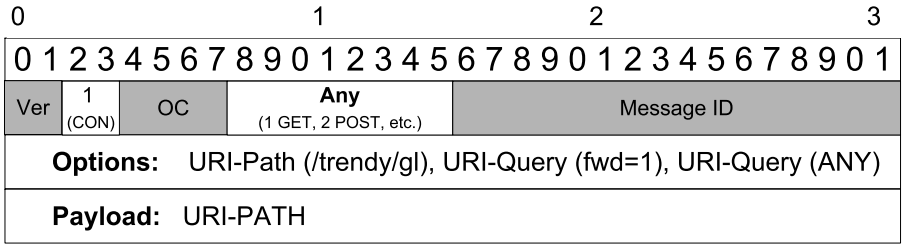


Figure 8.22: FWD: request to a group

Actions: The GL sends an acknowledgement message to the DA. Then it formulates the corresponding query message to forward. The *GET* method is not allowed in a forwarded request. The message is made non-confirmable (commands, e.g., Switch off the lights, etc.) as presented in Figure 8.23.

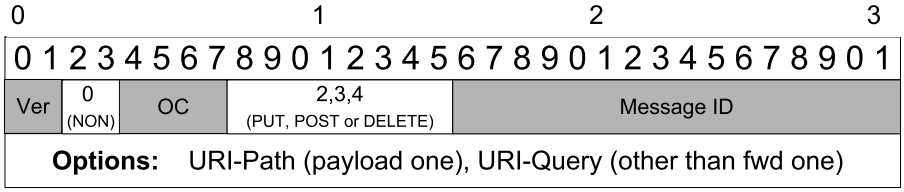


Figure 8.23: FWD: Non confirmable request forwarded by the GL to all GMs

8.7 Experiments: Group Scalability

This section covers the detail of experiments, including scenarios, experimental setup and discussion of the results performed to analyse the grouping impact on different metrics with the increasing size of a group.

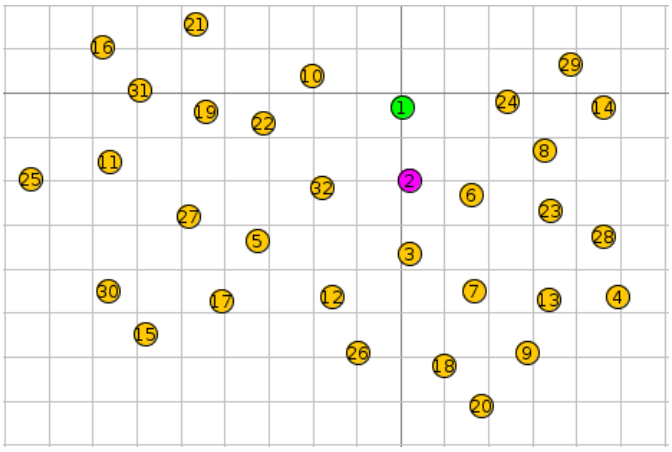
8.7.1 Introduction

Grouping is designed to achieve two goals: localise status maintenance to reduce the number of hops for a packet and to avoid a bottleneck, and to provide a base for service composition. However, the application-level topology formed by grouping can be incongruent to the routing topology and can even increase the traffic overhead. This can decrease the nework lifetime when a homogeneous node is acting as a GL. The experiments are performed using emulated hardware nodes to analyse the impact of RPL, employed routing protocol, and to find the grouping overhead in different RPL DODAGs. The number of nodes is varied, and grouping effect is analysed against scenarios with no grouping. Following scenarios are used in experiments:

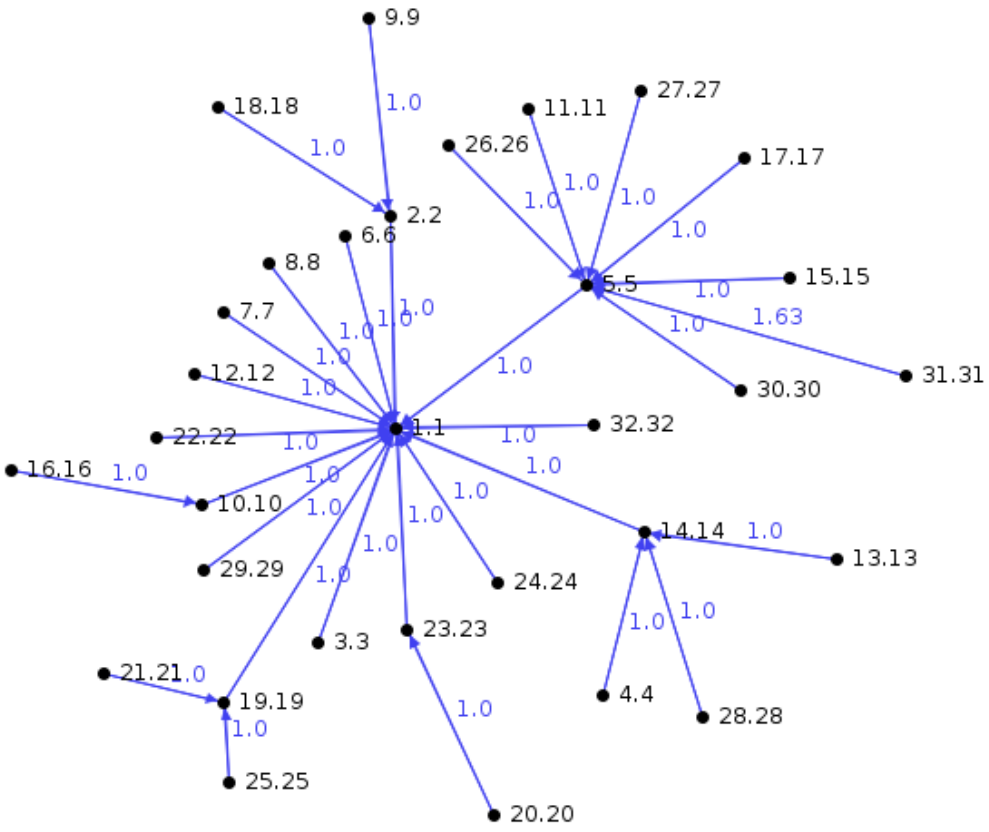
Case 1-4 - Basic TRENDY without grouping: This scenario only enables the basic functionality of the proposed solution where a UA gets a single best matched service information. The number of nodes are varied between 6, 11, 21 and 31 to analyse the network performance.

Case 5-8 - Basic TRENDY with 1 GL: These scenarios are the extension of case 1-4 by assigning the role GL to one of the nodes.

The simulations were configured with the same settings defined in Section 6.3 with the changes given in Table 8.1. The experiments for the above scenarios are conducted using two physical topologies given in Figures 8.24 and 8.25 with corresponding RPL's DODAGs. Both topologies are similar except that second one is configured in such a way that only one node is placed in the vicinity of the DODAG root by following the analogies defined in Section 6.3.1. In case of TRENDY grouping, one physical topology translates into three different topologies: physical topology, RPL topology and application-level TRENDY tree topology formed by grouping.

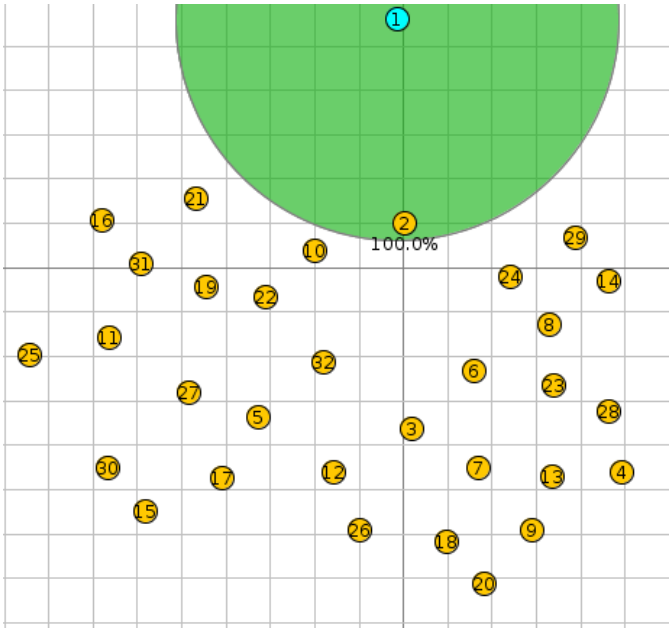


(a) Physical Topology No.1

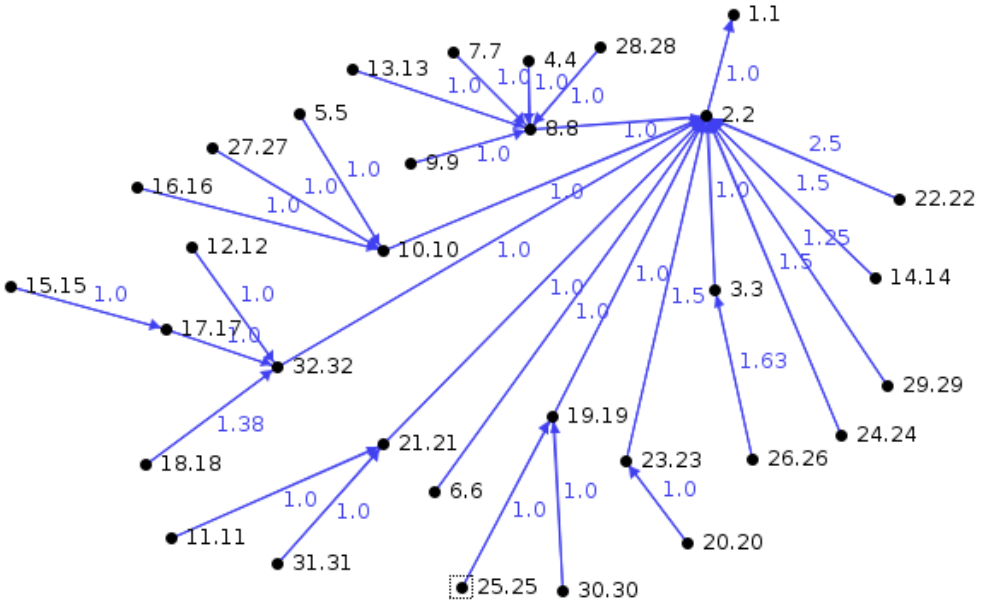


(b) RPL tree for Topology No.1

Figure 8.24: Different views of Topology 1 to evaluate grouping scalability



(a) Physical Topology No.2



(b) RPL tree for Topology No.2

Figure 8.25: Different views of Topology 2 to evaluate grouping scalability

Table 8.1: Configurations for the experiments of grouping scalability

Total nodes	6, 11, 21 and 31 nodes + 1 border router
Area	$150m \times 100m$
Physical topologies	1 and 2 (Explained in this section)
Number of locations	1 location for all nodes
Number of cases	$4 \times 2 = 8$
Total Service Discovery queries	100
Total Service Invocation queries	100
First UA Query	At 600 seconds
DA time window	5 minutes = 300 seconds
Number of time windows	30
Simulation Duration	$300 \times 30 = 9000$ seconds

This section continues with the discussion of results generated by the simulations.

8.7.2 Control packet overhead

Figure 8.26 shows the detail of application-level control packet overhead for all scenarios using topology 1 and 2. It is evident that grouping has only marginally increased the traffic at the application layer. However, Figure 8.27 provides a deeper insight into the actual overall traffic in the whole network. This figure explains the impact of RPL's DODAG in diminishing the benefit of grouping in topology 1, where more nodes have direct links to the root. In this case, all nodes are grouped with the GL (node 2), which has only two leaf nodes that have chosen it as a parent in the DODAG. This will cause all the status maintenance traffic from the nodes to reach at the GL via the root. Consequently, packets need to traverse more hops and the number of packets also increases. On the contrary, topology 2 represents a case on other extreme where the DA has only GL as its neighbour, so the formed DODAG has only GL at its level one. This forces all traffic to bypass the GL and as a result all packets traverse the GL path to reach at the DA. It is evident that this actually increases traffic considerably; however, in this case grouping improves the traffic overhead of the network.

Both physical topologies are similar with only the mentioned difference (placement of GL and DA) that's why the number of RPL and other messages remain same. Figure 8.28 presents the detailed view of CoAP packets. Topology 1 has fewer service invocation messages compared to topology 2 because of closer SAs are available near its root. Topology 2 with grouping incurs fewer numbers of packets compared to topology 1 due to localise traffic in it. This is important to

mention that the efficiency will increase further and will become more visible in the long run because of localised traffic. In conclusion, the efficiency of grouping depends upon the formation of the routing tree and better GL selection, which can serve for a long run. This can be achieved by manipulating the RPL by forcing a resource-rich node to be at the top of a DODAG and using the designed GL selection mechanism to take an informed decision before assigning the GMs to a GL.

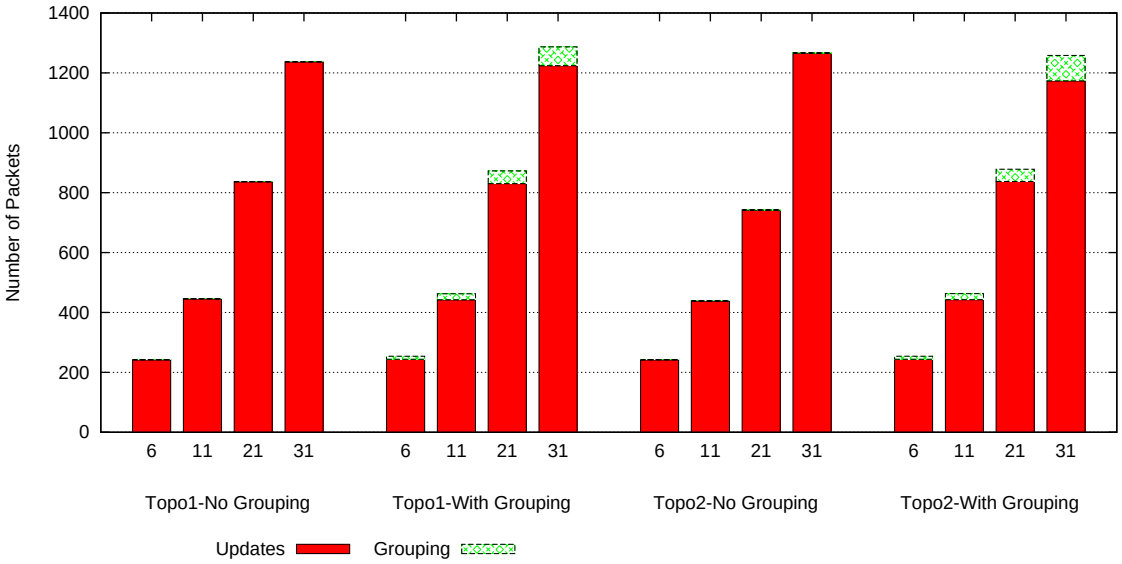


Figure 8.26: Detailed aggregated Control packet overhead for all scenarios for topologies 1 and 2

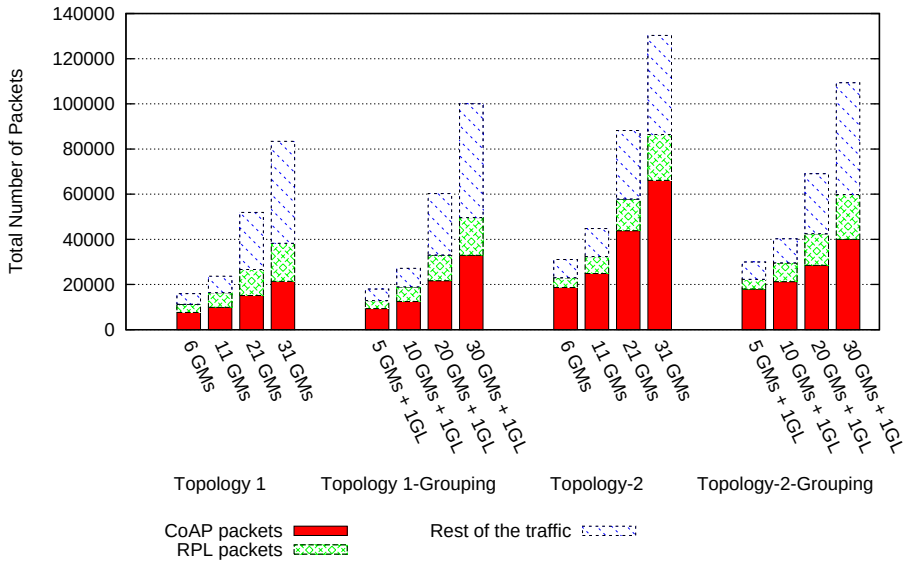


Figure 8.27: Overall network traffic for all scenarios for topologies 1, and 2

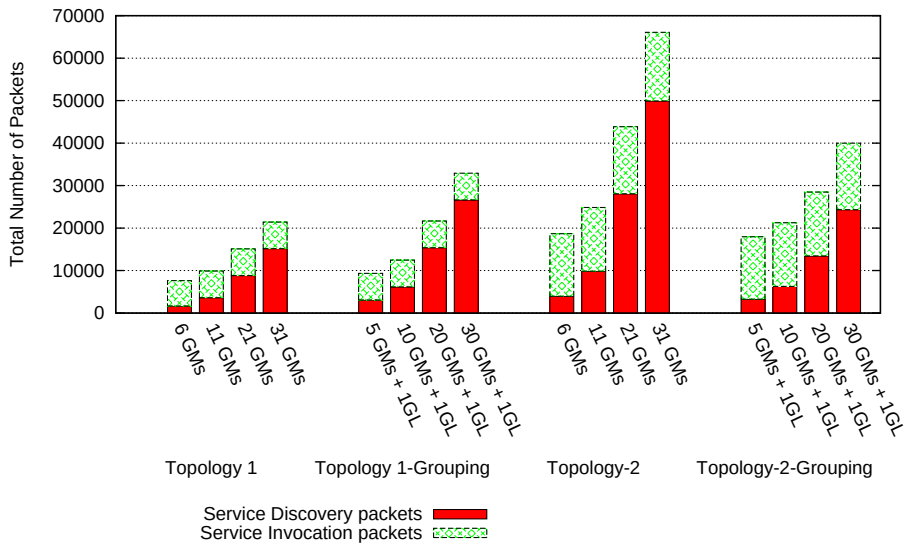


Figure 8.28: Overall CoAP traffic for all scenarios for topologies 1 and 2

8.7.3 Energy Consumption and network lifetime

Figure 8.29 shows that the nodes in grouping scenarios consume more energy in all scenarios for topology 1. Moreover, the energy consumption increases considerably for the GL when 30 GMs are assigned to it. On the other hand, topology 2 demonstrates that grouping improves the energy efficiency in its scenarios, as shown in Figure 8.30. This is because the GL node is in the way to the DA for all nodes. However, this topology suffers a drawback as well that it causes the problem where all traffic even service invocation messages needed to pass through a particular node in both cases, which increases the energy consumption considerably for a single node at DODAG's level 1. Figure 8.31 and Table 8.2 shed light on this issue. It is evident from both figure and table that the energy consumption for the top node is doubled or in other words, network lifetime (our definition) reduced to half when topology 2 is used in both cases. However, grouping increases the energy efficiency and network lifetime for topology 2. Nevertheless, the grouping is not expensive in both topologies and even increases efficiency in some cases. Furthermore, if a resource-rich GL is available with a better power source, then the network lifetime issue in topology 2 will be eliminated and a preconfigured GL node with RPL configurations can provide the base for service composition while optimising the network lifetime as well.

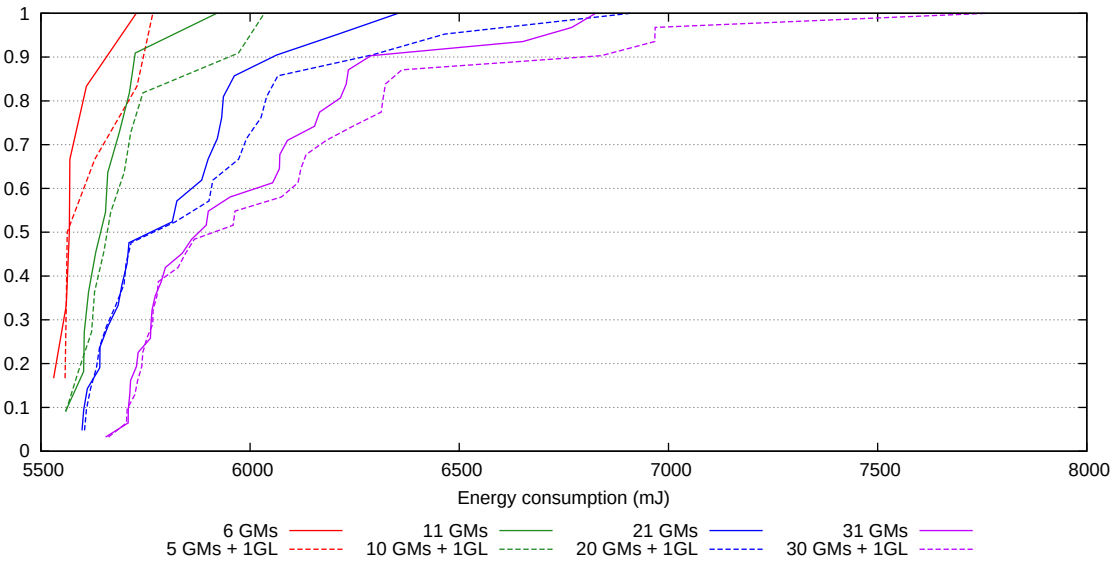


Figure 8.29: CDF graph of energy consumption per node for 35 nodes for topology 1

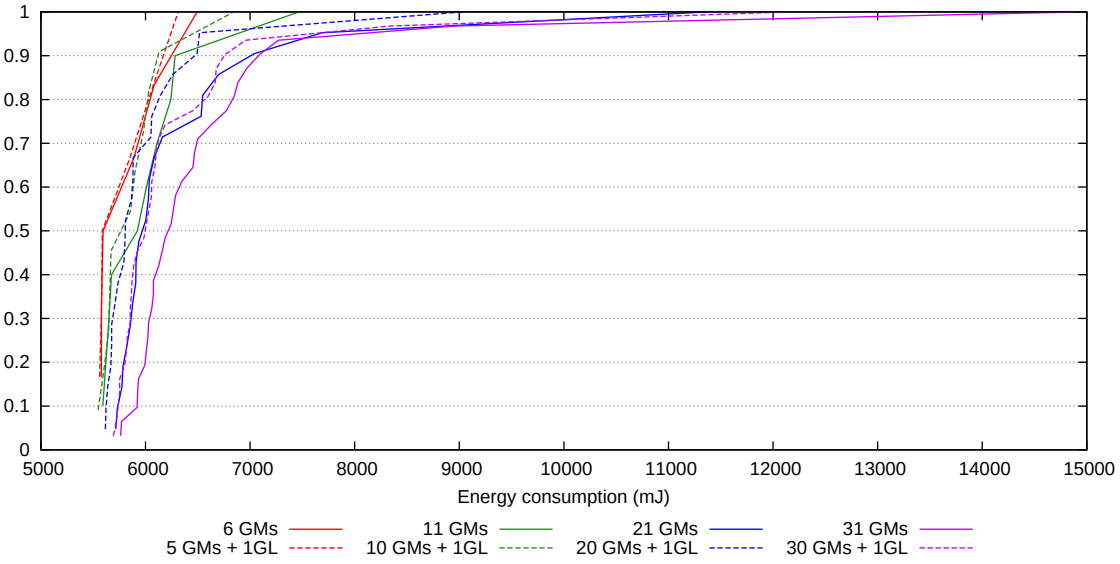


Figure 8.30: CDF graph of energy consumption per node for 35 nodes for topology 2

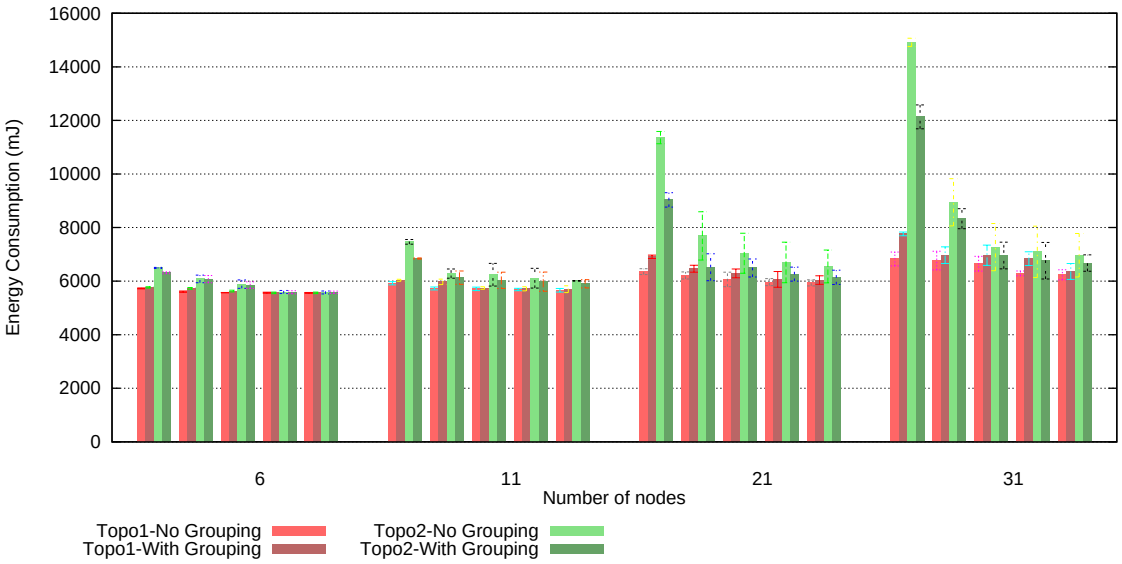


Figure 8.31: Top five nodes in terms of energy consumption for topologies 1 and 2

Table 8.2: Network lifetime approximation for different number of nodes for topologies 1 and 2

Network Lifetime (days)				
Total	**Topology #1**		**Topology #2**	
nodes	**No-GL**	**With-GL**	**No-GL**	**With-GL**
6	490	485	430	445
11	475	465	375	410
21	440	405	245	311
31	410	360	180	230

8.7.4 Scalability factor: Packets to the DA

Grouping aims to increase scalability while keeping the number of application-level messages towards the DA to the minimum. Figure 8.32 shows that the grouping has substantially decreased the number of application-level packets at the DA by assigning the task to the GL. However, the actual traffic in the whole network depends upon the RPL's DODAG. One application-level message can generate several packets and in the worst case, can go at the root first to reach at the GL. This process can cause the network in terms of high-traffic overhead, delays and energy consumption. Figure 8.33 depicts the detail of the total number of packets sent or received at the DA in all scenarios at all layers. It is evident in the figure that grouping has reduced the number of packets at the DA in all cases, but topology 1 still incurs high number of packets for scenarios with 21 and 31

nodes, because messages need to travel extra hops to reach at the GL. Overall, the grouping reduces the traffic towards a single point of the network in both topologies.

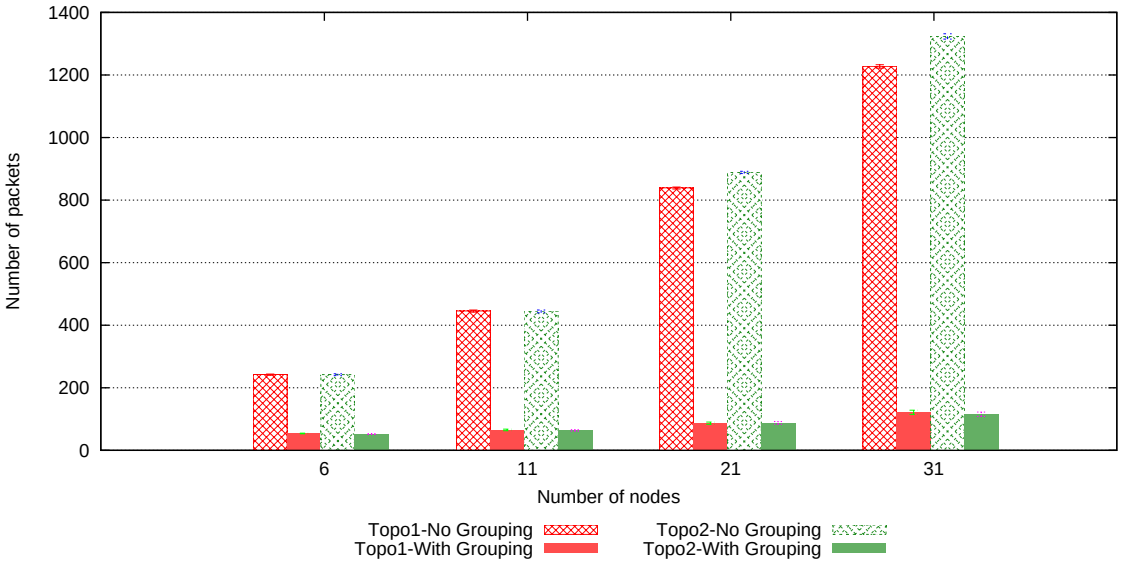


Figure 8.32: Application-level packets at the DA for all scenarios for topologies 1 and 2

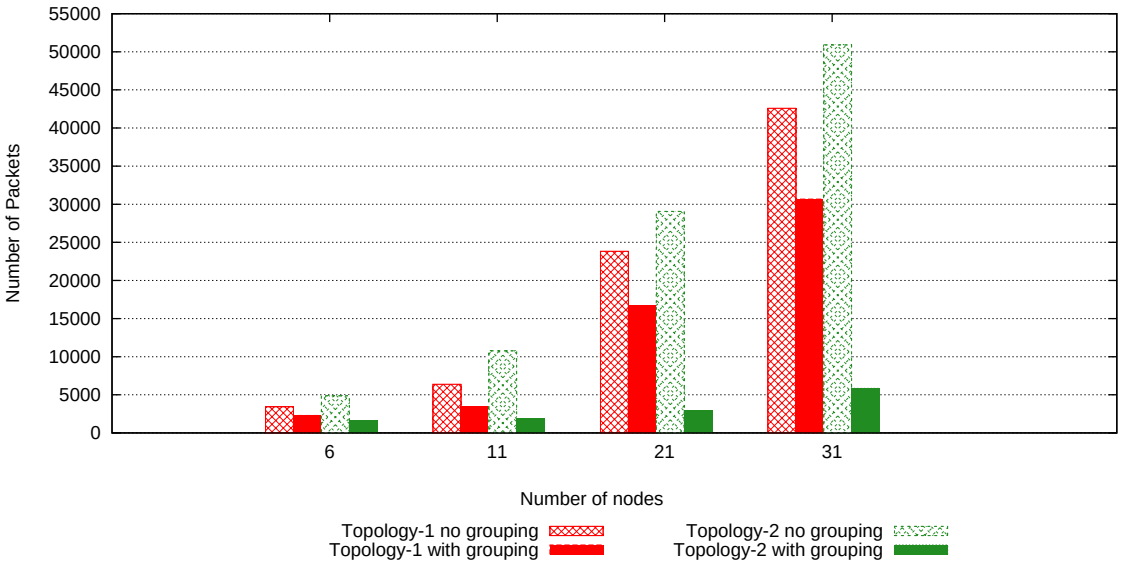


Figure 8.33: Total traffic at the DA over the time for all scenarios

8.7.5 Service Invocation Delay

The grouping mechanism is not focused on improving service invocation delay. Consequently, it is evident from Figure 8.34 that the service invocation delay

remains almost same for all scenarios. However, a slight improvement can be noticed for scenarios of 21 and 31 nodes using topology 2. This benefit comes as a repercussion of localised traffic, which improves the delay for queries as illustrated in Figure 8.35, but it becomes only evident in a large network. Overall, topology 1 performed better in terms of average service invocation delay because it has SAs available one-hop away from the DA.

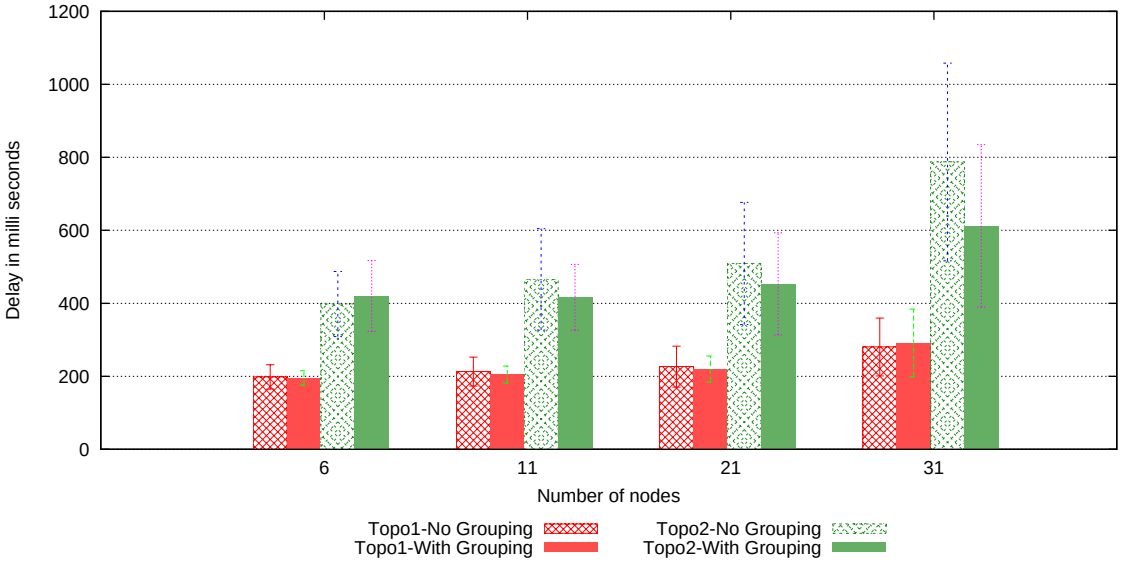


Figure 8.34: Average Service Invocation delay in all cases for both topologies

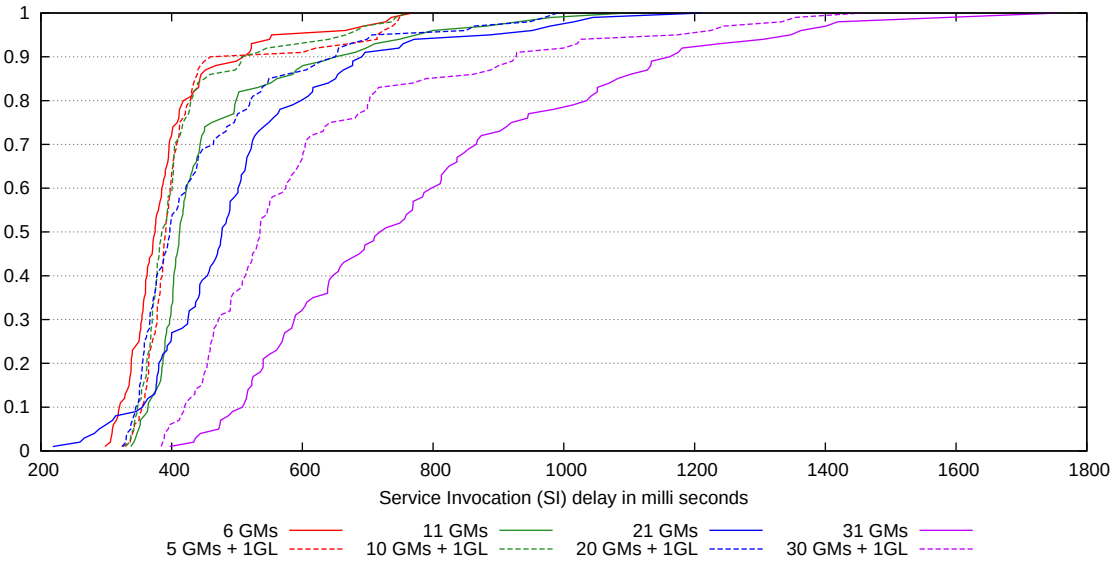


Figure 8.35: CDF of Service Invocation delay in all cases for topology 2

8.7.6 Reliability and Accuracy

Grouping makes the GL responsible for informing about the missing nodes, so chance of obsolete registry entries is eliminated. In experiments all service invocation requests and responses are validated and were found accurate, so all the communicated information was correct and reliable.

8.8 Experiments: Grouping with different groups

This section covers the detail of experiments, including scenarios, experimental set-up and discussion of the results performed to evaluate the impact of grouping and creation of multiple groups in networks with different topologies.

8.8.1 Introduction

The scalability testing of grouping is done in Section 8.7 where experiments were done only for one group of nodes with a GL. This section scrutinise the grouping mechanism further by increasing the number of groups in different topologies. Following scenarios are used in experiments.

Case 1 - Basic TRENDY without grouping: This scenario only enables the basic functionality of the proposed solution where a UA gets a single best matched service information.

Case 2 - Basic TRENDY with 5 GL: This case has grouping mechanism enabled with 5 GLs that are placed one in each of the 5 locations, on top of case 1's functionality. In all topologies, node 2-6 (No. 1 is the border router) were assigned the role of GLs for their groups.

Case 3- GL failure scenario with 6 GLs: This scenario has 6 GLs (node 12 as well), where one of them fails during the simulation (after 1000 seconds) and other GL is reselected by the DA. All topologies are same like case 2, but topology 1 is slightly changed for this case and position of node 2 is swapped with node 7 to maintain the DODAG tree after the GL's failure.

The simulations were configured with the same settings defined in Section 6.3 with the changes given in Table 8.3. In the case of TRENDY with grouping, one physical topology translates into three different topologies: physical topology, RPL topology and application-level TRENDY tree topology formed after grouping. The experiments for the above scenarios are conducted using three physical topologies

explained in Section 6.3.1. The GL nodes are employed one hop away from the DA and only perform the group formation and status maintenance responsibilities. In addition, all the GLs do not offer any other resource to sense the environment. Therefore, all scenarios have 30 nodes that offer resources, to make a better comparison.

Table 8.3: Configurations for the experiments of Grouping

Total nodes	35 nodes + 1 border router
Physical topologies	1, 2 and 3 (Section 6.3.1)
Number of locations	5
Number of SAs hosting services	6 in each location
Number of cases	6
Total Service Discovery queries	1000
Total Service Invocation queries	1000
First UA Query	At 600 seconds
DA time window	5 minutes = 300 seconds
Number of time windows	30
Simulation Duration	$300 \times 30 = 9000$ seconds

This section continues with the results generated by the simulations.

8.8.2 Control packet overhead

Figure 8.36 shows the aggregated application-level control overhead for all 35 nodes for cases 1-2 and 34 nodes in the case 3, because one node dies in that scenario. The grouping scenarios have slightly more packets because of the extra communication overhead for grouping (YGM and YGL) messages. It can be noticed that case 3 still has the same overhead as the case 2, because it's just showing the control overhead for 34 nodes. The figure also emphasises that the grouping overhead is marginal. The grouping overhead can increase in those cases where load balancing is done more often; however, TRENDY reduces this risk by utilising a GL selection algorithm while selecting a GL for a group.

The overall traffic in the network at all layers is shown in Figure 8.37 that explains that the density of a network plays a key role in network traffic when grouping is enabled. The network analysis includes the runs with the different number of GLs to demonstrate the trend of network's reaction when the number of GLs is increased. The networks with topology 1 and 2 produce a similar trend where the overall traffic actually decreases when GLs are employed, because of the topology configurations. On the contrary, the traffic in topology 3 increases with the employment of each GL, because of its high density where more nodes

are at the level one of the DODAG as no preference is made during its formation. Therefore, most of the packets travel upward towards the root and then downwards towards the GL, which not only increase the number of hops but also more delay is added because of the high traffic at the DA. However, the traffic in all the topologies increases considerably in the case of a GL failure, where the traffic at the layers below RPL experienced a heavy load. This behaviour is the result of the increase in neighbour discovery packets that are communicated more frequently when a node becomes unreachable. Figure 8.38 explains this behaviour by showing the neighbour discovery traffic for first ninty minutes in all topologies, where the high spikes after 20 minutes represent the neighbour discovery reaction to the unavailability of a node. Topology 3 performs worst in this case because of its density.

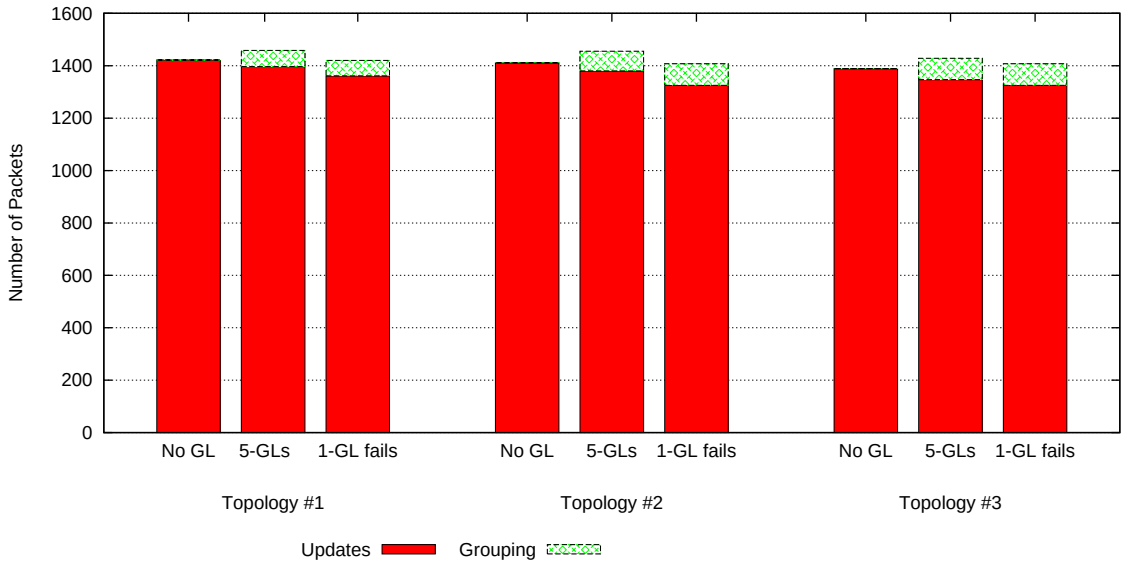


Figure 8.36: Detailed aggregated Control packet overhead for all scenarios (case-7 with 34 nodes and rest of the cases with 35 nodes)

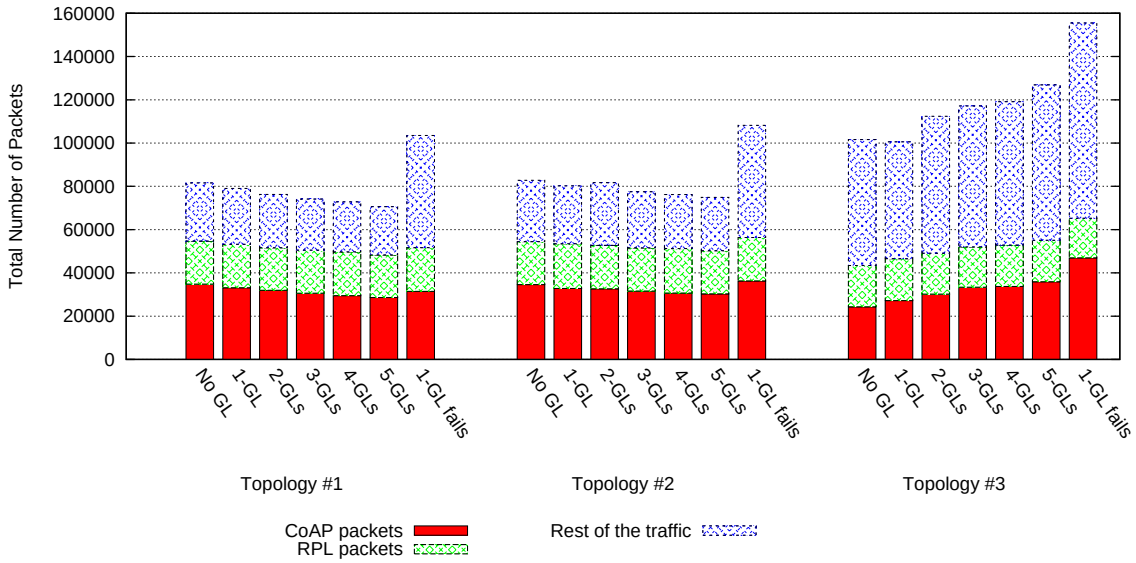


Figure 8.37: Overall network traffic in all scenarios for topologies 1, 2 and 3

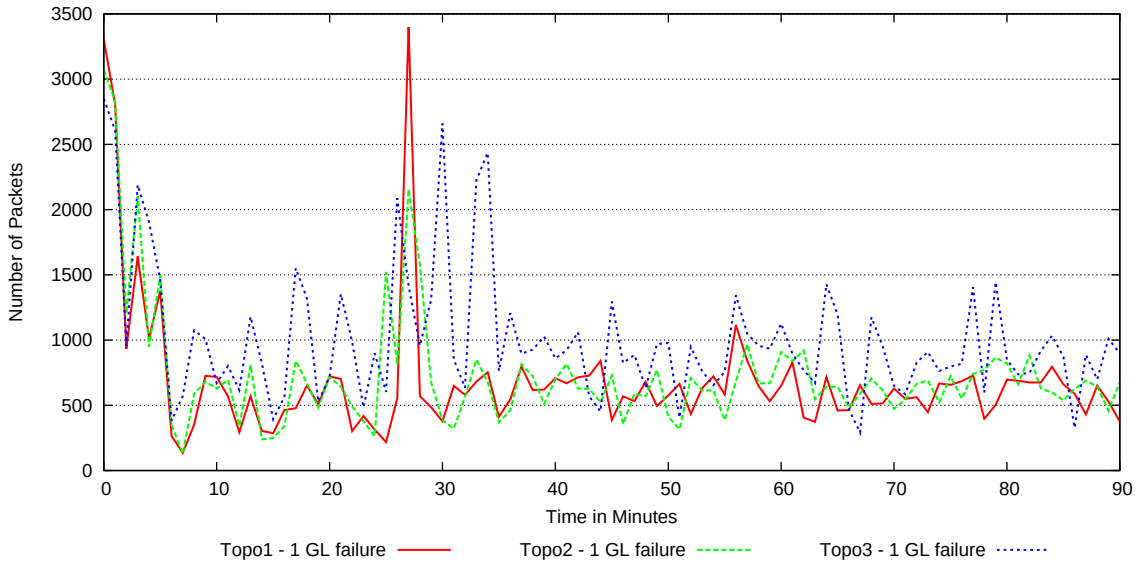


Figure 8.38: GL failure case: Neighbour discovery traffic in first 90 minutes for topologies 1, 2 and 3

8.8.3 Energy Consumption and network lifetime

Figure 8.39 shows that the grouping mechanism has not affected the energy consumption for the node significantly for all topologies. The worst scenario for all topologies remains the GL failure case where re-grouping is done. This scenario explains the benefit of choosing a best GL, which can serve for a longer period of time to improve the energy consumption. However, topology 1 has suffered

less compared to other topologies, because of its RPL formation. Furthermore, Figure 8.40 presents the top five nodes in energy consumption, which emphasises the burden of re-grouping on a single node. Moreover, both topologies 2 and 3 have different nodes topped the energy consumption in 10 iterations because of the random seed values. This is the result of the overlapping radios in a dense environment, and high density topology 3 has more randomness in this perspective.

Table 8.4 uses the energy consumption values of the top most node in energy consumption to estimate the network lifetime by projecting that linearly using simulation time. It is evident from the table that grouping increases the network lifetime only for topologies 1 and 2. In summary, grouping only causes extra energy burden on one GL node (top in energy consumption) and rest of the nodes actually save energy in a network. This issue can be mitigated by employing a GL with a better energy source.

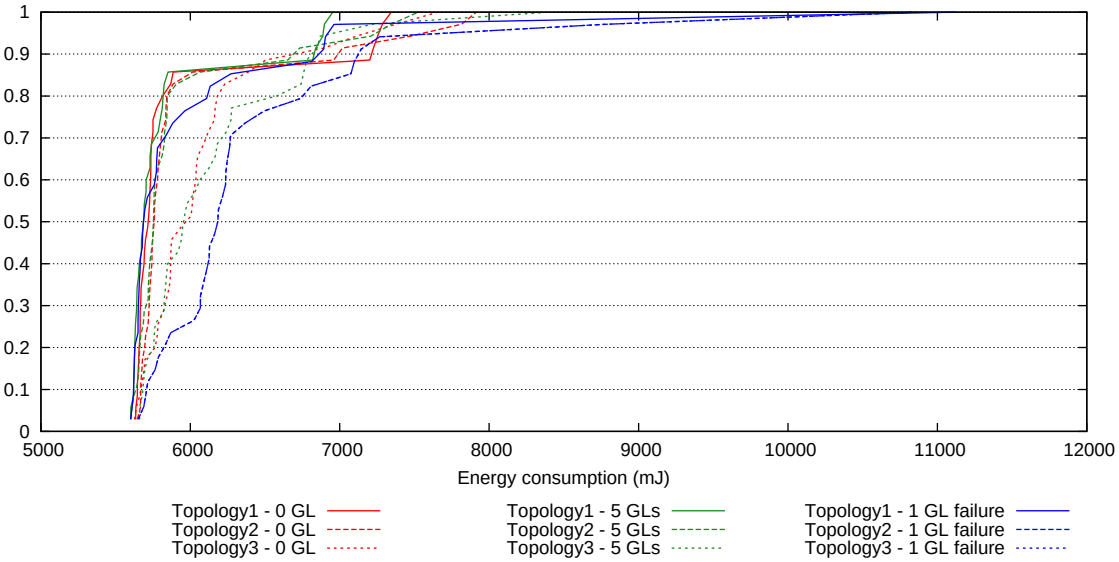


Figure 8.39: CDF graph of energy consumption per node for 35 nodes

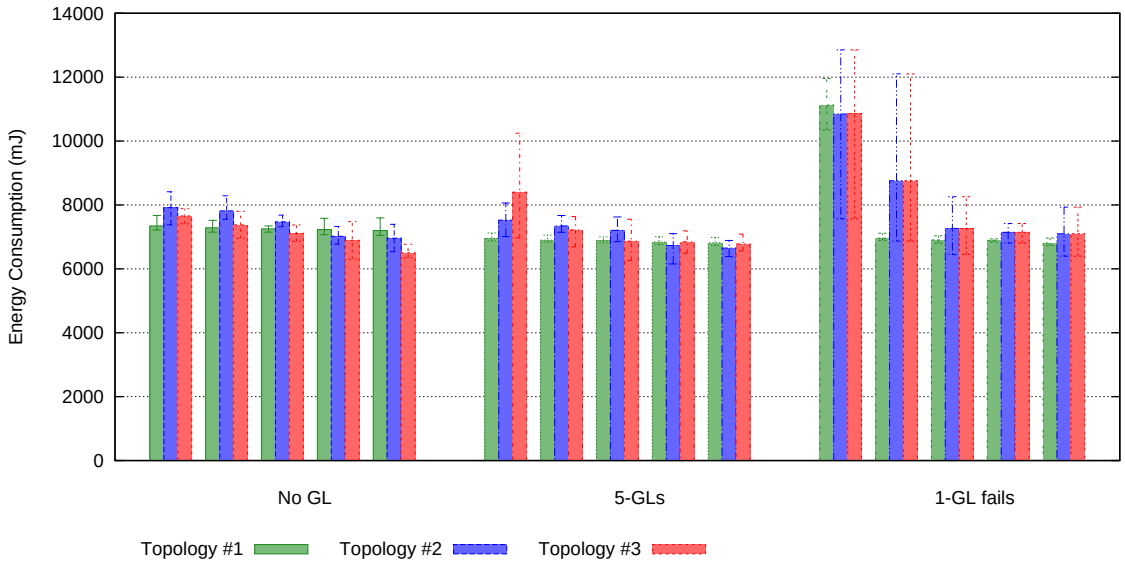


Figure 8.40: Top five nodes in terms of energy consumption for topologies 1, 2 and 3

Table 8.4: Grouping: network lifetime approximation in days for all topologies

Topology	No Grouping	5 GLs	1 GL fails
1	380	400	250
2	355	370	255
3	365	330	255

8.8.4 Scalability factor: Packets to the DA

Figure 8.41 shows that the number of application-level packets sent towards the DA from all SAs decreases considerably when grouping is enabled. In the 5-GLs scenarios where a GL is placed in each location, the number of packets sent to the DA from nodes decreased by 6-7 times compared to the scenarios where no grouping was enabled. The case with GL failure performed equivalent to the scenario with 5 GLs, because the GMs were immediately regrouped under a new GL. This phenomenon is also supported by Figure 8.42, which presents the number of packets in all cases over simulation time. However, the sniffed overall network traffic, shown in Figure 8.43, highlights a different perspective for topology 3 where the number of packets only slightly reduced. This is due to the generated DODAG for topology 3 that cancels out the traffic localisation benefit of grouping, because the DODAG is incongruent to the physical placement of the node, so all traffic routes via the root.

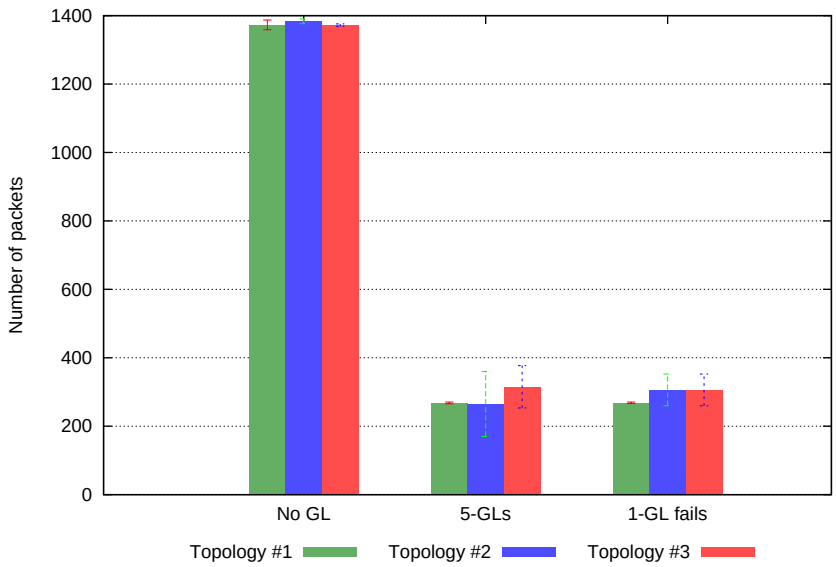


Figure 8.41: Packets at the DA for all scenarios

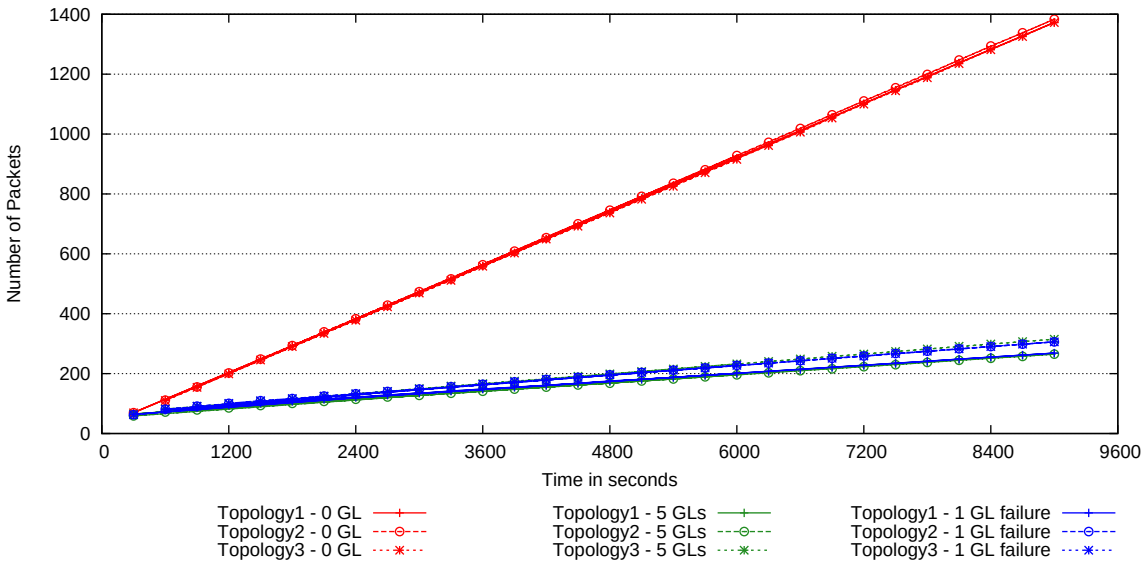


Figure 8.42: Application-level packets at the DA over the time for all scenarios

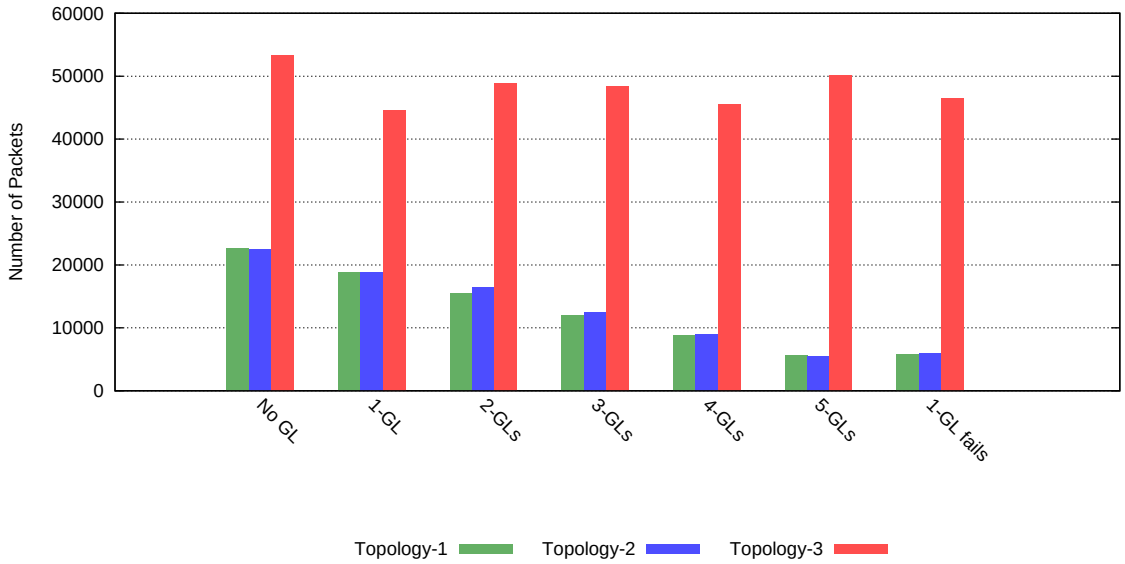


Figure 8.43: Total traffic at the DA over the time for all scenarios

8.8.5 Service Invocation Delay

The grouping mechanism is not focused on improving service invocation delay. Consequently, it is evident from Figure 8.44 that the service invocation delay remains almost same for all scenarios. However, the slight improvement of SI delays in some scenarios of topology 3 is the consequence of the availability of many SAs in the neighbourhood of the DA.

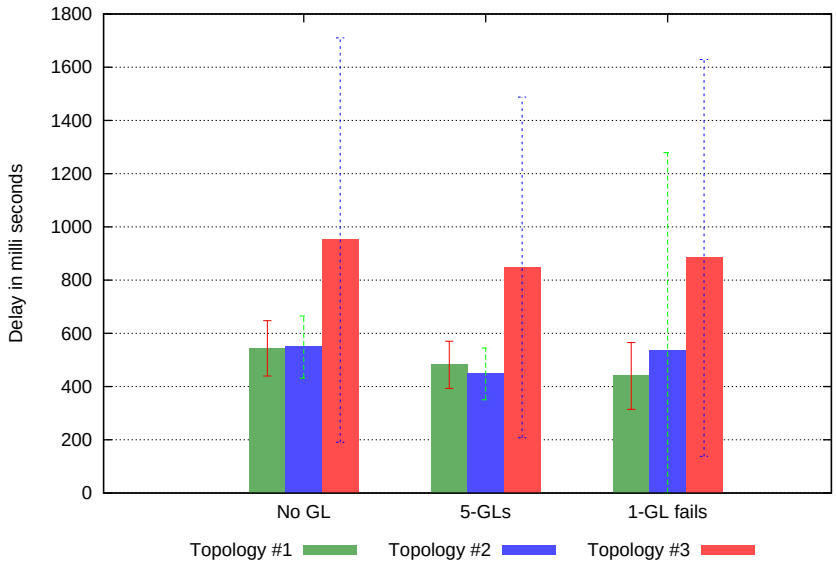


Figure 8.44: Average Service Invocation delay for all cases

8.8.6 Reliability and Accuracy

Grouping distributes the status maintenance task by allocating GLs in different areas of a network. The GLs report to the DA regarding each unresponsive SA to ensure the accuracy of service information maintained at the DA. All service invocation responses are validated and found successful that explains that the DA provided accurate information to all SD requests.

8.9 Summary and Discussion

This chapter covered the detail of a context-aware grouping mechanism that localises status maintenance traffic and enables a paradigm for service composition to facilitate the future WoT applications. Furthermore, it eradicates the bottleneck problem associated with a centralised directory and makes a network more scalable, by localising the status maintenance traffic. It also devises modular profiles for more capable devices to gain benefits of distribution of jobs. The idea of grouping becomes useful when less load balancing is needed, which can be ensured by utilising proposed GL selection algorithm.

The scalability of grouping is evaluated in simulations by varying the size of a group, and it is found that grouping improves the control overehead, energy efficiency and total DA traffic for all cases in topologies with different densities. However, it is also found that the energy consumption and total traffic of a network depend on the RPL's DODAG and the number of direct neighbours of a node. Therefore, it increases the energy consumption of the nodes when the congruency between grouping topology and DODAG increases, which is common in a network with high node density. The scrutiny of grouping mechanism in a network of few groups is also done, which emphasises the role of network's density on total traffic and energy consumption of the network. It is also concluded from the experiments that if a GL fails during the lifetime of the network, the load balancing will increase the overhead, and the newly selected GL pays the price in form of energy consumption. Therefore, the role of the GL selection algorithm becomes important to reduce the probability of a GL failure by selecting a wise choice during the GL selection. In summary, all the results prove that the grouping mechanism considerably reduces the number of packets towards the DA and improves the energy efficiency and network lifetime while marginally increasing the control overhead.

The next chapter covers the detail of the proposed adaptive caching mechanism.

Chapter 9

Adaptive Piggybacked Publishing (APPUB): An Algorithm for Adaptive Caching

9.1 Introduction

Chapters 5, 7 and 8 covered the details of the proposed context-aware SDP, adaptive timer and grouping mechanisms, respectively. However, only service selection mechanism addresses the service invocation delay to improve user response. This chapter focuses on improving the service invocation delay by offering adaptive caching mechanism. Moreover, the adaptability requirement (Section 4.4.3) demands a technique that can adaptively arrange cached values for user applications. In addition, the high number of service invocations increases the traffic and energy consumption for nodes in a multi-hop network by keeping them awake most of the time. Furthermore, the constraints of involved devices can restrict them to be available more frequently, as these are prone to failure because of depleted batteries. Therefore, the effect of the high number of service invocations will decrease the network lifetime. This chapter proposes a technique that arranges cache values by sensing the popularity of services.

9.2 Aims

The aims for the required adaptive caching technique are:

1. The process of maintaining cache values should have low control overhead.
2. The algorithm should be adaptive to a context-aware criteria, e.g., demand of services, so that criteria should be checked first before making the decision

of caching a resource.

3. The design should enable a SA to take the decision of making cache available (PUSH based approach defined in Section 3.12.2).

9.3 Design

This section explains the APPUB technique and its design details including, roles of different entities, algorithmic details.

9.3.1 APPUB Process

APPUB is a demand-based caching technique that balances the trade-off between service invocation delay and network efficiency. It adapts to the demand of a resource for caching rather than blindly maintaining cache of all resources in a network.

The SA implements APPUB algorithm to adaptively send cache of a resource with corresponding lifetime to the DA, when the number of invocations exceeds the hit count threshold. This enables a SA to get help from the registry to share the burden by acting as a proxy in busy times. The DA also passes the resource's cache with the SA's IP address to a UA, if the fresh (not expired) cached value of the resource is available. Figure 9.1 shows how the cache is pushed by a SA and then used by the DA to serve a UA SD query. The benefit of cache hits will decrease the service invocation delay while reducing the number of service invocations queries, which will consequently improve energy consumption of SAs.

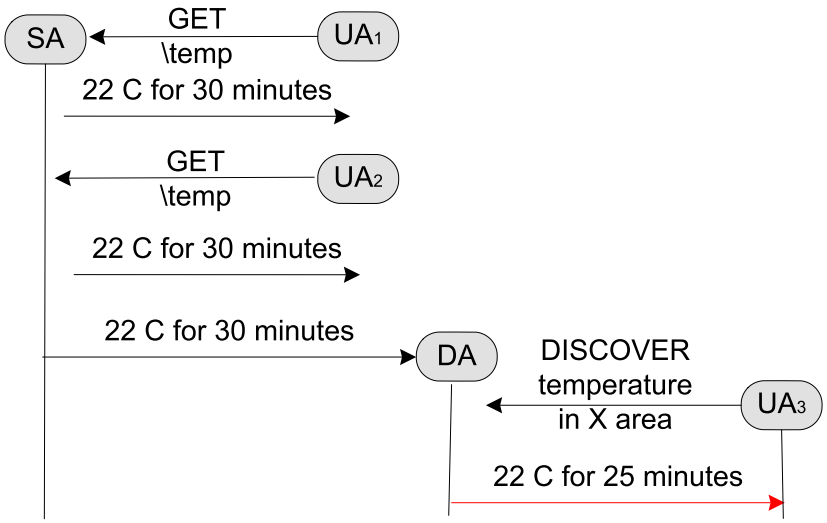


Figure 9.1: APPUB Example Scenario

9.3.2 DA's role

The DA maintains cached values with cached time and cache lifetime for each service. This cache related attributes remain empty until updated by the SA. On top of that, the DA has a `proxy_mode` attribute that defines different proxy modes depending on the ability of a DA. One of the following proxy modes can be used by the DA:

- **'1' - Basic:** This is the default DA mode. The DA sends this proxy mode to inform SAs that no caching is supported, but context attributes and service information is accepted.
- **'2' - Caching is also enabled:** This proxy mode tells the SA that it can support caching, but leave the choice up to the SA to decide when to send the cache.

The DA needs to set the `proxy_mode` at 2 to allow APPUB mechanism. When the DA receives a resource's cache, it saves the received cache with its lifetime and caching time.

9.3.3 SA's role

The SA is the main decision maker to send cache of a service towards the DA. In APPUB, each SA implements following attributes:

- `proxy_mode`: A proxy level assigned by the DA. The default value is 1.
- `num_of_service_invocation_threshold`: This defines the threshold limit of service invocation messages after which a cache is sent. The SA sends the cache of a resource to the DA, when that resource is already invoked `num_of_service_invocation_threshold` times. This value can be set to any value depending on the capability of a SA.
- `resource_hit_count`: This parameter is maintained for each resource to keep the record of its hit counts (service invocations).
- `life_time`: This attribute is kept for each resource and indicates the total life time of a resource in minutes (e.g., temperature after 20 minutes, and light after 10 minutes) as covered in Section 5.9.
- `current_life_time`: It is kept for each resource and updated with the `life_time` whenever resource's cache is sent. This works as a timer to keep track of the remaining time, before which a new cache is not sent.

- `resource_APPUB_enabled`: A resource's cache is only sent if this counter is above 0 and the registry has `proxy_mode` set for caching. This counter is decremented each time a resource's cache is sent.

The APPUB process consists of following steps:

Step 1 - Initiation: The SA starts with the default value of `proxy_mode`. It also has a `num_of_service_invocation_threshold` value to react to the number of service invocations. Each resource has its `resource_hit_count`, `life_time`, `current_life_time` and `APPUB_enabled` to take a decision using APPUB technique.

Step 2 - SA at receiving proxy mode: A SA receives a `proxy_mode` from the DA in response to a status update. It updates its current proxy mode with the received one by following Algorithm 6. The algorithm schedules the next status update message as soon as possible if newly received value is greater than the existing proxy mode. Otherwise, it follows normal scheduling.

Algorithm 6 APPUB: SA at receiving a proxy mode

```
1: prev_proxy_mode ← proxy_level
2: proxy_mode ← received_proxy_level
3: if proxy_mode < 2 then                          ▷ No caching allowed
4:     resource_APPUB_enabled[all] ← 0
5: end if
6: if prev_proxy_mode < proxy_mode then            ▷ Send ASAP
7:     set_next_report_event(in_fixed_smallest_interval)
8: else
9:     schedule_next_interval_normally()           ▷ Normal scheduling
10: end if
```

Step 3 - SA at service invocation: The SA keeps a watch on the number of service invocations of its resources. Each time a resource is invoked, the SA uses Algorithm 7 to decide about the possibility of sending the resource's cache in the next status update message. The resource's hit count is incremented (if DA allows caching) at each service invocation and the resource's cache is scheduled to be sent in next update message (if threshold is surpassed).

Step 4 - SA at status update: APPUB allows the cache to be piggybacked in a status update message. The SA follows Algorithm 8 for every resource to decide whether to piggyback its cache in a status update message.

Algorithm 7 APPUB: SA at service invocation

```
1: if proxy_mode == 2 then                          ▷ If DA enables caching
2:     resource_hit_count++
3:     if resource_hit_count >= service_invocation_threshold then
4:         resource_APPUB_enabled++    ▷ high demand: enable send cache
5:         resource_hit_count = 0
6:         set_next_report_event(in_fixed_smallest_interval)
7:     end if
8: end if
```

Algorithm 8 APPUB: SA while deciding to piggybacked a resource's cache

```
1: if proxy_mode > 1 and resource_APPUB_enabled > 0 and
   resource_current_life_time == 0 then
2:     Piggyback_value_in_payload(resource_value)
3:     resource_current_life_time = resource_life_time
4:     resource_APPUB_enabled--
5: end if
```

9.3.4 Cache Format

APPUB describes a default format to piggyback a resource's cache; however, it is extensible as `Content-Format` CoAP option (Section 2.5.3) can be used to specify other agreed formats. The basic service description format (Section 5.4) is extended to append the cache with the URL within curly braces and placing its lifetime after it. Following are few examples of piggybacking cache in default format.

- `temp{23}30;` - This defines a temperature service with URL *temp*, 23 as resource's cache and 30 minutes lifetime.
- `light{262:814}10;` - This defines a light service with URL *light*, 262 : 814 as cache and 10 minutes lifetime.

9.4 Message Format

9.4.1 Update for TRENDY's UPD reporting message

APPUB technique requires only a small update in DA's response for status maintenance updates (Section 5.11.2). The DA's response now carries a `proxy_level` value (Section 9.3.2) in the payload to inform SAs that it supports caching. Figure 9.2 shows a DA response which mentions that it supports caching.

In the case of a reset message the SA will reset its APPUB related attributes (Section 9.3.3) to default.

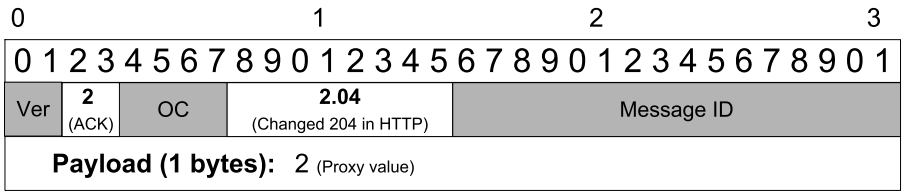


Figure 9.2: APPUB: Changed UPD response from the DA

9.5 Experiments and Results

This section covers the detail the results of experiments performed to analyse the performance of APPUB with the different level of demands and topologies. The performance analysis is done to scrutinise the role of APPUB in network's efficiency and scalability using metrics, including, service invocation delay, energy consumption, control packet overhead.

9.5.1 Introduction

The number of queries is varied (100 and 1000) to repeat the following scenarios:

Case 1 - Basic TRENDY without APPUB: This scenario enables the basic functionality of the proposed solution. The UA gets a best matched resource's URL and IP address of the SA hosting the matching service.

Case 2 - Basic TRENDY with APPUB threshold 2: This scenario has APPUB enabled with `num_of_service_invocation_threshold` fixed at 2 on top of case 1's functionality.

Case 3 - Basic TRENDY with APPUB threshold 1: This case is exactly like case 2 with `num_of_service_invocation_threshold` fixed at 1.

The `num_of_service_invocation_threshold` is varied to check the effect of APPUB in scenarios with different topologies and service demand. The reaction of APPUB depends on multiple factors, including, the number of specific queries for a location, and the number of times when services of the same SA are preferred by the service selection mechanism. In experiments, the UA queries are generated arbitrarily and sent after a random interval, that's why only two threshold values are tested, as even their result was found similar. However, depending on the application environment and its requirements, any value can be chosen for the hit count threshold or it can be changed dynamically.

The simulations are configured with the same settings defined in Section 6.3 with the changes given in Table 9.1. Furthermore, all SAs use the cache lifetime of

10, 30 and 60 minutes for light, temperature and humidity resources, respectively. These cache lifetimes were selected randomly and have no impact on the query generation process.

Table 9.1: Configurations for the experiments of APPUB

Total nodes	35 nodes + 1 border router
Physical topologies	1, 2 and 3 (Section 6.3.1)
RDC	ContikiMAC and NullRDC
Number of locations	5 with 7 SAs in each location
Total Service Discovery queries	100 and 1000
Total Service Invocation queries	100 and 1000
Number of cases	3 (cases)× 2 (query variations) = 6
First UA Query	At 600 seconds
DA time window	5 minutes = 300 seconds
Number of time windows	30
Simulation Duration	300 × 30 = 9000 seconds

This section continues with the generated results of simulations to analyse the effect of the APPUB mechanism.

9.5.2 Service Invocation Delay and Cache hits

The APPUB mechanism adaptively makes cached values available at the DA, which are then used to serve future queries to decrease the overall service invocation delay and overhead incur while passing a message over a multi-hop network. Figure 9.3 shows the number of cache hits (the number of times maintained cache is used) in all cases. Both APPUB cases have served $60 - 70\%$ of UA queries with cached values regardless of the employed RDC. There is no significant difference between case 2 and 3, because of the randomness introduced by query generation process, and service selection's preference for choosing a SA to serve a query. Furthermore, the figure shows a contradiction where more sensitive hit-count threshold value actually proved less efficient in some scenarios. However, the scenarios with small threshold value show better performance in the cases of 100 queries, because of the randomness in the query generation and arrival process.

Figure 9.4 demonstrates the impact of APPUB for first 100 UA queries in scenarios with 1000 queries using ContikiMAC. It is evident that the DA started serving the queries with cache values after first 40 randomly generated queries. This reaction comes from the adaptive nature of APPUB that senses the high number of queries and arranges cache at the DA. Furthermore, the small hit count threshold value reacted to the load quickly to serve the query with a cached value

than the large value. The service invocation delay for topology 3 is the highest because of its density, which produces a more dense RPL's DODAG near the root node (covered in Section 6.3.1).

The effect of cache hits is evident in Figure 9.5a for ContikiMAC scenarios, which shows that the mean service invocation delay for 1000 queries reduced to more than half in both APPUB scenarios. However, the NullRDC scenarios, shown in Figure 9.5b, have performed better than ContikiMAC scenarios, because the radio is never switched off in these scenarios which ultimately increase the energy cost for the nodes. On the other hand, ContikiMAC keeps the radio switched off for most of the time, but has performed closer to NullRDC when APPUB is employed. Moreover, Figure 9.6 shows that the cache-hits also optimises the delay for those queries which are not served by a cached value and thus need a separate message to invoke a service. The major benefit is noticed for topology 3, which decreases the delay because of the reduction in the number of messages communicated inside a dense 6LoWPAN.

In conclusion, the APPUB mechanism adapts to the number of service invocations in a network and arranges cache at the DA to reduce the load in a 6LoWPAN. The behaviour of APPUB remains the same in different topologies where it diminishes the average service invocation delay. In addition, the evaluation explains that the cache hits eliminate the need of separate service invocation queries that improve the response time in a dense network.

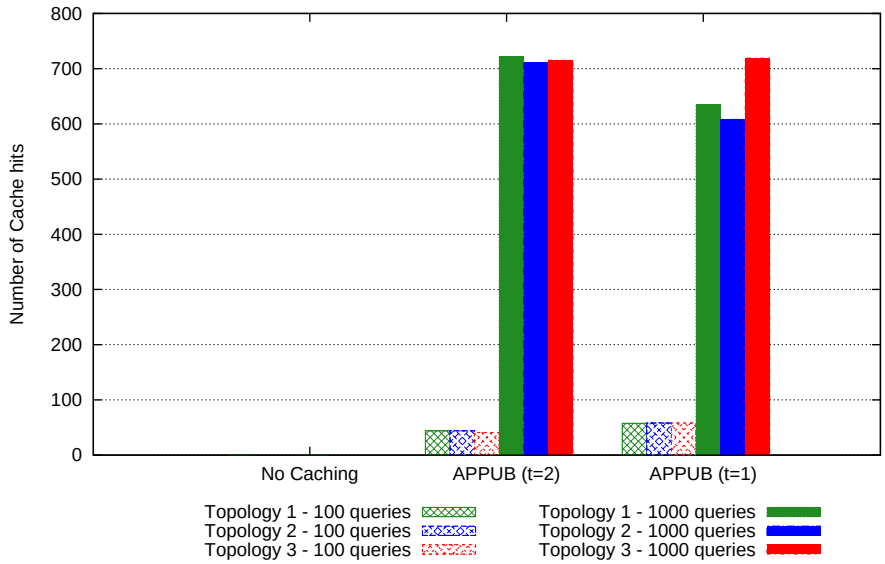
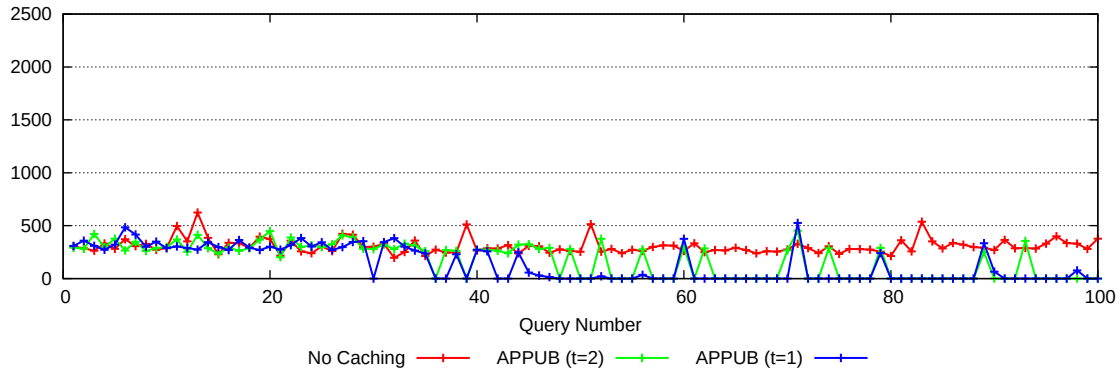
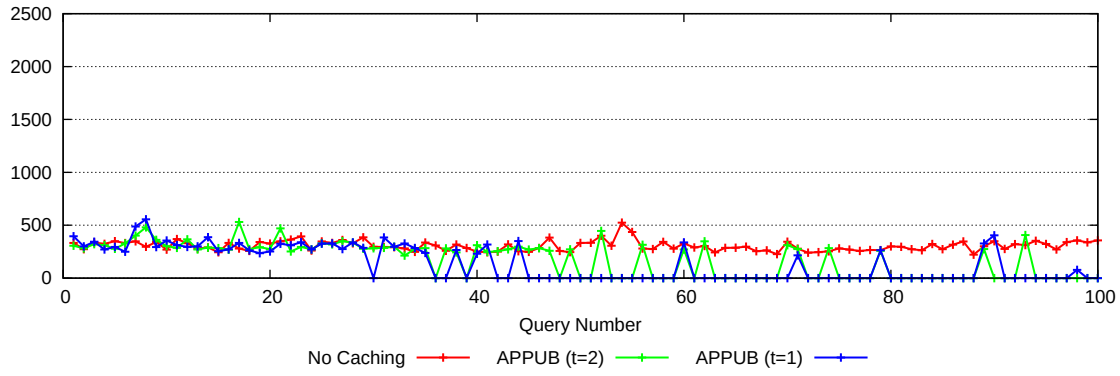


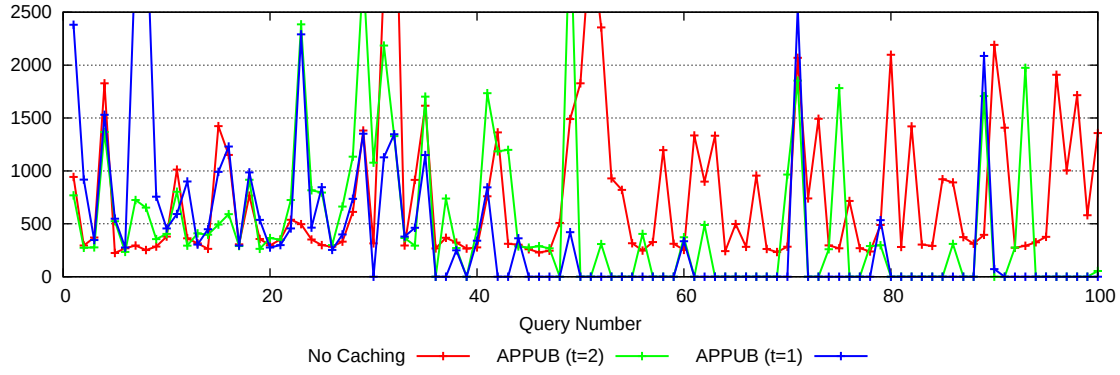
Figure 9.3: Cache hits for queries for topologies 1-3 (same for both ContikiMAC and NullRDC scenarios)



(a) Topology 1

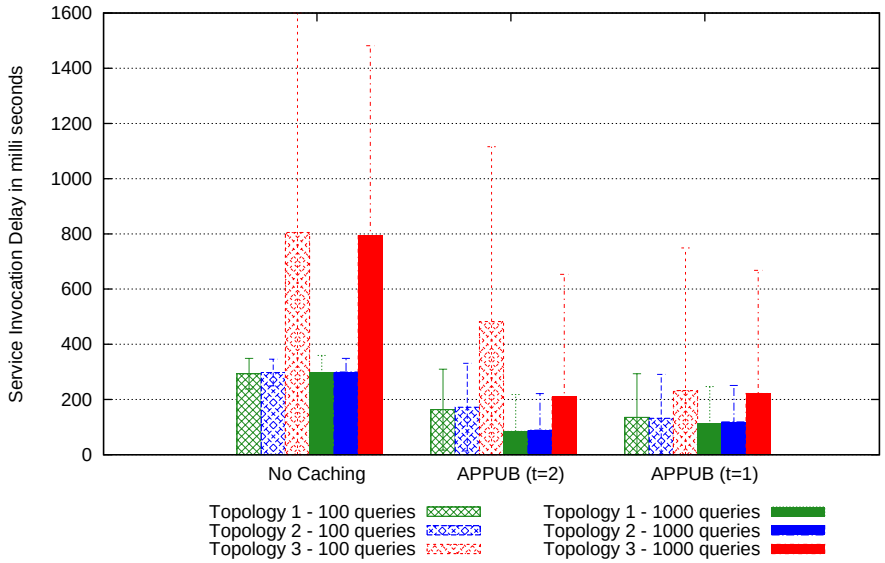


(b) Topology 2

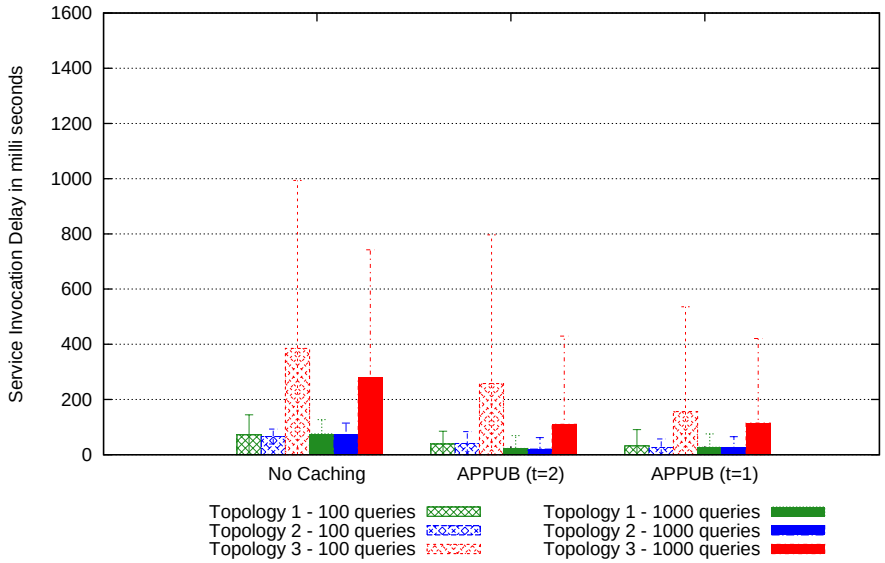


(c) Topology 3

Figure 9.4: APPUB trend: Service Invocation delay for first 100 queries using ContikiMAC

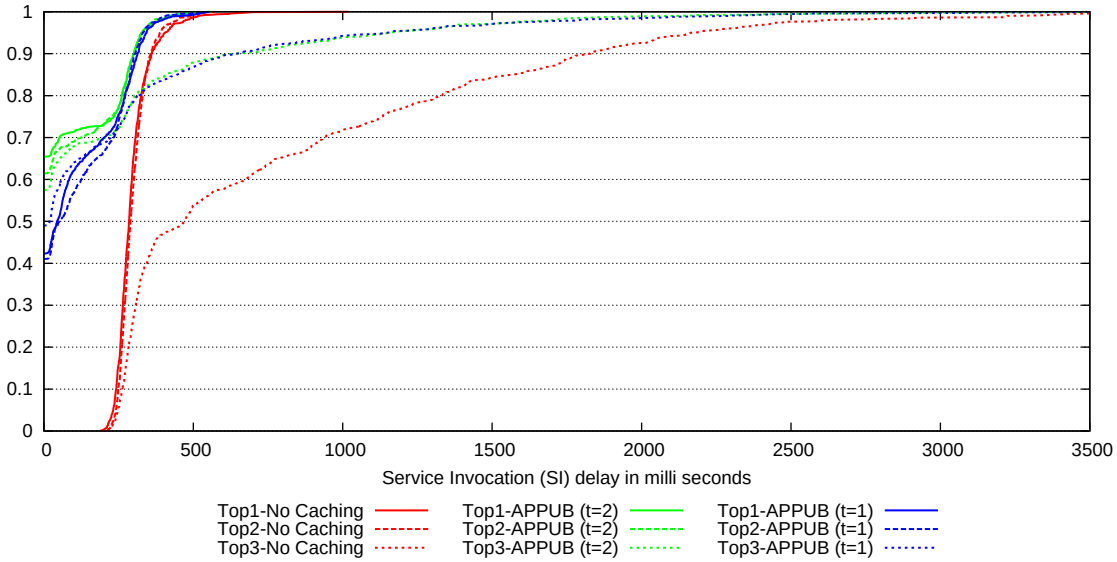


(a) ContikiMAC scenarios

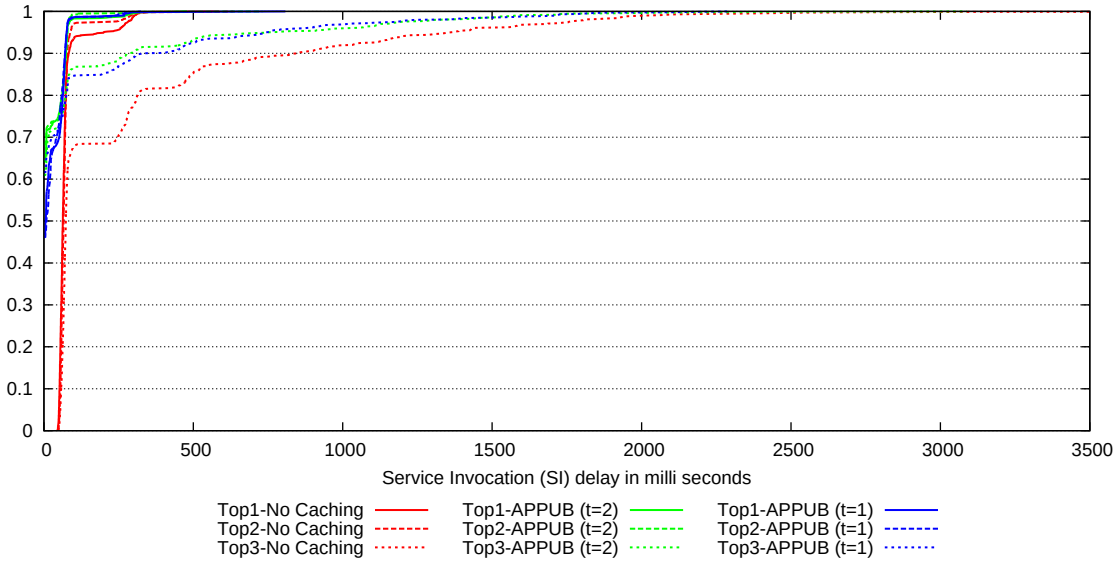


(b) NullRDC scenarios

Figure 9.5: Average Service Invocation delay for topologies 1, 2 and 3



(a) ContikiMAC scenarios



(b) NullRDC scenarios

Figure 9.6: CDF graph of service Invocation delay with 1000 queries for topologies 1-3

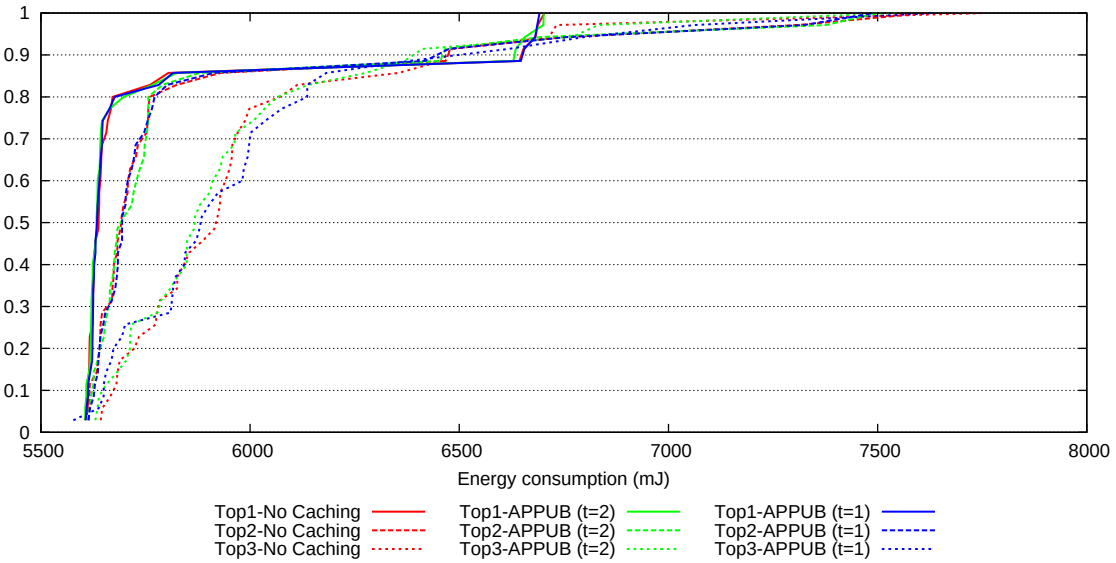
9.5.3 Energy Consumption and network lifetime

This section covers the detail of energy consumption of nodes in ContikiMAC scenarios, whereas NullRDC cases are not compared, because keeping the radio always on consumes $70 - 75$ times more energy than ContikiMAC. Additionally, if the radio is constantly on, then no technique can improve the energy consumption of nodes, because radio listening and receiving cost more energy than any other activity.

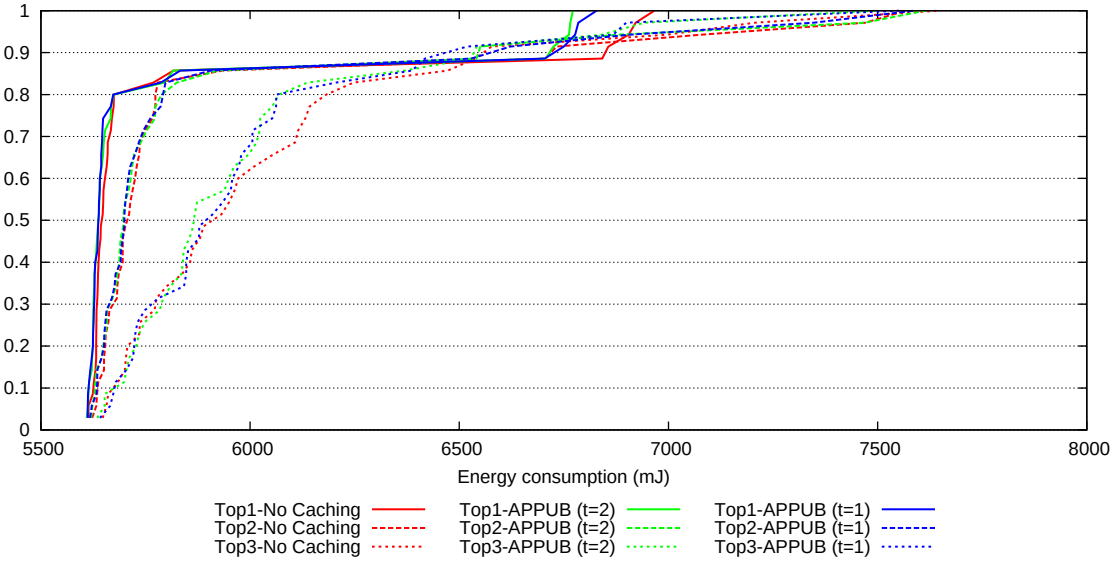
Figure 9.7 shows the individual energy consumption of nodes in case of 100 and 1000 queries. The APPUB enabled scenarios have performed slightly better in case of 1000 queries to reduce the energy consumption of nodes, because the cache hits have avoided the need of service invocation requests. Nevertheless, it can be noticed that the overhead of APPUB has increased the energy consumption for the upper $10th$ percentile of the nodes in scenarios with fewer queries. On the other hand, topology 1 shows efficiency for top nodes in energy consumption because of its topology configurations. However, topologies 2 and 3 have not demonstrated much benefit in energy-efficiency terms, because of their densities and formed DODAGs. Figure 9.8 emphasises this phenomenon where topology 2 and 3 perform worst for the top node in terms of energy consumption in all cases. Furthermore, Table 9.2 projects the energy consumption of top nodes linearly to approximate the network lifetime, and supports the same phenomenon. Therefore, it can be concluded that the benefit of APPUB in increasing lifetime of a network is actually depends on the combination of different factors, including the arrangement of the physical topology, the generated RPL tree and the number of service invocation requests.

Table 9.2: Network lifetime approximation for different number of queries for topologies 1-3

Network Lifetime (days)						
Case	**Topology #1**		**Topology #2**		**Topology #3**	
	100	**1000**	**100**	**1000**	**100**	**1000**
No caching	415	400	365	370	360	365
with APPUB (t=2)	415	415	375	365	370	375
with APPUB (t=1)	415	410	375	370	365	370



(a) Scenarios with 100 queries



(b) Scenarios with 1000 queries

Figure 9.7: CDF graph of service Invocation delay for topologies 1-3

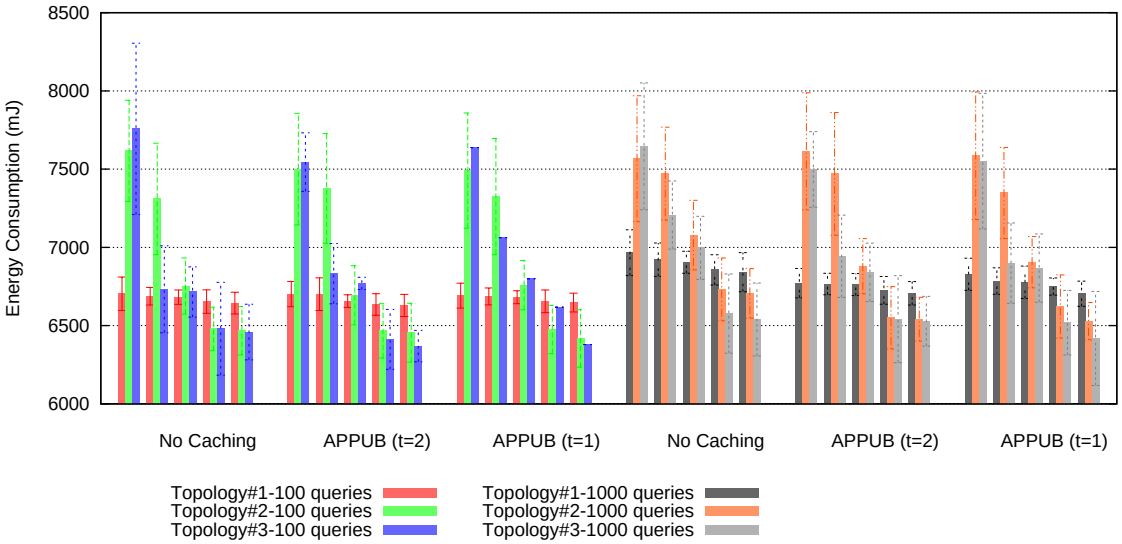


Figure 9.8: Energy consumption of top 5 nodes for scenarios with 100 and 1000 queries only ContikiMAC scenarios for topologies 1-3

9.5.4 Control packet overhead

APPUB increases the control packet overhead because of its extra messages that are used to push cache values at the DA. In case of high demand of a resource, the APPUB mechanism sends the next status update before the expiration of the earlier scheduled interval. This difference is more visible in Figure 9.9 which shows that some nodes have sent more packets when APPUB is employed. Anyhow, the overall overhead is marginal as shown in Figure 9.10, because a SA switches back to normal scheduling after it receives the response from the DA. However, this overhead translates to a better network response to UAs, as the DA uses cached values to serve queries for a determined time, which eradicates the need for extra service invocation messages. Therefore, APPUB actually decreases the control overhead, if the equation also includes the number of service invocation messages. Figure 9.11 depicts this phenomenon by showing that the overall traffic in the network actually decreased with the employment of APPUB. The effect is more evident in cases with 1000 queries because the number of service invocation messages reduced because of caching, as shown in Figure 9.12. A more detailed view of service invocation messages in terms of request and response is depicted in Figure 9.13, which explains the same phenomenon.

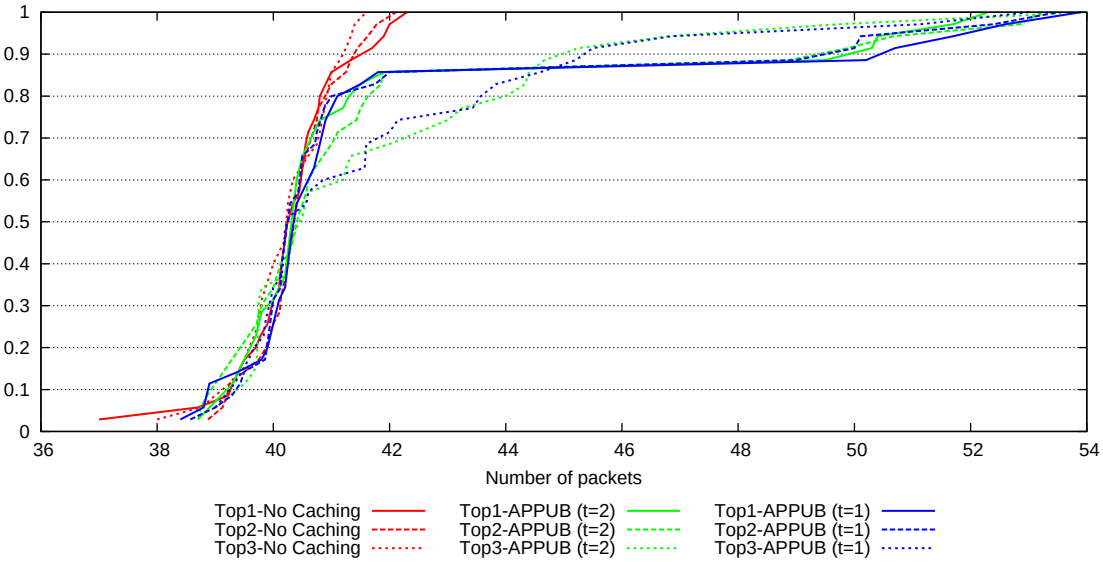


Figure 9.9: CDF of application-level control packet overhead per node for each node for all scenarios

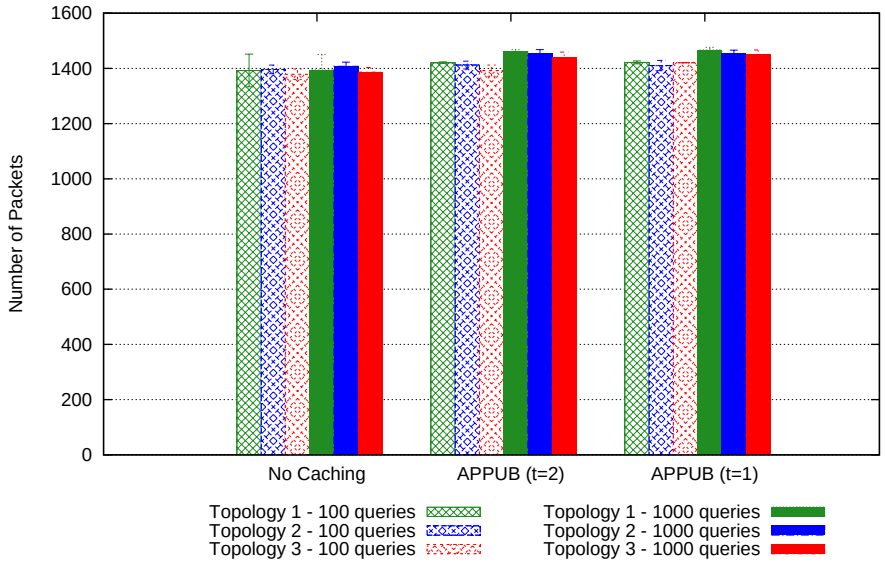


Figure 9.10: Aggregated application-level control packet overhead for all scenarios

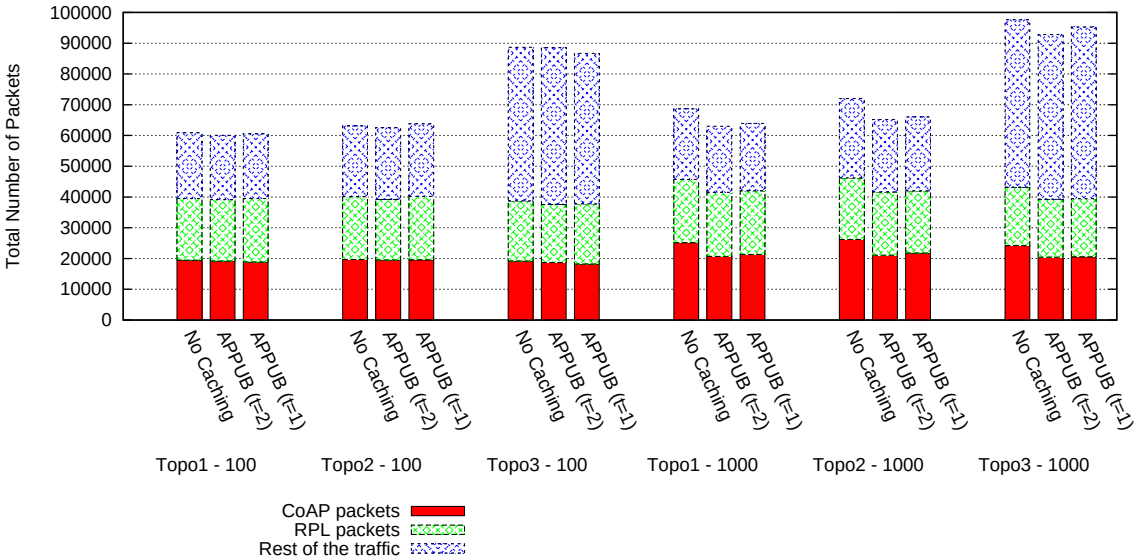


Figure 9.11: Total network traffic in ContikiMAC and NullRDC cases for topologies 1, 2 and 3

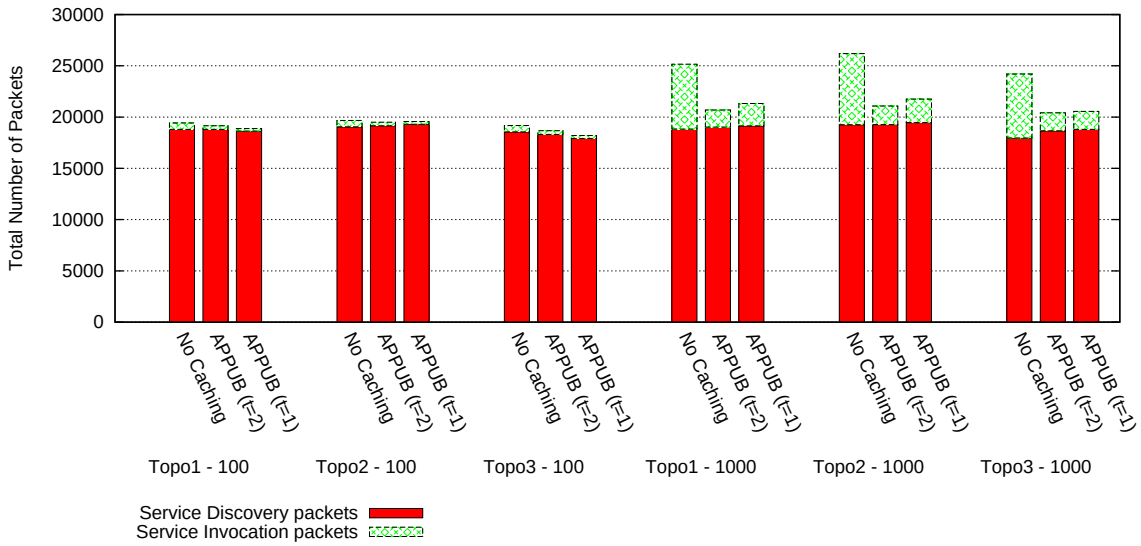


Figure 9.12: CoAP's traffic details in ContikiMAC and NullRDC cases for topologies 1, 2 and 3

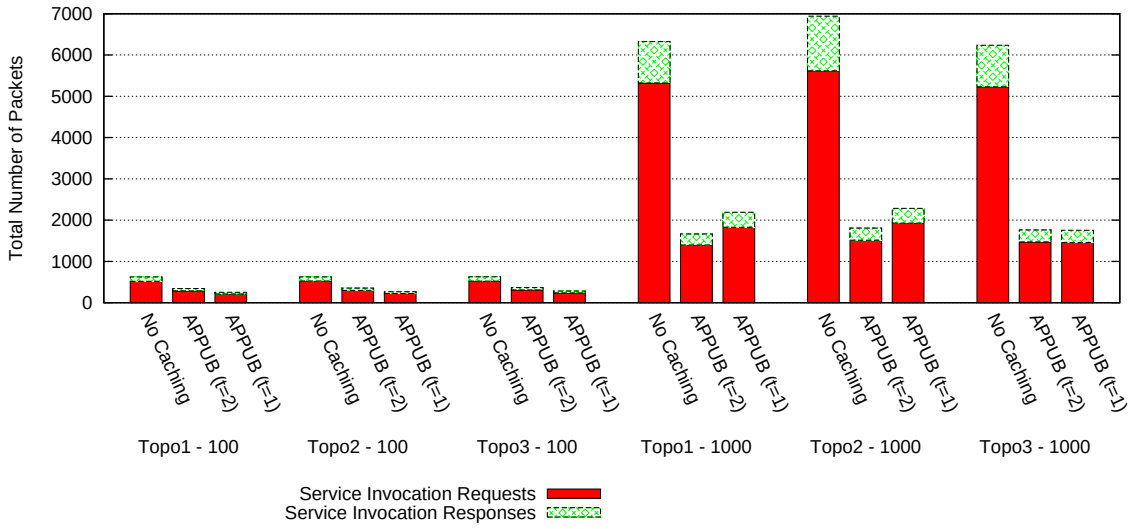


Figure 9.13: Service Invocation traffic details for ContikiMAC scenarios for topologies 1, 2 and 3

9.5.5 Scalability factor: Packets to the DA

The APPUB mechanism has no effect on the scalability factor, as the application-level messages received at the DA remain the same as the number of control packets generated by SAs, as shown in Figure 9.10. Furthermore, Figure 9.14 shows the number of packets sent or received between the DA and 6LoWPAN nodes (excluding service invocation messages). The density of topology 3 is the main reason of its high number of packets, which are increased further with APPUB. However, the overall impact remains marginal for all topologies.

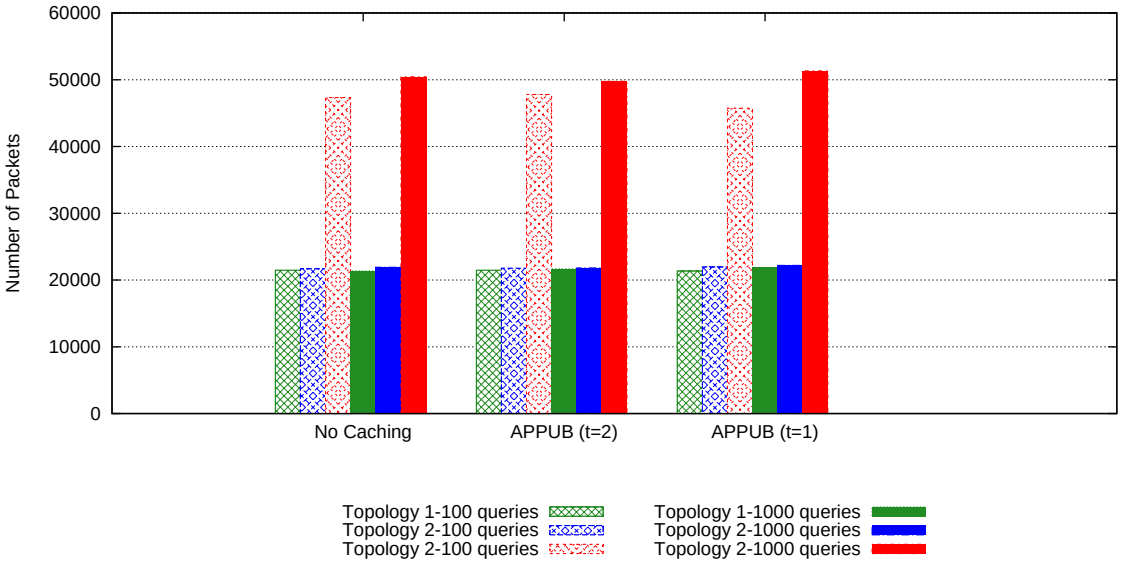


Figure 9.14: Total traffic at the DA over the time for all scenarios

9.5.6 Reliability and Accuracy

All service invocation responses are verified and found successful. However, APPUB provides the caching mechanism on top of TRENDY by enabling the DA to serve SD queries with cached values. The accuracy (freshness) of these values is ensured by a cache lifetime attribute, which is sent by SAs with each cache (Section 9.3.4). The DA saves the cached time to ensure that it will not serve any query with an expired cached value. Therefore, all cached values provided by the DA will be considered accurate.

9.6 Summary and Discussion

This chapter has covered the detail of the proposed APPUB, adaptive caching technique, that balances the trade-off between service invocation delay and network

efficiency. This method provides an alternative to proxy behaviour by reacting to the increasing number of service invocations. It adapts to the demand of a resource for caching rather than blindly caching all resources in a network. Each SA implements APPUB functionality to reduce the burden of service invocations by sending cached values and corresponding lifetime values to the registry. This mechanism offers multiple benefits: it decreases the service invocation delay and also allows service hosts to share their load with the resource directory.

Several simulation experiments with emulated hardware nodes validate the benefit of using APPUB technique in terms of low service invocation delay, overall control packet overhead and energy consumption. The evaluation of the results supports the efficiency of APPUB, but it also draws some important conclusions. The density of a network plays a significant role in increasing the overall packet overhead of the network, which is further augmented by the extra packets used for caching. Nevertheless, the employment of APPUB reduces the number of service invocation messages and consequently, decreases the overhead of a constrained network. In addition, the low service invocation delay with APPUB improves the user experience. However, this benefit comes only when the network is enough busy and maintained cache is being used by the DA to serve the queries, so it depends on the carefully chosen sensitivity-level of the APPUB hit-count threshold. This research project arguments that it should be left to the ease of a SA, which can choose a hit count threshold depending on its available resources. In this case, different SAs are allowed to have a diverse range of threshold values. Even though, this makes APPUB a PUSH-based approach, it still can face challenging situations where high-capacity nodes can have high threshold values and can cost more energy for constrained nodes within a multi-hop network. This can be improved by changing the RPL OF settings, so the high-capacity nodes remain on high level of a DODAG tree and never use a path of constrained nodes. Another perspective which is out of scope of this thesis is the case of a dense network where a DODAG's root has many nodes at the first level that increases neighbour discovery traffic and consequently, incurs high delays. Furthermore, a node can select a lifetime for its cache value depending on its constraints and the criteria and requirements of a network. It can use any algorithm to change this lifetime value adaptively.

The next chapter explains the effect of combining different TRENDY techniques proposed in Chapters 7, 8 and 9.

Chapter 10

TRENDY: a Trend-based Service Discovery Solution for the IoT

10.1 Introduction

Chapters 7, 8 and 9 have introduced new adaptive and grouping techniques for the TRENDY protocol to cope with the IoT requirements gathered in Chapter 4. The proposed techniques require small changes in TRENDY's message formats (Section 5.11). Furthermore, new rules are needed when a combination of these techniques is used together. This chapter explains the usage of TRENDY with the combination of proposed techniques. In addition, it discusses the evaluation done by undertaking experiments to analyse the effect of using a combination of these techniques in a network.

10.2 TRENDY Protocol with Adaptive Techniques

This section describes the TRENDY protocol with the integration of adaptive timer, APPUB and grouping techniques.

10.2.1 Message Format

The merger of timer, grouping and APPUB together with TRENDY requires only one small update in the response of the DA for status maintenance updates (Section 5.11.2). In the new format, the DA sends both `trendy_counter` and `proxy_mode` in the payload to the SA. Figure 10.1 shows the order of attributes in the updated message format. Grouping mechanism requires no change in the TRENDY message formats (Section 5.11); however, the YGL message now needs to

include the `trendy_counter` and `proxy_mode` values in the payload. Furthermore, the GL will respond to status update similar to the DA (Figure 10.1).

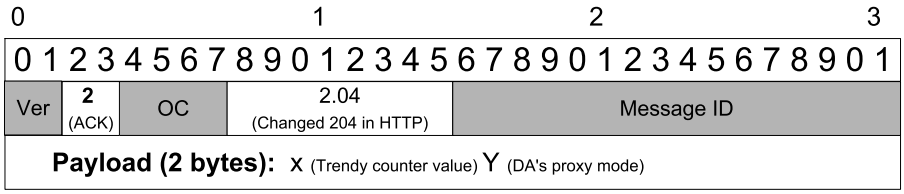


Figure 10.1: The DA's response to a status update message

10.2.2 SA roles

A SA can be a GM or GL. Each SA needs to implement capabilities defined in Sections 5.2.2, 7.3.1 and 8.3.1. On the other hand, the APPUB technique (Section 9.3.3) and GL role (Section 8.3.2) are optional features, which can be enabled depending on the capability of a SA.

10.2.3 Policies

All GLs need to specify the `trendy_counter` and `proxy_mode` attributes in response to status updates, as shown in Figure 10.1. If a GL only supports status maintenance (Section 8.3.2), then it should set the `proxy_mode` at '0' to stop GMs from sending any extra information including, context attributes. The GM maintains another `GL_proxy_mode` attribute, which is updated after receiving YGL message. All GMs that implement the APPUB role (Section 9.3.3) use `proxy_mode` to decide about the adaptive caching. In case of APPUB, the cache is always sent to the DA. After sending cache to the DA, a GM needs to update its local GL as soon as possible (if it's grouped).

10.2.4 Framework

TRENDY's framework (Section 5.10) is updated by the inclusion of new techniques to offer a more efficient SD solution and to fill the research gaps. Figure 10.2 shows the updated version of the framework.

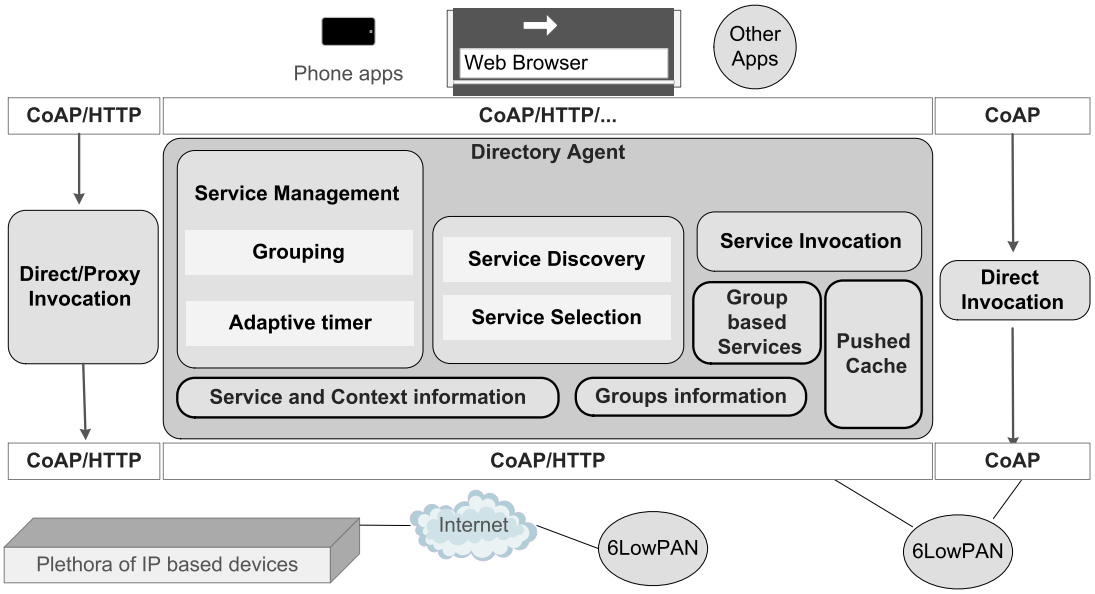


Figure 10.2: New framework of TRENDY

10.3 Experiments and results

10.3.1 Introduction

Different combinations of TRENDY techniques are employed in the experiments to evaluate the performance metrics (Section 6.2). The employed GLs in grouping scenarios only manage status maintenance of their GMs. The number of queries are varied (100 and 1000) for the following scenarios:

Case 1 - Basic TRENDY Service discovery (SD): This scenario enables the basic functionality of TRENDY with service selection. All SAs send their status reports to the DA in each time window.

Case 2 - Adaptive timer and grouping: In this scenario, grouping and timer (`hit_count_threshold` fixed at 2) techniques are employed with the functionality of case 1.

Case 3 - APPUB and timer: This scenario employs the APPUB technique with the `service_invocations_threshold` fixed at 2 and the timer is configured similar to case 2.

Case 4 - APPUB with timer and grouping: In this scenario, grouping is also enabled on top of the case 3.

The simulations are configured similarly as defined in Section 6.3 with the changes given in Table 10.1. Furthermore, all SAs in APPUB scenarios use

the cache lifetime of 10, 30 and 60 minutes for light, temperature and humidity resources, respectively. All scenarios have 30 SAs hosting the mentioned services. In grouping scenarios 5 SAs act as GLs, whereas in non-grouping scenarios 5 SAs offer no service for a better comparison for all scenarios.

Table 10.1: Configurations for the experiments of TRENDY techniques

Total nodes	35 nodes + 1 border router
Physical topologies	1, 2 and 3 (Section 6.3.1)
RDC	ContikiMAC
Number of locations	5 with 7 nodes in each location
Number of SAs hosting services	6 in each location
Number of cases	4 (cases)× 2 (query variations) = 8
Total Service Discovery queries	100 and 1000
Total Service Invocation queries	100 and 1000
First UA Query	At 600 seconds
DA time window	7 minutes = 420 seconds
Number of time windows	20
Simulation Duration	420 × 20 = 8400 seconds

The impact of different TRENDY techniques on different performance metrics is covered in Chapters 7, 8 and 9. This section focuses on their effect when those techniques are used together in a scenario by analysing the results of simulations.

10.3.2 Service Invocation Delay and Cache hits

Figure 10.3 shows that the average service invocation delay decreased considerably with the employment of APPUB and to some extent with timer in cases 2-4. The delay for UA queries in topology 3 is higher than other topologies in all scenarios, because of its high density. Overall, the substantial improvement is only experienced in the scenarios where the number of queries is higher and APPUB is employed by SAs. This performance is further elaborated in Figure 10.4, which depicts that APPUB enabled the DA to serve around 70% queries with cache in scenarios with 1000 queries. Moreover, timer in case 2 reduces the delay for topology 3, because the reduction in the number of status updates eventually translates into low delay only in a dense network. Figure 10.5 expresses this benefit in a CDF graph that supports the same hypothesis. Furthermore, it explains that provision of cache by APPUB reduces the 6LoWPAN's burden and consequently, decreases the delay for those queries where no cache is found at the DA, and separate service invocation requests are sent.

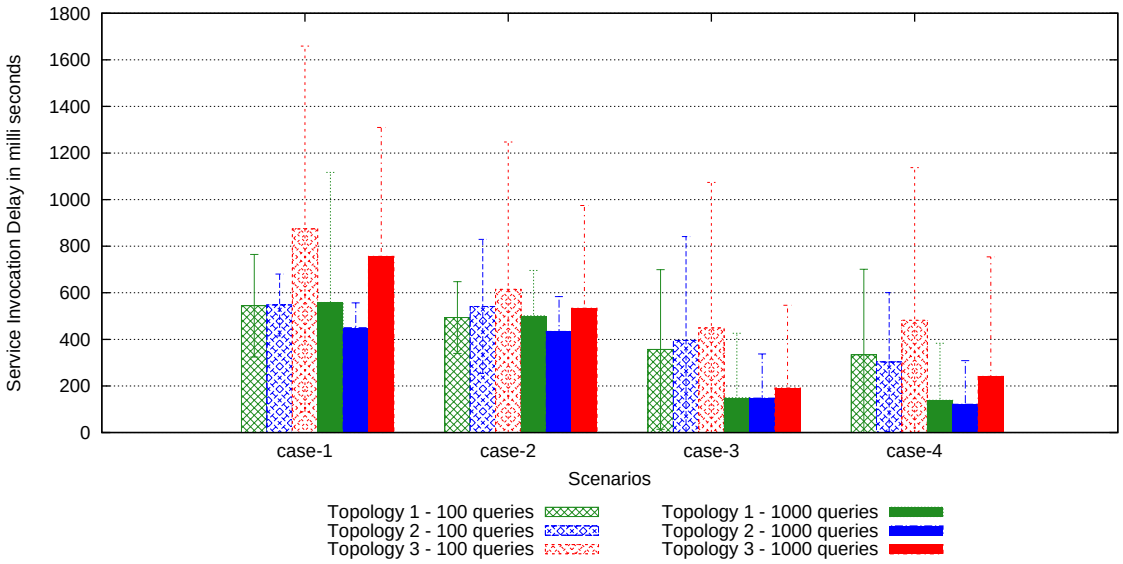


Figure 10.3: Average service Invocation delay for topologies 1-3 with the different number of queries

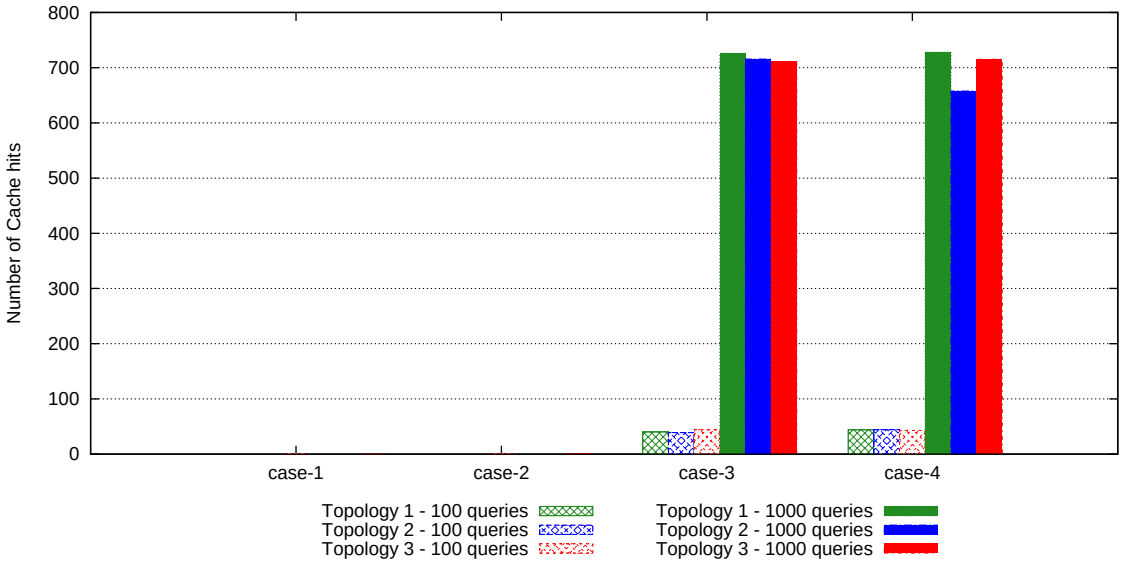


Figure 10.4: Total cache hits for topologies 1-3 with the different number of queries

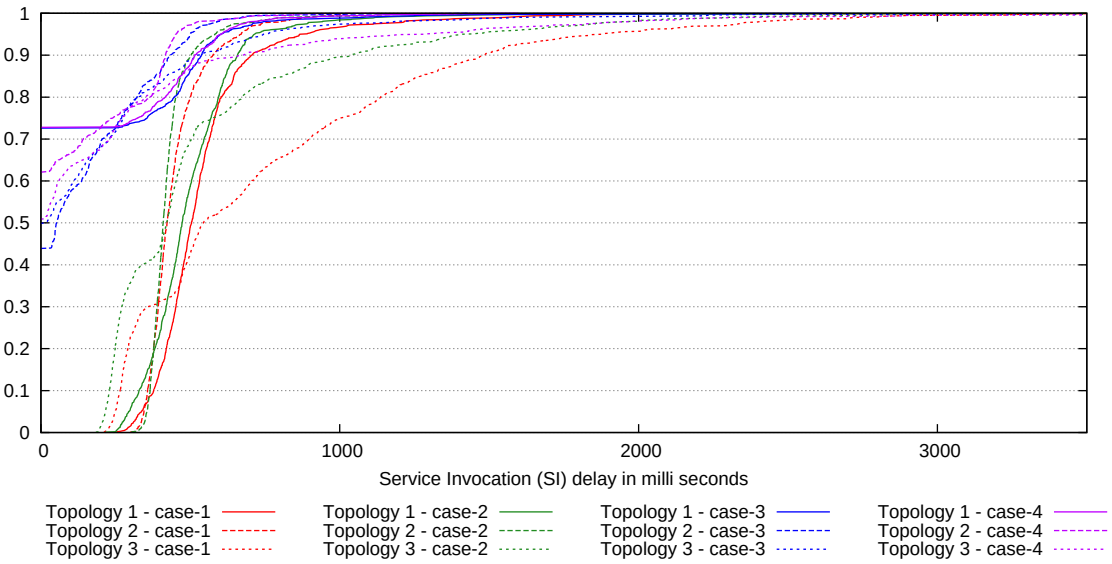


Figure 10.5: CDF graphs of service Invocation delay for topologies 1-3 with 1000 queries

10.3.3 Control packet overhead

Figure 10.6 shows the significance of using the adaptive timer mechanism, as cases 2-4 with timer have low application-level control overhead for all topologies. However, case 4 has created more control traffic compared to cases 2 and 3, because it uses both grouping and APPUB techniques, which require extra traffic to create groups and to send status updates for caching (updates sent before normal scheduled time).

The application-level control overhead can produce many times higher traffic in the whole network, which depends on the network's density and formed RPL's DODAG. Therefore, a total network control packet overhead is also analysed to measure the impact of control traffic, as shown in Figure 10.7. The analysis shows that timer reduces total traffic in cases 2-4 for all topologies. Furthermore, networks with topologies 1 and 2 incur almost the same packet overhead. However, topology 3 produces more traffic when grouping is enabled in cases 2 and 4, because of its density. In addition, Figure 10.8 shows that the number of packets below the routing layer are much higher for topology 3. Moreover, Figure 10.9 emphasises that the high overhead for scenarios with 1000 queries is the result of extra traffic generated by service invocation messages that eventually decreases in cases 3-4 for all topologies, where APPUB is employed.

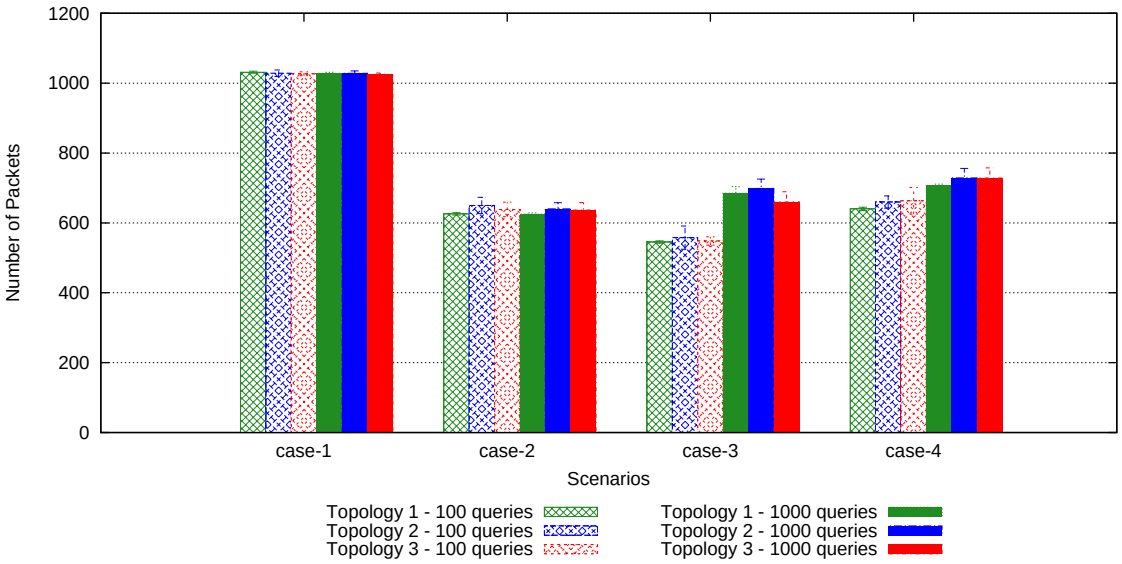


Figure 10.6: Application-level control packet overhead for topologies 1-3 with the different number of queries

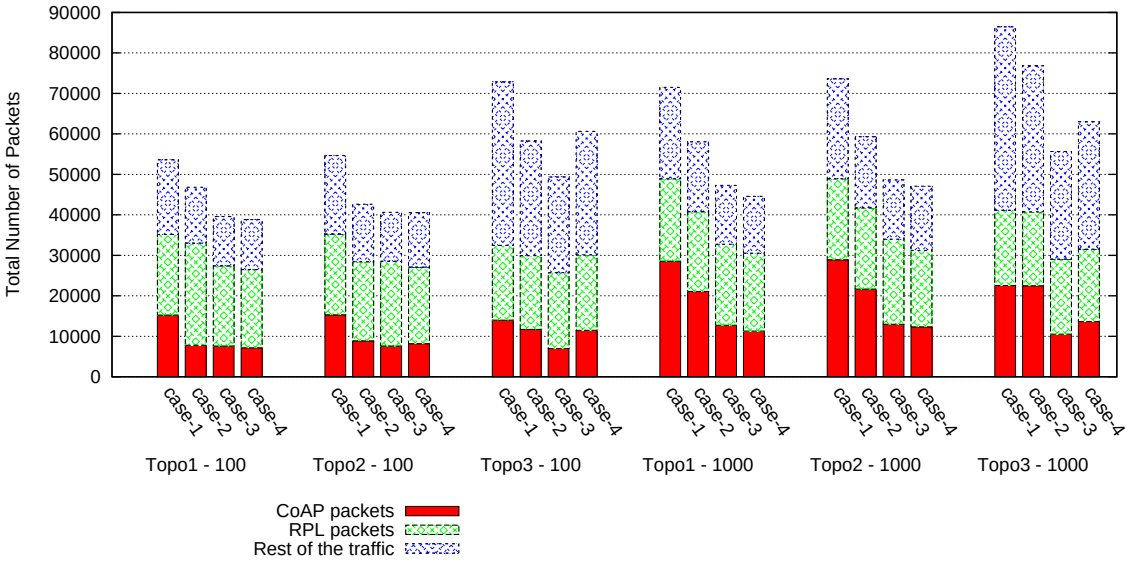


Figure 10.7: Overall traffic for topologies 1-3 with the different number of queries

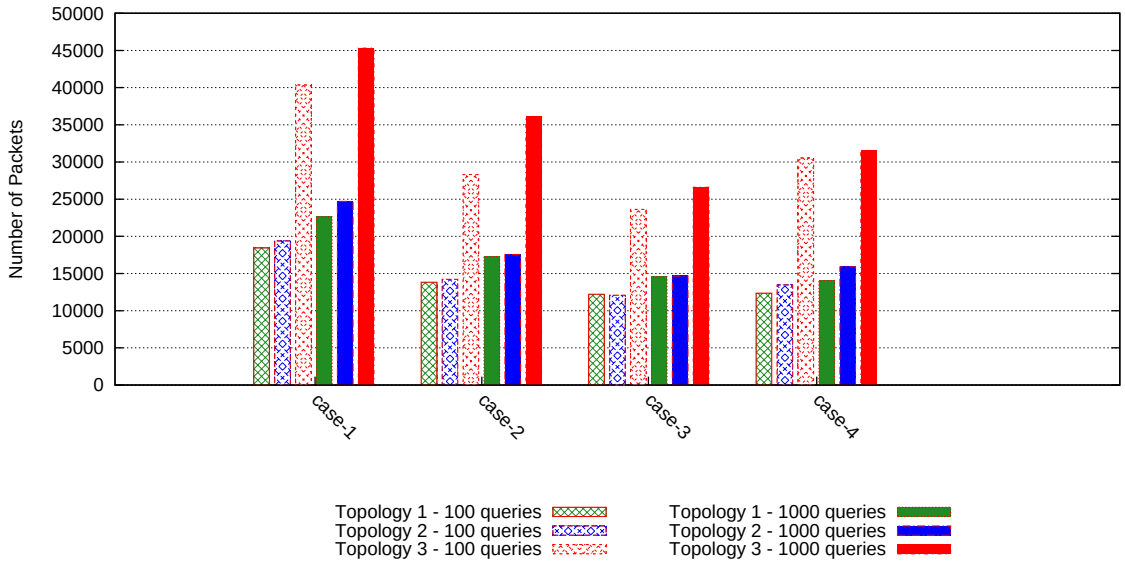


Figure 10.8: Traffic at layers below RPL for topologies 1-3 with the different number of queries

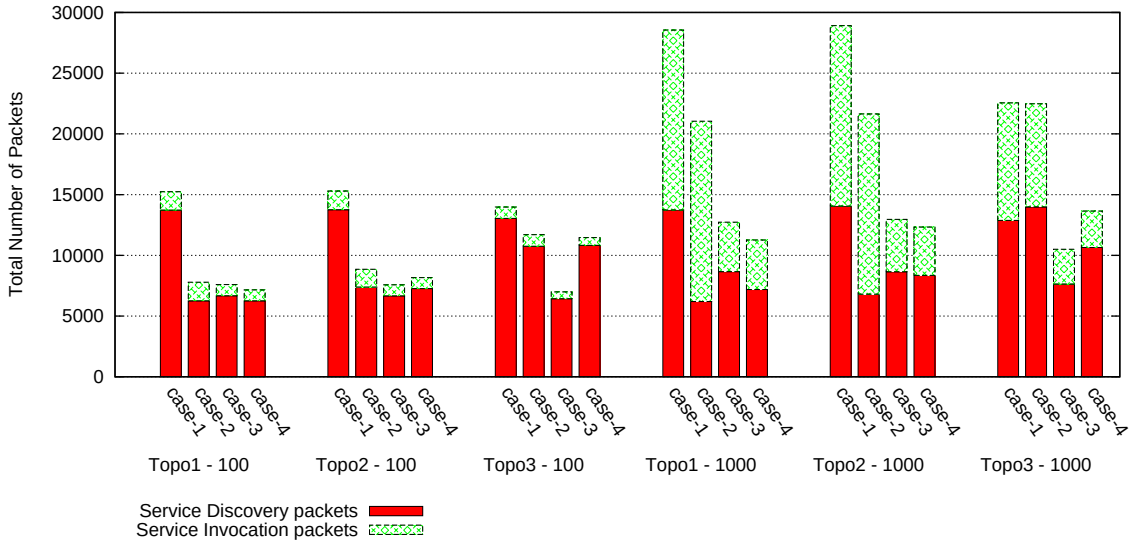


Figure 10.9: CoAP traffic for topologies 1-3 with the different number of queries

10.3.4 Scalability factor: Packets to the DA

TRENDY timer and grouping techniques are designed to reduce the number of packets sent towards the DA from a 6LoWPAN by decreasing and localising the traffic. Figure 10.10 depicts the same phenomenon by further explaining that benefit of both techniques aggregated when their combination is used in cases 2 and 4. The slight increase in case 4 with 1000 queries is due to the extra traffic generated

by APPUB. Figure 10.11 shows the equivalent trend over the simulation's period for scenarios with 1000 queries, which explains that the efficiency in cases with grouping will remain effective in the long-run. Whereas, case 3 also demonstrate timer's efficiency, but it decreases the overhead adaptively, because the adaptation of the timer's counter value over the time. The rise in case 4 compared to case 2 is caused by extra packets needed to send for caching; however, both cases performed almost similarly after 5000 seconds because no cache message is sent after that time.

The total number of packets sent and received by the DA (excluding service invocation messages) are shown in Figure 10.12. The number is much higher than the application-level packets, but the overall trends remain the same. However, topology 3 shows the impact of its density by increasing the actual traffic then what is anticipated. Moreover, it is important to notice that, in reality, grouping has increased the traffic at the DA in topology 3, because of the incongruity exist between the application-level grouping topology and the generated RPL topology.

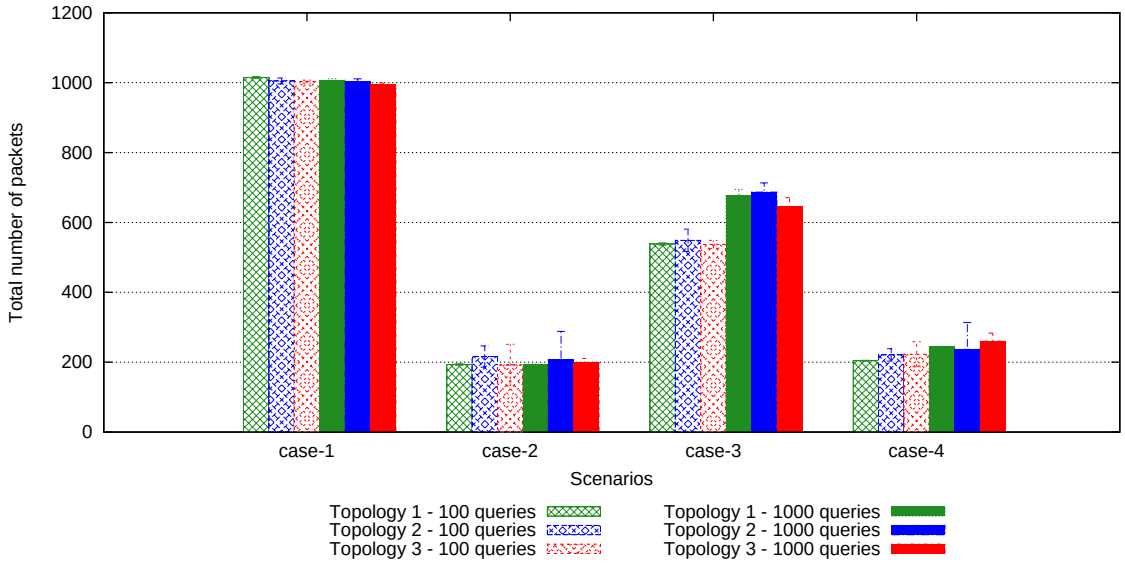


Figure 10.10: Application-level packets received at the DA for topologies 1-3 with the different number of queries

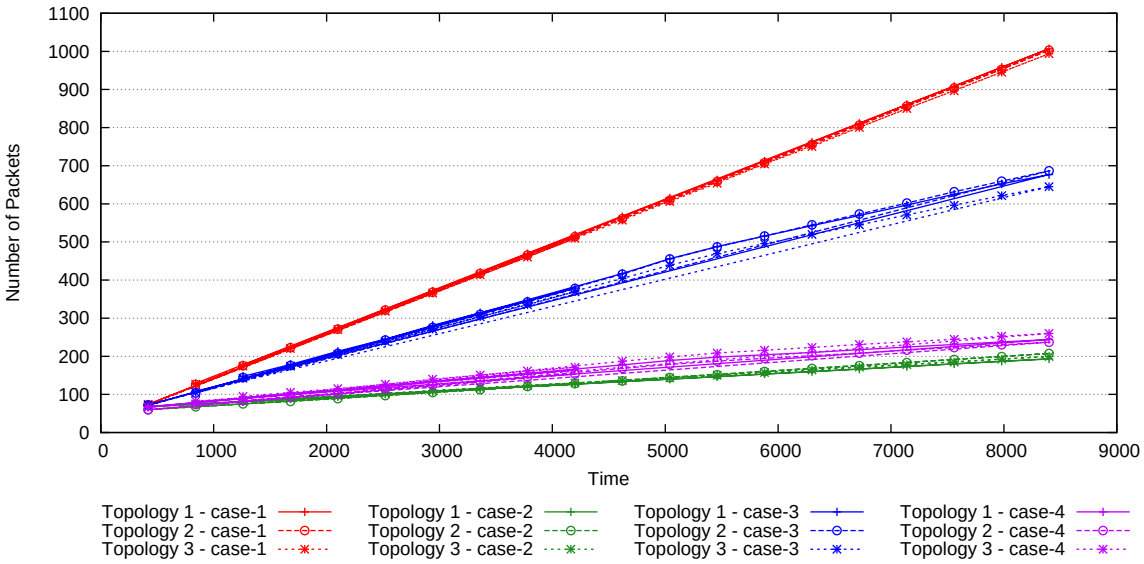


Figure 10.11: Application-level packets received at the DA over the time for topologies 1-3 with the different number of queries

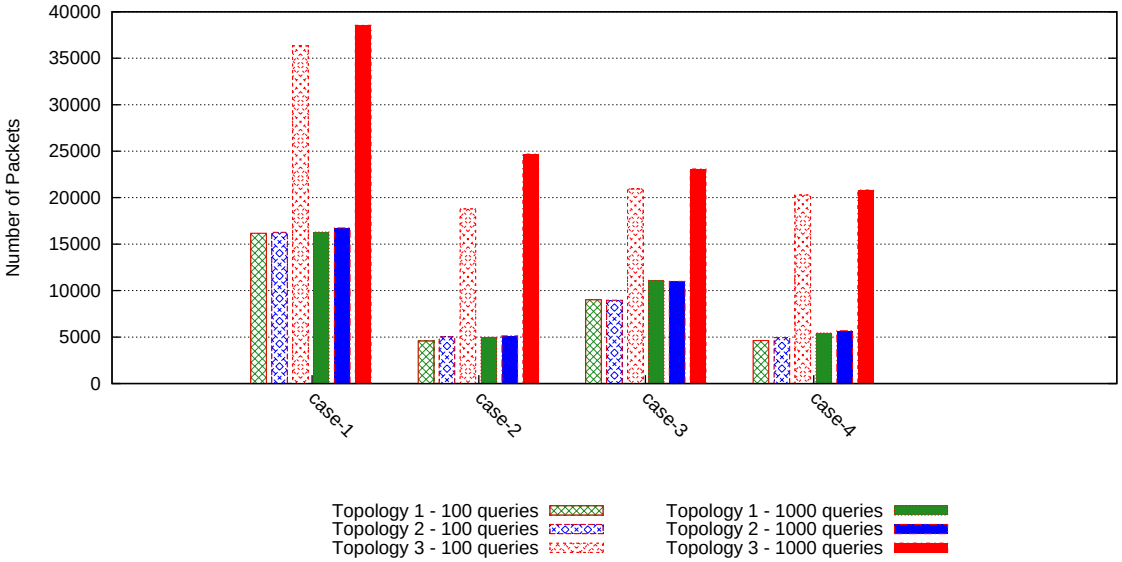


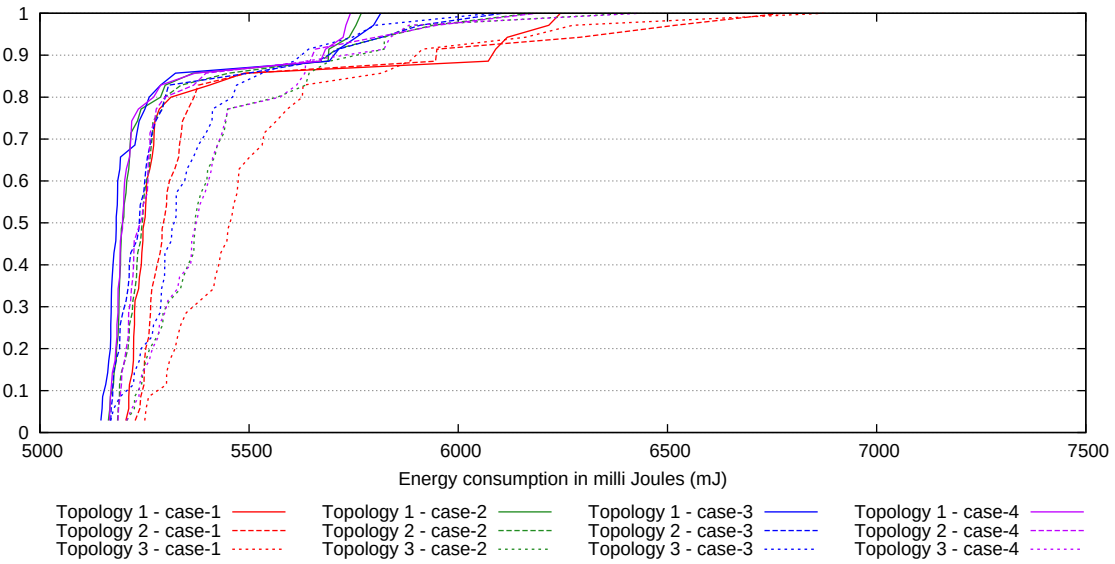
Figure 10.12: Total traffic at the DA for topologies 1-3 with the different number of queries

10.3.5 Energy Consumption and Network Lifetime

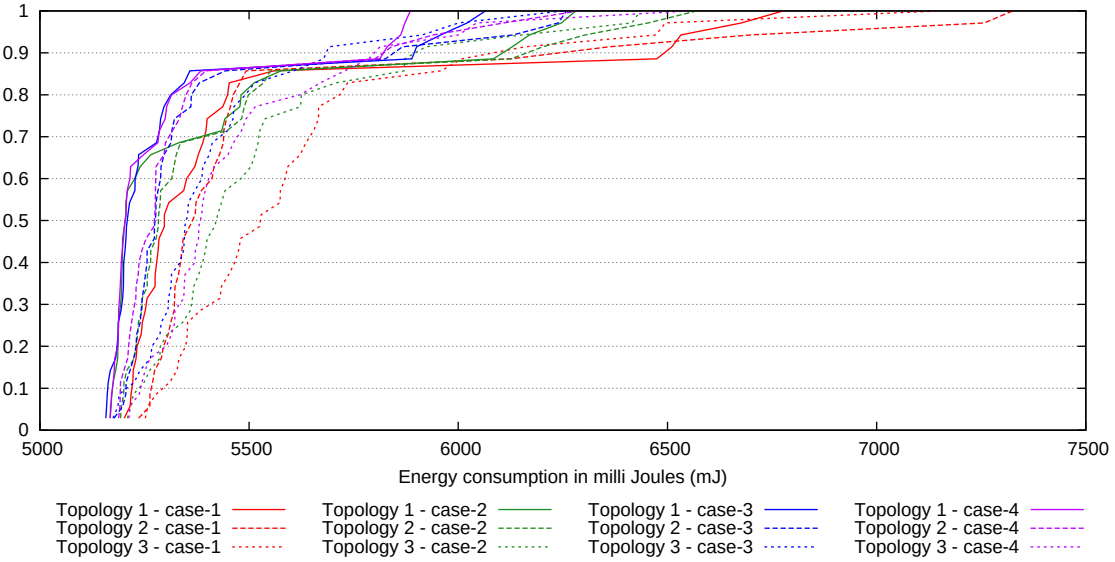
The individual energy consumption for all nodes is shown in Figure 10.13 for scenarios with 100 and 1000 queries. Overall, nodes saved energy when TRENDY techniques are enabled for all topologies. In general, topology 3 performed worst in energy consumption, whereas topology 1 remained the best one. In case of topology 3, case 3 proved to be the best case, because in cases 2 and 4, grouping

affects the energy consumption for 10% of the nodes, which are acting as GLs. It is also evident that nodes consume more energy in scenarios with 1000 queries; however, APPUB, in cases 3-4, saves more energy by offering cached values and brings the energy consumption close to the results of scenarios with 100 queries.

Figure 10.14 shows the top five nodes in energy consumption for all scenarios. It suggests that cases with TRENDY techniques will increase the network lifetime for all topologies. However, grouping has affected the network lifetime for topology 3 by increasing the energy consumption of top nodes in cases 2 and 4. This again points out the impact of incongruent grouping and RPL topologies. The network lifetime approximations in Table 10.2 depicts that the network lifetime increases about 10-15% when all TRENDY techniques are employed, and more efficiency is gain for busier network in case of 1000 queries. It's important to notice that grouping increased the lifetime for topology 1 and remained neutral for topology 2, whereas it decreases the network lifetime for topology 3. Therefore, grouping maintained the same behaviour as discussed in Section 8.8.3.



(a) Scenarios with 100 queries



(b) Scenarios with 1000 queries

Figure 10.13: CDF graph of energy consumption per node for topologies 1-3

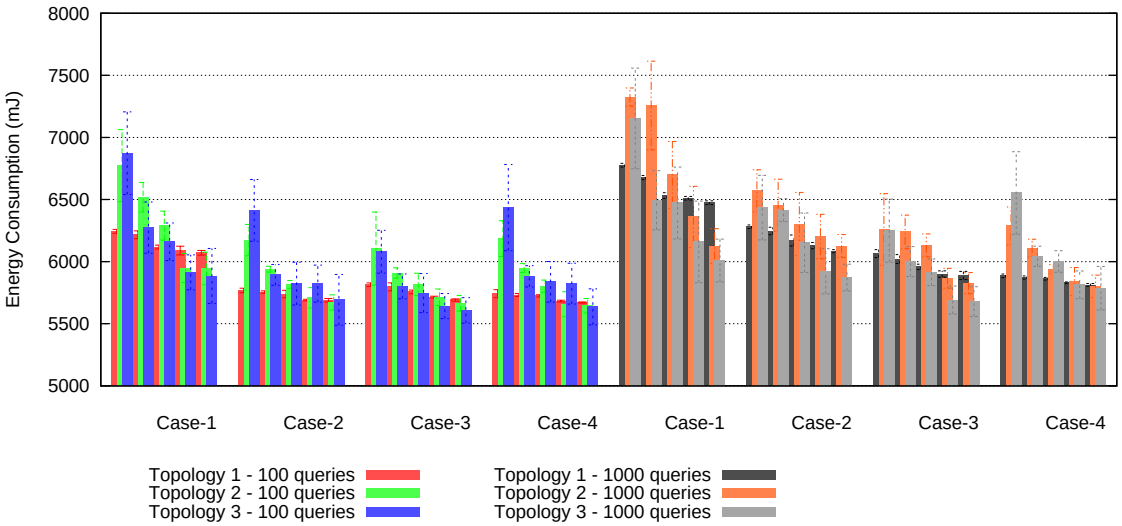


Figure 10.14: Top five nodes in energy consumption for topologies 1-3 with the different number of queries

Table 10.2: Network lifetime approximation for topologies 1-3 with the different number of queries

Network Lifetime (days)						
Technique	**Topology #1**		**Topology #2**		**Topology #3**	
	100	**1000**	**100**	**1000**	**100**	**1000**
Basic TRENDY	420	385	385	355	380	365
With timer and grouping	445	415	425	395	410	405
With timer and APPUB	445	430	425	415	430	420
With all mechanisms	455	440	425	415	405	400

10.3.6 Memory Requirements

The COOJA supports hardware-level simulations (Section 2.7.2); therefore, Tmote Sky devices were emulated as SAs in all scenarios, and code is compiled with MSP430 GCC compiler. This enables the same implementation to work directly on the hardware as well. The size of the code is presented by `dec`, which is further divided into `data`, `bss` and `text`. The `data` portion of the program contains defined constants and initialised variables. The `bss` (Block Started Symbol) part represents those variables that remain uninitialised in the start. Finally, `text` contains the program instructions to be executed.

Table 10.3 shows the memory requirements for different features of TRENDY. Erbium [70] (C implementation of CoAP) is downsized by eliminating extra options and features to fit on the devices, but still requires around 6KB. The size of three

CoAP resources also include the extra driver code required to enable the sensors on the device. The APPUB algorithm has consumed around 1KB of memory, because a SA is the main decision maker (Section 9.3). Overall, TRENDY can be implemented by a GM with all algorithms at cost of 1.7KB, and for a GL, this cost will increase to 3.2KB (to manage 12 GMs without APPUB). In summary, the compiled code with CONTIKI, CoAP, and other TRENDY features is compact enough to directly run on Tmote Sky hardware, which has only 64KB of memory.

Table 10.3: Size of TRENDY protocol: Compiled with MSP430 GCC Compiler

Feature	text	data	bss	dec
CoAP (Downsized version)	5552	32	324	5908
3 sensors with CoAP resources	1892	56	8	1956
TRENDY basic	1328	0	206	1534
Timer	94	0	0	94
APPUB	942	0	50	992
SA as GM	130	18	0	148
SA as GL	1382	44	294	1720

10.4 Summary and Discussion

This chapter scrutinised the effect of using the combination of proposed timer, grouping and APPUB techniques with TRENDY. It required a change in a message format and new policies while using the techniques together. The roles of a SA now vary to a GM or GL with different implemented techniques.

The analysis of several experiments concludes that the TRENDY techniques maintain their individual efficiency, and total performance improves with the aggregation of all their benefits. The timer mechanism improves the control overhead both at application layer and in the whole network, which also reduce the service invocation delay in a dense network. APPUB adapts to the increasing demand of the service invocation requests and consequently, reduces the delay while decreasing the traffic load. However, if a network is not busy enough, then more sensitive APPUB can become a burden on the network. Therefore, the sensitivity of its threshold value should be selected carefully, as in a multi-hop network of heterogeneous devices, which is a norm in the IoT. Hence, the choice of one device can affect the sleeping schedule and ultimately energy consumption of other nodes. In experiments, it remains fixed for all nodes, but APPUB design allows a node to dynamically change the sensitivity-level depending on its constraints and network dynamics. The grouping mechanism localises the traffic in different vicinities and consequently, improves the availability of the link towards the sink. However, it

is concluded from the analysis of the results that it actually makes the situation worse for GLs, if grouping topology is incongruent to RPL topology. Anyway, this can't be an issue in those scenarios where a GL has better battery source and is a resource-rich device. Even though, the results of multiple topologies convey that if RPL's parameters are configured administratively to create more congruent DODAG, then situation becomes suitable for grouping to improve the performance of a network.

The next chapter compares the TRENDY SD solution with other prospective techniques from different perspectives.

Chapter 11

Comparison with other solutions

11.1 Introduction

There is a plethora of SDPs devised for Ad-Hoc, MANET, WSN and IP networks (Chapter 3). Industry-standard IP-based SDPs including SLP, UPnP, JINI and Salutation are not directly applicable to 6LoWPANs because of their verbose and complex data formats, high communication overhead and dependence on protocols, which are unsuitable for constrained networks. On the other hand, the solutions designed for constrained networks just focus on increasing the efficiency for a single application (Section 3.9). Therefore, most of these existing standards do not address the issues which need a solution to balance the trade-off between the requirements of the IoT and efficiency for constrained domains (Chapter 4). Consequently, this research project proposes TRENDY to provide context-aware SD and selection. Moreover, it devises the timer, grouping and APPUB techniques for adaptability to network dynamics to cope with the challenges.

This chapter covers the functionality based comparison of the proposed solution with other suitable options for 6LoWPANs. Two protocols are selected for comparison, which are the adaptations of existing widely used IP-based standards: uBonjour [64] (Bonjour Adaptation) and SSLP-based [18] (SLP Adaptation) SD solutions. These adaptations have considered constraints of 6LoWPANs and have reduced the complexity, size and overhead of existing protocols. Furthermore, those solutions are backward compatible to their complex versions, but need a gateway for such a translation. Another option could be the IETF Resource directory which is closely related to TRENDY. However, it is still in a design phase, and can benefit from the proposed techniques to enhance its performance. The proposed solution focused on addressing the key requirements of efficiency, adaptability and context-awareness. This chapter qualitatively compares the proposed solution with the selected protocols from different perspectives related to SD and its requirements

in IoT environment.

11.2 Context-awareness

Context-awareness is an important feature that enables a user application to define a better SD query to search the relevant services in the IoT. Furthermore, it can enable a SD solution to assist the user application to select a better service based on available context information.

uBonjour: uBonjour has no support for context-awareness. Thus, it only allows the query with a fixed DNS based URL.

SSLP-based solution: This solution is extended [19] to enable context-awareness, but no detail of context registration has been described. However, the idea allows the user agents to send context-aware SD queries. Its service selection only considers proximity (close in hops) as a service selection criteria.

TRENDY: This SDP has focused on context-awareness by defining a default simple attribute-value pair format to describe context attributes (Section 5.5); however, it allows the employment of any other semantic format as well. TRENDY allows a discovery query to define service types, location and other relevant information (e.g., semantic information). Its service selection mechanism (Section 5.8) uses maintained context information including, battery source, distance (number of hops), remaining power and popularity to enable the discovery of an appropriate service based on user and network's context.

11.3 Extensibility

The IoT environment demands extensible solution to accommodate a range of heterogeneous devices (resource-awareness) and new semantic formats for service descriptions. Following is the comparison of SD solutions from the extensibility perspective:

uBonjour: uBonjour uses fixed service description format by only allowing DNS URI to describe a service. It has not described any specific roles for heterogeneous devices.

SSLP-based solution: SSLP-based solution uses fixed SLP service types as service descriptions. However, it describes two profiles for nodes: SA and

master node. The master nodes serves as local registries and localises the information by enabling a hierarchical architecture.

TRENDY: TRENDY has a simple service description format, but allows the usage of any other format (Section 5.4). It proposes different roles with modular design to enable nodes to implement distinctive features depending on their capabilities (Section 10.2.2).

11.4 Interoperability

Interoperability of protocols and systems is important to allow their usage with other existing solutions.

uBonjour: uBonjour uses DNS-SD to offer interoperability with DNS infrastructure, but it has not defined an efficient way to maintain a central directory while employing mDNS inside the network.

SSLP-based solution: SSLP-based solution offers interoperability with SLP; however, it requires an application gateway for translation between protocol.

TRENDY: TRENDY employs CoAP that enables the RESTful web service paradigm to offer an open and interoperable framework (Section 10.2.4). It allows the discovery using either CoAP or HTTP, and central registry can be updated to offer interoperability by passing services in many formats, including, WSDL and DNS-URIs (Section 5.11.3).

11.5 Constraints Considerations

Following is the list of the constraints discussed for each of the protocols being compared:

Devices with sleep cycles: uBonjour uses mDNS and multiple packets for service registration and status updates, which needs more data to be passed on in a multi-hop network and thus is an unsuitable choice for devices with sleep cycles. SSLP-based solution utilises unicast for discovery, but still requires service information to be communicated between different local directories and the DA. This demands high-bandwidth usage, as service status maintenance is required at multiple places. TRENDY addresses this challenge in multiple ways: it uses unicast and timer mechanism to decrease the control overhead, and its APPUB technique allows the nodes to sleep more even in case of high number of service invocations.

Code and Packet Size: All protocols have considered 6LoWPANs code and packet size issues (Section 4.4.4), and can run on constrained devices with compact packet size.

11.6 Dependencies

Following is the dependency list of all protocols being compared:

uBonjour: uBonjour requires multicast support.

SSLP-based solution: This needs an application gateway to translate between SLP and SSLP messages. In addition, local master nodes are required to be deployed in each area.

TRENDY: This protocol requires CoAP-to-HTTP seamless proxy where a UA wants to use HTTP for service invocation (Section 10.2.4). Furthermore, some nodes are required to act as GLs to reap the benefits of the grouping mechanism.

11.7 Performance Metrics

The performance metrics for a SD solution consists of: Service Discovery delay, Service Invocation delay, energy consumption, scalability in terms of protocol's architecture and control overhead (Section 6.2). This section compares TRENDY with other solutions from these metrics perspectives.

11.7.1 Service Discovery delay

A UA mostly queries from outside the network in a IoT environment, which needs a centralised registry to deal with SD queries. Thus, in that case, all solutions will perform similarly; however, the service discovery delay will be variable for queries that are initiated from within a network as these solutions use different architectures (Section 11.7.3).

11.7.2 Service Invocation support

A SD solution can support different levels of service invocation (Section 3.10.5). TRENDY provides service invocation support, and its APPUB technique provides cached values to improve the service invocation delay. On the other hand, none of the compared techniques support service invocation other than finding a host device.

11.7.3 Scalability

The scalability of a SD solution depends upon the architecture of a protocol and its behaviour in response to the increasing number of nodes and services within a network in terms of control overhead. The control overhead of a protocol is the sum of all service registration, advertisement and status update messages.

In a centralised architecture, all nodes register their services at the central registry, whereas in a distributed directory based architecture, all services are registered to local registries. On the other hand, directory-less solutions allow nodes to pro-actively advertise their services within vicinity, which are cached by their neighbours for a quick SD response to a user agent. This cached service information at a registry or neighbour node requires status maintenance to ensure the reliability of SD responses, which is achieved by refreshing the status of registered services. However, this increases the control overhead of a protocol, and result in extra energy consumption of nodes and can also cause a bottleneck problem in centralised architectures.

This section analyses the scalability in the context of control overhead of SDPs including, uBonjour, SSLP-based solution and TRENDY.

uBonjour: uBonjour uses a directory-less approach, which generates high multicast traffic to announce services in a vicinity that are cached by the neighbours. However, it still needs a centralised registry, e.g., service proxy [101] or a mDNS daemon [64] to allow a user application from outside the network to efficiently discover a service. In uBonjour, a node needs to send 2-4 packets [64, 65] for the registration of a single service. This adds extra burden on the network as multicast is used to announce services inside the network. The effect is multiplied in case of status maintenance as it will require the same number of status updates for all advertised services. Therefore, uBonjour entails heavy traffic for service registration and status maintenance in a IoT scenario that hosts 100s of services, or otherwise it will increase service discovery delay tremendously in absence of an service advertisement mechanism.

SSLP-based solution: SSLP-based solution has a centralised architecture; however, it devises DPAs (Directory Proxy Agents) based local registries to enable distributed architecture. In this solution, all host devices send their registrations to dedicated DPAs in their vicinity that forwards the information towards the centralised registry after storing it in its registry. Its packet format is compact, which requires one packet for each service. However, all SAs register their services at a DPA in the vicinity, which forwards this

cache to the centralised registry. Similarly, the status updates are done by re-registering services to maintain the soft values at both DPAs and centralised directory. This process doubles the control overhead for service registration and status maintenance.

TRENDY: TRENDY has a centralised architecture, but it also proposes grouping mechanism to localise the traffic in a specific area depending on the physical location of nodes. In TRENDY, nodes declare their implemented roles while registering at the DA, and later some nodes are selected by the DA to act as group leaders (GLs) for their vicinities. The role of a GL varies as it depends on the the capability of a node. Load balancing policy allows GLs to take turns to prolong the network's lifetime. The grouping mechanism enables distributed behaviour of the architecture by localising information in different groups. Each SA can register multiple services to the centralised registry by using one registration message (Section 5.11.1). In experiments (Section 10.3.1), three service are registered together with context attributes in a single 6LoWPAN packet. TRENDY describes a single 6LoWPAN packet to update the status of all services hosted by a SA (Section 5.11.2). Furthermore, it introduces a demand-based adaptive timer that further reduces the control overhead (Section 7).

11.7.4 Energy Consumption

Energy consumption in a constrained network depends on the employed RDC mechanism and control overhead of a protocol. High control overhead means that radio will be switched on more often to send and receive control messages between nodes, which translates into more energy consumption. This causes a ripple effect in case of a multi-hop network, where more nodes will need to be awake to pass the messages towards the receiver node. Section 11.7.3 has described that how TRENDY will require fewer numbers of control packets compared to other protocols. Furthermore, uBonjour uses multicast for service registration and advertisement, which require most of the devices to listen and forward to the messages not intended for them that further increases the energy overhead. Thus, nodes will consume less energy when TRENDY will be used.

11.8 Summary

Table 11.1 presents the comparison of uBonjour, SSLP-based solution and the proposed TRENDY solution from different aspects. In summary, the proposed

solution offers a better context-aware, adaptive and efficient SD solution compared to other protocols.

Table 11.1: Comparison of SDPs for the IoT

	uBonjour	**SLP adaptation**	**TRENDY**
Context-Aware	No	Yes	Yes
Service Description	Fixed	Fixed	Any
Discovery Scope	Local	Local	Local
Interoperability	DNS-SD	SLP with gateway	CoAP, and HTTP with seamless proxy
Dependencies	Multicast	Application gateway	Proxy for HTTP
Architecture	Service cache on each node	Multiple directories	One or many directories
Registration	2-4 messages for each service	1 message for each service	1 message for multiple services
Status maintenance	Same as registration	Same as registration	1 message for all services
Adaptive status maintenance	No	No	Adaptive timer mechanism
Discovery query	DNS URI	Service type with scopes	Service type and context information
Service selection	No	Yes (number of hops)	Yes (multiple attributes)
Service invocation	No	No	Yes
Caching	No	No	Yes
Adaptive Caching	No	No	Yes (APPUB)
Support for sleepy nodes	No	No	APPUB
Resource-aware	No	Yes (2 profiles)	Yes (Multiple profiles)
Message size	Fits in a packet	Fits in a packet	Fits in a packet

Chapter 12

Conclusions and Future Work

This research project is focused on providing an efficient, context-aware and adaptive SD solution for the IoT. In this chapter, the completed work is summarised and the potential future work is discussed.

12.1 Conclusions

The IoT paradigm has introduced new perspective to be realised for the future Internet. Isolated islands of WSNs are now enabled to merge with the Internet by new technologies, such as 6LoWPAN. This merger allows IP based connectivity of constrained networks with the Internet. SD and service selection are the key to change the way we perceive constrained networks. Furthermore, web services can provide a standard interface to offer interoperability with other existing similar solutions. However, the design of such a solution faces the underlying challenges of these networks, for example; small packet size and sleep cycles, etc. On one hand, the sleep cycles demand the registry based architecture. On another hand, the limited bandwidth and energy required to decrease the control overhead in such networks. Services are mostly hosted on battery operated devices, which are prone to failure. Therefore, status maintenance is required in case of registry based solution, which increases the bandwidth utilisation and energy consumption. All these challenges require a solution that deals with the heterogeneity and interoperability requirements of the IoT, while addressing the constraints posed by WSNs.

This research project has proposed TRENDY: an adaptive and context-aware SD solution for the IoT. To deal with the interoperability, this solution employs CoAP-based RESTful web services, which enable application-layer integration of constrained domains and the Internet. Moreover, the context-aware service selection mechanism assists users to discover appropriate services by using available

user- and network-based context. The accuracy of service information is ensured by keeping soft values at the DA and maintaining the status maintenance.

The trade-off between status maintenance load and reliability is managed by a demand-based adaptive timer. This timer mechanism is even beneficial in scenarios where the basic interval for status updates is an hour or more, as the timer's maximum counter limit can be configured to increase the interval adaptively. A IoT environment can have peaked usage times where a constrained network can easily be overwhelmed by the increased demand of its resources.

The research project devises the APPUB technique to deal with such situations by enabling service hosts to share their load with the resource directory. Therefore, SAs push the cache values of busy services towards the registry, which decreases the service invocation delay because of the cache-hits. The DA ensures the accuracy of cached information by using the respective cache lifetime values to send only valid cached values.

The proposed context-aware grouping technique divides the network at the application layer, by creating location-based groups. This grouping of nodes localises the control overhead and provides the base for service composition, localised aggregation and processing of data. The simulation results show that TRENDY's techniques decrease the control overhead, energy consumption and service invocation delay. Additionally, the grouping technique considerably decreases the number of packets towards the sink and thus improves scalability in a multi-hop network. However, the performance of the grouping depends on the congruity between the grouping and routing topologies.

Following is a brief summary about the features of the proposed solution:

Web Service Paradigm: TRENDY uses a RESTful web service paradigm. Entities use either CoAP (default) or HTTP (in case the targeted host understands it or DA is acting as a proxy) to define their services and to communicate with each other. The use of CoAP/HTTP simple proxy can seamlessly translate requests from both protocols. This blends the real-world devices into existing web and enables the WoT paradigm.

Context-awareness: In TRENDY, the DA stores all service and contextual information, including service descriptions, location, battery consumed, and registration time for all registered nodes. Furthermore, it maintains a hit counter for each service, which is incremented whenever a service is discovered and selected. The maintained context information is used by the service selection and better GL selection mechanisms.

Grouping: Context-aware grouping serves several purposes, including simple localisation of status maintenance, execution of group-based queries to offer

an optional local service repository. It costs in terms of some packet overhead. However, networks can get the benefit in the form of localised communication, which conserves energy. In addition, this enables a DA to compose and offer group-based services, e.g., to actuate a command in a certain area. The experiments (Section 8.8) demonstrate the benefit of grouping in terms of scalability and energy efficiency, while slightly increasing the control overhead.

Hybrid architecture: Basically, TRENDY has a centralised architecture that converges to a distributed one when the DA uses context information to group GMs.

Service Management: All service records have soft states at the DA, therefore, needs to be updated depending on the reliability and other requirements. TRENDY introduces an adaptive timer to increase or decrease the update interval depending on the demand of a service. This marginally decreases the bandwidth by reducing the number of update messages. The experiments (Section 7.5) show that the employment of timer mechanism reduces the control overhead and consequently, improves the energy efficiency and scalability of TRENDY.

Service Discovery: The DA determines the matching service from the registry using the attributes of the UA request. Subsequently, it responds back to the UA by appending the service information (resource's URL and IP address of the host) of one or more matching services in the payload.

Service Selection: A UA can ask for the DA's assistance in selecting the best matching host if multiple prospective hosts are found. In this case, the DA determines the most appropriate service (if multiple services have been discovered) using available user and network context information, e.g., battery, hops count, UA location, etc. The simulation experiments (Section 6.5) show that service selection lowers the service invocation delay while improving the energy efficiency as well.

Service Invocation: SD is completed when an application gets the response with a service identifier and address of its host. TRENDY, however, enables service invocation using a RESTful web service interface and takes a step ahead by defining APPUB an adaptive caching technique. The simulations (Section 9.5) supports the point that APPUB reduces the service invocation delay closer to the NullRDC performance for ContikiMAC scenarios.

12.2 Future Work

Following are few dimensions and recommendations based on the discussions from previous chapters, which can be consider for potential future work:

Extension to the protocol's functionality: TRENDY has considered different aspects of the IoT by offering a context-aware SD solution; however, following aspects can be further investigated to allow a broader applicability of the protocol.

- **Dynamic Service mashups:** The context-awareness of TRENDY enables the DA to understand different aspects of services and devices in a network. Furthermore, its grouping mechanism changes the DA's perspective of the network. After grouping the nodes, the DA can disseminate a command in an area by passing one command to all GLs in the area. This platform allows a DA to mashup services to create higher-level services. The IoT scenarios (Section 4.2) can be better served with service mashups; for example, a user can execute one command for the whole area without searching and executing services individually. In addition, the DA can smartly offer related services by using user's context details. Therefore, an investigation can be done to define, implement and analyse an extension to grouping mechanism to create service mashups dynamically using user's and network's context information.
- **Semantic discovery:** An investigation to improve context-awareness by analysing the feasibility to use compact and rich formats to enable semantics based discovery.
- **Adaptability of timer:** The adaptability of a timer currently considers the number of times a node has sent updates and the demand of its services. An investigation can consider other attributes e.g., node's energy etc. to analyse the behaviour of the timer and its efficiency.
- **Decision making in APPUB :** Currently, SAs solely decide about pushing the cache at the DA, this can be changed for some scenarios to find out the impact of pull-based approach where a DA requests for cache values to serve queries.

Expansion of experimental analysis: Several experiments are conducted using different RDC, number of queries and topologies. Following are some recommendations to investigate and broaden the experiments:

- **RPL configurations and performance:** The impact of RPL's DODAG has been extensively mentioned in experimental analysis. An investigation to change the RPL's OF to better suit the network dynamics can be conducted.
- **Different routing protocols:** Other routing protocols can be employed to analyse their impact on the performance of TRENDY.
- **Mobility:** Mobile nodes are common in IoT environments, as local mobile phones can become a part of the network to use its services and also share their services (hosted sensors) with the registry. Therefore, an investigation to find out TRENDY's behaviour scenarios can be investigated and can be improved.
- **Compare multicast:** The group queries can benefit from multicast technique [97] to execute group based commands. However, multicast is costly so an investigation can be conducted to compare available multicast techniques and test their applicability to execute group-based queries.
- **Large-scale Network:** Another perspective could be to design the collaboration of different small networks to form a large-scale SD system. Moreover, the solution can be deployed in a real-life scenario to materialise the discussed scenarios in Section 4.2.
- **Searching in a large-scale network:** Another dimension is searching the objects on the Internet using a search engine so the important questions for a future work in a broader perspective include: how services offered by one network can be made available to the existing DNS infrastructure or what context or semantic information will be required to facilitate the user to discover appropriate services using a search engine (related work [24, 85]).

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *ACM SIGOPS Operating Systems Review*, volume 33, pages 186–201. ACM, 1999.

[2] R. Ahmed, R. Boutaba, F. Cuervo, Y. Iraqi, D. Li, N. Limam, J. Xiao, and J. Ziembicki. Service discovery protocols: A comparative study. In *Proceedings of IM*, pages 15–18, 2005.

[3] H. Alex, M. Kumar, and B. Shirazi. Service discovery in wireless and mobile networks. *Wireless information highways*, page 251, 2005.

[4] H. Alex, M. Kumar, and B. Shirazi. Midfusion: An adaptive middleware for information fusion in sensor network applications. *Information Fusion*, 9(3):332–343, 2008.

[5] Z.B. Alliance. Zigbee specification. *ZigBee document 053474r06, version*, 1:378, 2006.

[6] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009.

[7] Fatima Muhammad Anwar, Muhammad Taqi Raza, Seung-Wha Yoo, and Ki-Hyung Kim. Enum based service discovery architecture for 6lowpan. In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, pages 1–6. IEEE, 2010.

[8] K. Arabshian and H. Schulzrinne. Gloserv: Global service discovery architecture. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 319–325. IEEE, 2004.

[9] Ken Arnold, Robert Scheifler, Jim Waldo, Bryan O'Sullivan, and Ann Wollrath. *Jini Specification*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[10] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[11] M. Balazinska, H. Balakrishnan, and D. Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. *Pervasive Computing*, pages 149–153, 2002.

[12] SIG Bluetooth. Bluetooth specification version 1.1. *Available HTTP: http://www. bluetooth. com*, 2001.

[13] Carsten Bormann, Angelo P Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *Internet Computing, IEEE*, 16(2):62–67, 2012.

[14] S Bradner, L Conroy, and K Fujiwara. The e. 164 to uniform resource identifiers (uri) dynamic delegation discovery system (ddds) application (enum). *Internet Request for Comments, vol. RFC*, 6116, 2011.

[15] J. Buford, B. Burg, E. Celebi, and P. Frankl. Sleeper: A power-conserving service discovery protocol. In *Mobile and Ubiquitous Systems-Workshops, 2006. 3rd Annual International Conference on*, pages 1–9. IEEE, 2006.

[16] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. Gsd: A novel group-based service discovery protocol for manets. In *Mobile and Wireless Communications Network, 2002. 4th International Workshop on*, pages 140–144. IEEE, 2002.

[17] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. Toward distributed service discovery in pervasive computing environments. *Mobile Computing, IEEE Transactions on*, 5(2):97–112, 2006.

[18] Shafique Ahmad Chaudhry, Won Do Jung, Chaudhary Sajjad Hussain, Ali Hammad Akbar, and Ki-Hyung Kim. A proxy-enabled service discovery architecture to find proximity-based services in 6lowpan. In *Embedded and Ubiquitous Computing*, pages 956–965. Springer, 2006.

[19] S.H. Chauhdary, M.Y. Cui, J.H. Kim, A.K. Bashir, and M.S. Park. A context-aware service discovery consideration in 6lowpan. In *Third 2008 International Conference on Convergence and Hybrid Information Technology*, pages 21–26. IEEE, 2008.

[20] Y. Chen and Q. Zhao. On the lifetime of wireless sensor networks. *Communications Letters, IEEE*, 9(11):976–978, 2005.

[21] S. Cheshire and M. Krochmal. Multicast dns. Technical report, IETF, 2006.

[22] C. Cho and D. Lee. Survey of service discovery architectures for mobile ad hoc networks. *term paper, Mobile Computing, CEN*, 5531, 2005.

[23] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, et al. Web services description language (wsdl) 1.1, 2001.

[24] Benoit Christophe, Vincent Verdot, and Vincent Toubiana. Searching the'web of things'. In *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*, pages 308–315. IEEE, 2011.

[25] Walter Colitti, Kris Steenhaut, Niccolò De Caro, Bogdan Buta, and Virgil Dobrota. Evaluation of constrained application protocol for wireless sensor networks. In *Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, pages 1–6. IEEE, 2011.

[26] J. Coutaz, J.L. Crowley, S. Dobson, and D. Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[27] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86–93, 2002.

[28] A.K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[29] A. Dunkels. Full tcp/ip for 8-bit architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 85–98. ACM, 2003.

[30] A. Dunkels, J. Alonso, and T. Voigt. Making tcp/ip viable for wireless sensor networks. *SICS Research Report*, 2003.

[31] A. Dunkels, B. Gronvall, and T. Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462. IEEE, 2004.

[32] Adam Dunkels. The contikimac radio duty cycling protocol. Technical Report T2011:13, Swedish Institute of Computer Science, 2011.

[33] Adam Dunkels, Fredrik Osterlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 28–32. ACM, 2007.

[34] P.E. Engelstad, Y. Zheng, R. Koodli, and C.E. Perkins. Service discovery architectures for on-demand ad hoc networks. *International Journal of Ad Hoc and Sensor Networks*, 1(3), 2005.

[35] Muhammad Omer Farooq and Thomas Kunz. Operating systems for wireless sensor networks: a survey. *Sensors*, 11(6):5900–5930, 2011.

[36] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[37] J. Gao and P. Steenkiste. Rendezvous points-based scalable content discovery with load balancing. *Proceedings of NGC 2002*, 2002.

[38] J.A. Garcia-Macias and D.A. Torres. Service discovery in mobile ad-hoc networks: better at the network layer? In *Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on*, pages 452–457. IEEE, 2005.

[39] R.S. Gopinath, I. Khan, and Z. Suryady. Optimized web service architecture for 6lowpan. In *Information Networking, 2009. ICOIN 2009. International Conference on*, pages 1–3. IEEE, 2009.

[40] IETF Zeroconf Working Group et al. Zero configuration networking (zeroconf).

[41] M. Gudgin, N. Mendelsohn, M. Nottingham, and H. Ruellan. Xml-binary optimized packaging. *W3C Recommendation, Jan*, 2005.

[42] Dominique Guinard. *A Web of Things Application Architecture – Integrating the Real-World into the Web*. Ph.d., ETH Zurich, 2011.

[43] Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, 2009.

[44] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *Services Computing, IEEE Transactions on*, 3(3):223–235, 2010.

[45] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*, pages 97–129. Springer, 2011.

[46] E. Guttman. Service templates and service: Schemes. Technical report, IETF, 1999.

[47] E. Guttman, C. Perkins, J. Veizades, and M. Day. Slpv2: Service location protocol. Technical report, Version 2. RFC 2608, Jun, 1999.

[48] G. Halkes, A. Baggio, and K. Langendoen. A simulation study of integrated service discovery. *Smart Sensing and Context*, pages 39–53, 2006.

[49] H. Hassanein, Y. Yang, and A. Mawji. A new approach to service discovery in wireless mobile ad hoc networks. *International Journal of Sensor Networks*, 2(1):135–145, 2007.

[50] S. Helal, N. Desai, V. Verma, and C. Lee. Konark-a service discovery and delivery protocol for ad-hoc networks. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 2107–2113. IEEE, 2003.

[51] A. Helmy, S. Garg, N. Nahata, and P. Pamu. Card: a contact-based architecture for resource discovery in wireless ad hoc networks. *Mobile networks and applications*, 10(1):99–113, 2005.

[52] Robert M Hinden and Stephen E Deering. Ip version 6 addressing architecture. Technical report, IETF, 2006.

[53] Q. Huaifeng and Z. Xingshe. Context aware sensornet. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–7. ACM, 2005.

[54] M.C. Huebscher and J.A. McCann. Adaptive middleware for context-aware applications in smart-homes. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 111–116. ACM, 2004.

[55] J. Hui and P. Thubert. Compression format for ipv6 datagrams in 6lowpan networks. *draft-ietf-6lowpan-hc-04 (work in progress)*, 2008.

[56] Jonathan W Hui and David E Culler. Ip is dead, long live ip for wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 15–28. ACM, 2008.

[57] Bret Hull, Kyle Jamieson, and Hari Balakrishnan. Mitigating congestion in wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 134–147. ACM, 2004.

[58] Philipp K Janert. *Gnuplot in action: understanding data with graphs*. Manning Publications Co., 2009.

[59] C. Jardak, E. Meshkova, J. Riihijarvi, K. Rerkrai, and P. Mahonen. Implementation and performance evaluation of nanoip protocols: Simplified versions of tcp, udp, http and slp for wireless sensor networks. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pages 2474–2479. IEEE, 2008.

[60] Miloš Jevtić, Nikola Zogović, and Goran Dimić. Evaluation of wireless sensor network simulators. In *Proceedings of the 17th Telecommunications Forum (TELFOR 2009), Belgrade, Serbia*, pages 1303–1306, 2009.

[61] Patrick Olivier Kamgueu, Emmanuel Nataf, Thomas Djotio Ndié, and Olivier Festor. Energy-based routing metric for rpl. Rapport de recherche RR-8208, INRIA, Jan 2013.

[62] James Kempf and Jason Goldschmidt. Notification and subscription for slp. Technical report, RFC 3082, March, 2001.

[63] K Kim, S Yoo, H Lee, S Daniel Park, and J Lee. Simple service location protocol (sslp) for 6lowpan. *drraft-daniel-6lowpan-sslp-00*, 7, 2005.

[64] Ronny Klauck and Michael Kirsche. Bonjour contiki: a case study of a dns-based discovery service for the internet of things. In *Ad-hoc, Mobile, and Wireless Networks*, pages 316–329. Springer, 2012.

[65] Ronny Klauck and Michael Kirsche. Enhanced dns message compression Âŋoptimizing mdns/dnsÂŋsd for the use in 6lowpans. In *Proceedings of the 9th International Workshop on Sensor Networks and Systems for Pervasive Networks (PerSeNS 2013), co-located with the 11th IEEE International Conference on Pervasive Computing and Communications (PerCom 2013)*, March 2013.

[66] M. Klein, B. König-Ries, and P. Obreiter. *Lanes: A Lightweigth Overlay for Service Discorvery in Mobile Ad Hoc Networks*. Univ., Fak. für Informatik, 2003.

[67] M. Klein, B. Konig-Ries, and P. Obreiter. Service rings-a semantic overlay for service discovery in ad hoc networks. In *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pages 180–185. IEEE, 2003.

[68] R. Koodli and C.E. Perkins. Service discovery in on-demand ad hoc networks. *IETF draft*, 2002.

[69] Aleksandar Kovacevic, Junaid Ansari, and Petri Mahonen. Nanosd: A flexible service discovery protocol for dynamic and heterogeneous wireless sensor networks. In *Mobile Ad-hoc and Sensor Networks (MSN), 2010 Sixth International Conference on*, pages 14–19. IEEE, 2010.

[70] Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels. A low-power coap for contiki. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 855–860. IEEE, 2011.

[71] M. Krochmal and S. Cheshire. Dns-based service discovery. Technical report, IETF, 2011.

[72] Philip Alexander Levis, Neil Patel, David Culler, and Scott Shenker. *Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks*. Computer Science Division, University of California, 2003.

[73] Y. Liang, X. Zhou, Z. Yu, H. Wang, and B. Guo. A context-aware resource management framework for smart homes. In *Ubiquitous Information Technologies and Applications (CUTE), 2010 Proceedings of the 5th International Conference on*, pages 1–8. IEEE, 2010.

[74] H. Luo and M. Barbeau. Performance evaluation of service discovery strategies in ad hoc networks. In *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, pages 61–68. IEEE, 2004.

[75] W. Ma, B. Wu, W. Zhang, and L. Cheng. Implementation of a lightweight service advertisement and discovery protocol for mobile ad hoc networks. In *IEEE Globecom*, pages 1–5. Citeseer, 2003.

[76] X. Ma and W. Luo. The analysis of 6lowpan technology. In *Computational Intelligence and Industrial Application, 2008. PACIIA'08. Pacific-Asia Workshop on*, volume 1, pages 963–966. IEEE, 2008.

[77] S. Madden, W. Hong, J. Hellerstein, and M. Franklin. TinydbâĂŤa declarative database for sensor networks. *TinyDB web page. http://telegraph. cs. berkeley. edu/tinydb*, 2005.

[78] R. Marin-Perianu, C. Lombriser, P. Havinga, H. Scholten, and G. Tröster. Tandem: A context-aware method for spontaneous clustering of dynamic wireless sensor nodes. In *Proceedings of the 1st international conference on The internet of things*, pages 341–359. Springer-Verlag, 2008.

[79] R. S. Marin-Perianu, P. H. Hartel, and J. Scholten. A classification of service discovery protocols. Technical Report TR-CTIT-05-25, Centre for Telematics and Information Technology University of Twente, Enschede, June 2005.

[80] R. S. Marin-Perianu, J. Scholten, P. J. M. Havinga, and P. H. Hartel. Performance evaluation of a cluster-based service discovery protocol for heterogeneous wireless sensor networks. Technical Report TR-CTIT-06-61, Centre for Telematics and Information Technology University of Twente, Enschede, October 2006.

[81] Raluca Marin-Perianu, Hans Scholten, Paul Havinga, and Pieter Hartel. Energy-efficient cluster-based service discovery in wireless sensor networks. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 931–938. IEEE, 2006.

[82] RS Marin-Perianu, J. Scholten, PJM Havinga, and PH Hartel. Cluster-based service discovery for heterogeneous wireless sensor networks. *International Journal of Parallel, Emergent and Distributed Systems*, 23(4):325–346, 2008.

[83] B. Martin and B. Jano. Wap binary xml content format. *W3C Note*, 64, 1999.

[84] Simon Mayer and Dominique Guinard. An extensible discovery service for smart things. In *Proceedings of the Second International Workshop on Web of Things*, page 7. ACM, 2011.

[85] Simon Mayer, Dominique Guinard, and Vlad Trifa. Searching in a web-based infrastructure for smart things. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 119–126. IEEE, 2012.

[86] M. Mealling and R. Daniel. Rfc 2915: The naming authority pointer (naptr) dns resource record, 2000.

[87] E. Meshkova, J. Riihijarvi, M. Petrova, and P. Mahonen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks*, 52(11):2097–2128, 2008.

[88] Adnan Noor Mian, Roberto Baldoni, and Roberto Beraldi. A survey of service discovery protocols in multihop mobile ad hoc networks. *Pervasive Computing, IEEE*, 8(1):66–74, 2009.

[89] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Ipv6 over low-power wireless personal area networks (6lowpans): Overview, assumptions, problem statement, and goals. *Network working group, IETF*, 2007.

[90] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of ipv6 packets over ieee 802.15. 4 networks. *Internet proposed standard RFC*, 4944, 2007.

[91] G. Moritz, E. Zeeb, F. Golatowski, D. Timmermann, and R. Stoll. Web services to improve interoperability of home healthcare devices. In *Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009. 3rd International Conference on*, pages 1–4. IEEE, 2009.

[92] G. Moritz, E. Zeeb, S. Pruter, F. Golatowski, D. Timmermann, and R. Stoll. Devices profile for web services in wireless sensor networks: Adaptations and enhancements. In *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1–8. IEEE, 2009.

[93] G. Mulligan. The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 78–82. ACM, 2007.

[94] S.A. Munir, X. Dongliang, C. Canfeng, and J. Ma. Virtual overlay for service classification & discovery in wireless sensor networks. In *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, volume 2, pages 991–996. IEEE, 2009.

[95] M. Nidd. Service discovery in deapspace. *Personal Communications, IEEE*, 8(4):39–45, 2001.

[96] D. Noh and H. Shin. Spiz: an effective service discovery protocol for mobile ad hoc networks. *EURASIP Journal on Wireless Communications and Networking*, 2007(1):27–27, 2007.

[97] George Oikonomou, Iain Phillips, and Theo Tryfonas. Ipv6 multicast forwarding in rpl-based wireless sensor networks. *Wireless Personal Communications*, pages 1–28, 2013.

[98] Angela Orebaugh, Gilbert Ramirez, Josh Burke, and Larry Pesce. *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Syngress Publishing, 2006.

[99] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641–648. IEEE, 2006.

[100] Charith Perera, Arkady B. Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *CoRR*, abs/1305.0982, 2013.

[101] S Pohlsen, Carsten Buschmann, and Christian Werner. Integrating a decentralized web service discovery system into the internet infrastructure. In *on Web Services, 2008. ECOWS'08. IEEE Sixth European Conference*, pages 13–20. IEEE, 2008.

[102] M. Presser, P.M. Barnaghi, M. Eurich, and C. Villalonga. The sensei project: integrating the physical world with the digital world of the network of the future. *Communications Magazine, IEEE*, 47(4):1–4, 2009.

[103] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *ACM SIGCOMM Computer Communication Review*, 31(4):161–172, 2001.

[104] M.T. Raza, S.W. Yoo, K.H. Kim, S.S. Joo, and W.C. Jeong. Design and implementation of an architectural framework for web portals in a ubiquitous pervasive environment. *Sensors*, 9(7):5201–5223, 2009.

[105] G.G. Richard III. Service advertisement and discovery: enabling universal device cooperation. *Internet Computing, IEEE*, 4(5):18–26, 2000.

[106] F. Sailhan and V. Issarny. Scalable service discovery for manet. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 235–244. IEEE, 2005.

[107] D. Sathan, A. Meetoo, and RK Subramaniam. Context aware lightweight energy efficient framework. *World Academy of Science, Engineering and Technology*, 52, 2009.

[108] G. Schiele, C. Becker, and K. Rothermel. Energy-efficient cluster-based service discovery for ubiquitous computing. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 14. ACM, 2004.

[109] J. Schneider and T. Kamiya. Efficient xml interchange (exi) format 1.0. *W3C Working Draft*, 19, 2008.

[110] X. Shao, L.H. Ngoh, T.K. Lee, T.Y. Chai, L. Zhou, and J.C.M. Teo. Multipath cross-layer service discovery (mcsd) for mobile ad hoc networks. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 408–413. IEEE, 2009.

[111] Z Shelby. Core link format. *draft-ietf-core-link-format-07 (work in progress)*, 2011.

[112] Z. Shelby and C. Bormann. *6LoWPAN: the wireless embedded internet*, volume 33. Wiley, 2010.

[113] Z Shelby and S Krco. Core resource directory, draft-shelby-core-resource-directory-02. *IETF work in progress*, 2012.

[114] Zach Shelby. Embedded web services. *Wireless Communications, IEEE*, 17(6):52–57, 2010.

[115] Tmote Sky. Tmote sky: Ultra low power ieee 802.15.4 compliant wireless sensor module. Datasheet, Moteiv Corporation, 2006.

[116] A. Sleman and R. Moeller. Integration of wireless sensor network services into other home and industrial networks; using device profile for web services (dpws). In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–5. Ieee, 2008.

[117] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, and V. Trifa. Soa-based integration of the internet of things in enterprise services. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 968–975. IEEE, 2009.

[118] D. Steinberg and S. Cheshire. *Zero configuration networking: the definitive guide*. O'Reilly Media, Inc., 2005.

[119] V. Stirbu. Towards a restful plug and play experience in the web of things. In *Semantic computing, 2008 IEEE international conference on*, pages 512–517. IEEE, 2008.

[120] Harsh Sundani, Haoyue Li, Vijay K Devabhaktuni, Mansoor Alam, and Prabir Bhattacharya. Wireless sensor network simulators a survey and comparisons. *International Journal of Computer Networks (IJCN)*, 2(6):249–265, 2011.

[121] Vasughi Sundramoorthy. *At home in service discovery*. University of Twente, 2006.

[122] Pascal Thubert. Objective function zero for the routing protocol for low-power and lossy networks (rpl). Technical Report RFC 6552, IETF, 2012.

[123] A. Varshavsky, B. Reid, and E. de Lara. The need for cross-layer service discovery in manets. *Report CSRG-492, UofT Computer Science*, 2004.

[124] A. Varshavsky, B. Reid, and E. de Lara. A cross-layer approach to service discovery and selection in manets. In *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[125] J Vasseur, M Kim, K Pister, N Dejean, and D Barthel. Routing metrics used for path calculation in low power and lossy networks. Technical report, IETF, 2011.

[126] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, M Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends*, pages 9–52, 2011.

[127] C.N. Ververidis and G.C. Polyzos. Extended zrp: a routing layer based service discovery protocol for mobile ad hoc networks. In *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*, pages 65–72. IEEE, 2005.

[128] C.N. Ververidis and G.C. Polyzos. Avert: Adaptive service and route discovery protocol for manets. In *Networking and Communications, 2008. WIMOB'08. IEEE International Conference on Wireless and Mobile Computing,*, pages 38–43. IEEE, 2008.

[129] C.N. Ververidis and G.C. Polyzos. Service discovery for mobile ad hoc networks: A survey of issues and techniques. *Communications Surveys & Tutorials, IEEE*, 10(3):30–45, 2008.

[130] Qiang Wei and Zhi Jin. Service discovery for internet of things: a context-awareness perspective. In *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*, page 25. ACM, 2012.

[131] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.

[132] Erik Wilde. Putting things to REST. Technical Report 2007-015, School of Information, UC Berkeley, Berkeley, California, November 2007.

[133] T Winter, ABP Thubert, and et al. Clausen. Rpl: Ipv6 routing protocol for low-power and lossy networks. Technical Report RFC 6550, IETF, 2012.

[134] M.A. Wister and D.A. Torres. Lift: An efficient cross-layer service discovery protocol in manet. In *Advanced Information Networking and Applications*

Workshops, 2009. WAINA'09. International Conference on, pages 781–786. IEEE, 2009.

[135] A. Wood, J. Stankovic, G. Virone, L. Selavo, Z. He, Q. Cao, T. Doan, Y. Wu, L. Fang, and R. Stoleru. Context-aware wireless sensor networks for assisted living and residential monitoring. *Network, IEEE*, 22(4):26–33, 2008.

[136] D. Zeng, S. Guo, and Z. Cheng. The web of things: A survey. *Journal of Communications*, 6(6):424–438, 2011.

[137] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.

[138] F. Zhu, M. Mutka, and L. Ni. Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 235–242. IEEE, 2003.

[139] Fen Zhu, Matt W Mutka, and Lionel M Ni. Service discovery in pervasive computing environments. *Pervasive Computing, IEEE*, 4(4):81–90, 2005.

Appendices

Appendix A

Automation of experiments

This chapter covers the detail about the automation of experiments. Each experiments consists of four different processes: COOJA, router to connect the COOJA with other processes, DA and UA. Each process is started in an order depending on the parameters of a simulation run, as shown in Figure 6.5. Therefore, a bash script is written to automate the process for different runs and their multiple iterations. This chapter explains the automation process and presents examples from the actual codes used in simulations.

A.1 Scenarios automation

This research project has defined several bash scripts to execute simulations in a bulk. There are multiple scripts written to automate the process. A Code Listing A.1 gives an example script that is actually used in experiments. Furthermore, Figure A.1 shows the command given to this script to execute a set of simulation scenarios. The script uses the passed attributes to generate and update file appropriately, before actually starting all the process. The script makes changes in code files of various entities (SA, GM or GL) and generates new COOJA simulation files with the specified topology and settings. At the end of each simulation run, it saves the log files generated by all processes in a directory for debugging, validation and further processing.

Listing A.1: Simulations Automation script

```
1  #!/bin/bash

3  #Swicthing to correct GIT branches
   experimentDIR="/home/talal"
5  cd $experimentDIR/uatrendy/
   git checkout trendy-1
7  cd $experimentDIR/datrendy/
   git checkout trendy-1
9  cd $experimentDIR/sensinode-contiki/
   git add .
11 git reset --hard HEAD
   git checkout trendy-1-fullp
```

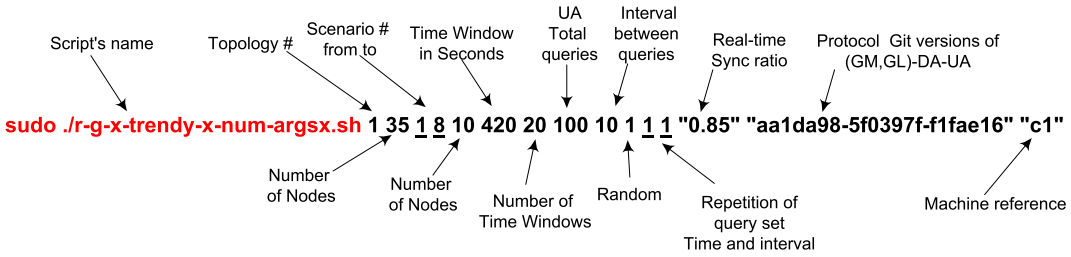


Figure A.1: Simulation automation: an example command to run bash script

```
13

15  #Checking for the right number of arguments
    noOfArgumentsReqd=12;
17  if [ "$#" -ne "$noOfArgumentsReqd" ]; then
      echo "Usage: Need $noOfArgumentsReqd arguments"
19    echo "Format: NUMBEROFNODES simModeFrom simModeTo ... DATotalTimeWindows simIterations
          simRef VMNo"
      exit 1
21  fi
     echo "topology= $1, simModeFrom=$2, simModeTo=$3, simIterations=$4, DATimeWindow=$5,
         DATotalTimeWindows=$6, UaStart = $7, queryInOneSet=$8, ratio=$9, nullRDC=${10}, simRef=$
         {11}, VMNo=${12}"
23
    topology=$1
25  simModeFrom=$2
    simModeTo=$3
27  simIterations=$4
    DATimeWindow=$5
29  DATimeWindow=$(( ${DATimeWindow#0} ))
    totalTimeWindows=$6
31  totalTimeWindows=$(( ${totalTimeWindows#0} ))
    DATotalTimeWindows=$6
33  UAStartTime=$7
    queryInOneSet=$8
35  queryInterval=10
    queryRandomInterval=1
37  querySetRepeatTimes=1
    querySetInterval=1
39  ratio=${9}
    nullRDC=${10}
41  simRef=${11}
    VMNo=${12}
43  NUMBEROFNODES=35

45  #Stopping Dropbox to free system's resources
    sudo service dropbox stop
47
    DA="$experimentDIR/datrendy/dist/datrendy.jar"
49  UA="$experimentDIR/uatrendy/dist/uatrendy.jar"
    totalSimulationDuration=$DATimeWindow*$DATotalTimeWindows
51
    #Logging Directories for all
53  BASE="/home/talal/Dropbox/experiments/results"
    gBASE="/home/talal/Dropbox/experiments/gresults"
55
    #Creating specific Directories
57  DATE=$(date +%F)
    TIME=$(date +%H%M)
59  FDIR="$BASE/$DATE"
    mkdir -p $FDIR
61  FDIR="$FDIR/$TIME-ALL-$simRef"
    mkdir -p $FDIR
63  echo $FDIR" is created"

65  gFDIR="$gBASE/$DATE"
    mkdir -p $gFDIR
67  gFDIR="$gFDIR/$TIME-$simRef"
    mkdir -p $gFDIR
69  echo $gFDIR" is created"
```

```
71
GDIR="$gFDIR/ALL-$NUMBEROFNODES-top$topology-tw$totalTimeWindows-q$queryInOneSet-
    r$queryRandomInterval-n$querySetRepeatTimes-for-$simIterations-[$VMNo]"
73 mkdir -p $GDIR
echo $GDIR" is created for Simulation record"
75
#Selecting a simulation variation
77 for ((i = $simModeFrom; i <= $simModeTo; i++)); do

79 SIMNAMEFORMATTED="case-$i-top$topology-tw$totalTimeWindows-q$queryInOneSet-
    r$queryRandomInterval-n$querySetRepeatTimes"
CONTIKI="$experimentDIR/sensinode-contiki"
81 timerMin=1;
timerMax=1;
83 timerThreshold=1;
timerStep=0;
85 #Setting configurations for each simulation
case "$i" in
87     "0")
           isgrouping=0;
89         numOfGLs=0;
           DAMode=0;
91         appubThreshold=0
           ;;
93     "1")
           isgrouping=0;
95         numOfGLs=0;
           DAMode=1;
97         appubThreshold=0
           ;;
99     "2")
           isgrouping=1;
101        numOfGLs=5;
           DAMode=4;
103        appubThreshold=0
           ;;
105    "3")
           isgrouping=0;
107        numOfGLs=0;
           DAMode=7;
109        appubThreshold=2
           ;;
111    "4")
           isgrouping=1;
113        numOfGLs=5;
           DAMode=8;
115        appubThreshold=2
           ;;
117    *)
           echo "Ending"
119        ;;
esac
121
#Random seeds for different number of iterations
123 for ((j = 1; j <= $simIterations; j++)); do

125 case "$j" in

127    "1")
           newrandomseed=1668841902061472829;
129        ;;
       "2")
131        newrandomseed=-8020676306221162569;
           ;;
133    "3")
           newrandomseed=-5174799744808039206;
135        ;;
       "4")
137        newrandomseed=5471226677158381259;
           ;;
139    "5")
           newrandomseed=2442086531776532400;
141        ;;
       "6")
143        newrandomseed=-52470407069163422;
           ;;
145    "7")
           newrandomseed=-290209011825205304;
147        ;;
```

```
    "8")
149         newrandomseed=-8489044479846245982;
            ;;
151     "9")
            newrandomseed=-2437497649994250447;
153         ;;
        "10")
155         newrandomseed=4869737434645930484;
            ;;
157     *)
            echo "Ending"
159         ;;
    esac
161
    #----------------COOJA SIMULATION GENERATION----------------------
163 #Total Nodes and/or grouping [Specific to Simulation]
    LOWPAN="$CONTIKI/work/trendy-gm"
165 COOJA="$CONTIKI/tools/cooja/dist/cooja.jar"

167 cd $LOWPAN
    headerFile="sim-header.cooja"
169 topologyFile="$NUMBEROFNODES-"g"-$numOfGLs"g"-$topology.topology"
    footerFile="sim-footer.cooja"
171
    SIMFILE="$NUMBEROFNODES-"g"-$numOfGLs"g"-$topology-grouping.csc"
173 cat $headerFile $topologyFile $footerFile > "$SIMFILE"
    SIMULATION="$SIMFILE"
175
    #---------Selecting COOJA SCRIPT-----------
177 scriptFile="$NUMBEROFNODES-$DATimeWindow-t$totalTimeWindows.js"
    cat "$LOWPAN/general-sim.js" > "$scriptFile"
179 echo "$scriptFile file is created"

181 #---------Changes in COOJA SCRIPT for configurations--------
    timeout=$(($DATimeWindow*$totalTimeWindows*1000))
183 timeoutPlus=$(($DATimeWindow*$totalTimeWindows*1000+100000))

185 linetochange="MAIN-TIMEOUT"
    newlinewithcoojascript="$timeoutPlus"
187 awk -v var1="$linetochange" -v var2="$newlinewithcoojascript" '{
      gsub(var1,var2)
189   print
    }' $LOWPAN/$scriptFile > temp
191 mv temp $LOWPAN/$scriptFile

193 linetochange="TIME-OUT-FINALIZE"
    newlinewithcoojascript="$timeout"
195 awk -v var1="$linetochange" -v var2="$newlinewithcoojascript" '{
      gsub(var1,var2)
197   print
    }' $LOWPAN/$scriptFile > temp
199 mv temp $LOWPAN/$scriptFile

201 linetochange="nrNodes = 0;"
    numOfNodesForJS=$(($NUMBEROFNODES+1))
203 newlinewithcoojascript="nrNodes = $numOfNodesForJS;"
    awk -v var1="$linetochange" -v var2="$newlinewithcoojascript" '{
205   gsub(var1,var2)
      print
207 }' $LOWPAN/$scriptFile > temp
    mv temp $LOWPAN/$scriptFile
209
    linetochange="total_reports_needed = 0;"
211 numOfNodesForJS=$(($NUMBEROFNODES+1))
    newlinewithcoojascript="total_reports_needed = $NUMBEROFNODES;"
213 awk -v var1="$linetochange" -v var2="$newlinewithcoojascript" '{
      gsub(var1,var2)
215   print
    }' $LOWPAN/$scriptFile > temp
217 mv temp $LOWPAN/$scriptFile

219 linetochange="RATIO-TO-CHANGE1"
    newlinewithcoojascript="$ratio"
221 awk -v var1="$linetochange" -v var2="$newlinewithcoojascript" '{
      gsub(var1,var2)
223   print
    }' $LOWPAN/$scriptFile > temp
225 mv temp $LOWPAN/$scriptFile

227 linetochange="RATIO-TO-CHANGE2"
```

```
    newlinewithcoojascript="$ratio"
229 awk -v var1="$linetochange" -v var2="$newlinewithcoojascript" '{
      gsub(var1,var2)
231   print
    }' $LOWPAN/$scriptFile > temp
233 mv temp $LOWPAN/$scriptFile

235 echo "$scriptFile file is CHANGED"

237 #----------------Java script for COOJA SIMULATION-----------------
    linetochange="<scriptfile></scriptfile>"
239 newlinewithcoojascript="<scriptfile>[CONTIKI_DIR]/work/trendy-gm/$scriptFile</scriptfile>"

241 awk -v var1="$linetochange" -v var2="$newlinewithcoojascript" '{
      gsub(var1,var2)
243   print
    }' $SIMULATION > temp
245 mv temp $SIMULATION

247 #-------------Changing Simulation speed-----------------
    linetochange="<speedlimit>0.85</speedlimit>"
249 newlinewithcoojascript="<speedlimit>$ratio</speedlimit>"
    awk -v var1="$linetochange" -v var2="$newlinewithcoojascript" '{
251   gsub(var1,var2)
      print
253 }' $SIMULATION > temp
    mv temp $SIMULATION
255
    #---------------------Changes in other files-------------------------
257 workDIR="$CONTIKI/work"
    SAVEIFS=$IFS
259 IFS=$(echo -en "\n\b")
    for ((gm = 1; gm <= 5; gm++)); do
261
    case "$gm" in
263
        "1")
265         gmFile="$workDIR/trendy-gm/group-member.c"
            ;;
267     "2")
            gmFile="$workDIR/trendy-gm group-2/group-member.c"
269         ;;
        "3")
271         gmFile="$workDIR/trendy-gm group-3/group-member.c"
            ;;
273     "4")
            gmFile="$workDIR/trendy-gm group-4/group-member.c"
275         ;;
        "5")
277         gmFile="$workDIR/trendy-gm group-5/group-member.c"
            ;;
279     *)
            echo "Ending"
281         ;;
    esac
283 echo "selected file: $gmFile and APPUB = $appubThreshold"

285 linetochange="#define hit_count_threshold"
    newlinewithcoojascript="#define hit_count_threshold $appubThreshold"
287 sed -i "s/$linetochange.*/$newlinewithcoojascript/" $gmFile
    done
289
    IFS=$SAVEIFS
291
    #---------------Selection of RDC-------------------
293 workDIR="$CONTIKI/work"
    SAVEIFS=$IFS
295 IFS=$(echo -en "\n\b")
    for ((sa = 1; sa <= 11; sa++)); do
297
    case "$sa" in
299
        "1")
301         saFile="$workDIR/trendy-gm/project-conf.h"
            ;;
303     "2")
            saFile="$workDIR/trendy-gm group-2/project-conf.h"
305         ;;
        "3")
307         saFile="$workDIR/trendy-gm group-3/project-conf.h"
```

```
            ;;
309     "4")
            saFile="$workDIR/trendy-gm group-4/project-conf.h"
311         ;;
        "5")
313         saFile="$workDIR/trendy-gm group-5/project-conf.h"
            ;;
315     "6")
            saFile="$workDIR/trendy-gl/project-conf.h"
317         ;;
        "7")
319         saFile="$workDIR/trendy-gl group-2/project-conf.h"
            ;;
321     "8")
            saFile="$workDIR/trendy-gl group-3/project-conf.h"
323         ;;
        "9")
325         saFile="$workDIR/trendy-gl group-4/project-conf.h"
            ;;
327     "10")
            saFile="$workDIR/trendy-gl group-5/project-conf.h"
329         ;;
        "11")
331         saFile="$CONTIKI/examples/ipv6/rpl-border-router/project-conf.h"
            ;;
333     *)
            echo "Ending"
335         ;;
    esac
337 echo "selected file: $saFile and nullRDC = $nullRDC"

339 linetochange="#define NULLRDC"
    newlinewithcoojascript="#define NULLRDC  $nullRDC"
341 sed -i "s/$linetochange.*/$newlinewithcoojascript/" $saFile
    done
343
    IFS=$SAVEIFS
345
    #--------------Loading new seed values in a simulation-----------
347 linetochange="<randomseed>generated</randomseed>"
    newlinewithrandomseed="<randomseed>$newrandomseed</randomseed>"
349
    awk -v var1="$linetochange" -v var2="$newlinewithrandomseed" '{
351   gsub(var1,var2)
      print
353 }' $SIMULATION > temp
    mv temp $SIMULATION
355
    echo $SIMULATION" with random seed = $newrandomseed is starting"
357
    #Specific to Simulation
359 SDIR="$FDIR/$NUMBEROFNODES-$SIMNAMEFORMATTED-for-$simIterations-[$VMNo]"
    mkdir -p $SDIR
361 echo $SDIR" is created for Simulation record"

363 unset DISPLAY
    cd $LOWPAN
365


367 #Log file for Simulation's terminal output
    LOGFILE="$SDIR/sim$j.log"
369 echo "GIT ID: $simRef by VM#$VMNo" >>$LOGFILE
    echo "The Simulation Duration = $totalSimulationDuration" >>$LOGFILE
371 echo "Total Nodes $NUMBEROFNODES" >>$LOGFILE
    echo " of $j th of $simIterations iterations" >>$LOGFILE
373 echo "DA last for $DATotalTimeWindows time windows each with $DATimeWindow seconds" >>$LOGFILE
    echo "UA will query $queryInOneSet queries with interval of $queryInterval seconds" >>$LOGFILE
375 echo "UA will repeat above query set for $querySetRepeatTimes times each with
        $querySetInterval seconds interval" >>$LOGFILE

377 #Cleaning up before ending
    rm *.testlog
379 rm *.txt
    rm *.log
381 rm *.dat
    rm *.pcap
383
    #----------Starting and scheduling processes-----------------------
385 sleep 120 && sudo make connect-router-cooja >> $LOGFILE&
    pid_cooja_router=$!
```

```
sleep $UAStartTime && java -jar $UA $queryInOneSet $queryInterval $queryRandomInterval
    $querySetRepeatTimes $querySetInterval $ratio >> $LOGFILE&
pid_ua=$!

java -jar $DA $DATimeWindow $DATotalTimeWindows $DAMode $ratio $timerMin $timerMax
    $timerThreshold $timerStep >> $LOGFILE&
pid_da=$!
/usr/bin/time --verbose -ocoojatime.log -a java -jar $COOJA -nogui=$SIMULATION -contiki=
    $CONTIKI >> $LOGFILE&
pid_cooja=$!

wait $pid_cooja

sleep 10
kill -9 $pid_ua
kill -9 $pid_da
kill -9 $pid_cooja_router

#General detailed log
cat daDetail.txt >> "$SDIR/$SIMNAMEFORMATTED-da-detail.log"
cat daxtradetail.txt >> "$SDIR/$SIMNAMEFORMATTED-daFullDetail.log"
cat uacompleteLog.txt >> "$SDIR/$SIMNAMEFORMATTED-ua-detail.log"
cat COOJA.testlog >> "$SDIR/$SIMNAMEFORMATTED-Lowpan-detail.log"
cat daperformance.txt >> "$SDIR/$SIMNAMEFORMATTED-daperformance.log"
cat uaperformance.txt >> "$SDIR/$SIMNAMEFORMATTED-uaperformance.log"
cat uaperformance-processed.txt >> "$SDIR/$SIMNAMEFORMATTED-uaperformance-processed.log"
cat l-energy-all.dat >> "$SDIR/$SIMNAMEFORMATTED-l-energy-all.dat"
cat l-energy-ind.dat >> "$SDIR/$SIMNAMEFORMATTED-l-energy-ind.dat"
cat l-raw-energy-ind.dat >> "$SDIR/$SIMNAMEFORMATTED-l-raw-energy-ind.dat"
cat l-packet-all.dat >> "$SDIR/$SIMNAMEFORMATTED-l-packet-all.dat"
cat l-packet-ind.dat >> "$SDIR/$SIMNAMEFORMATTED-l-packet-ind.dat"
cat radio-packets.dat >> "$SDIR/$SIMNAMEFORMATTED-radio-packets.log"
cat "radiolog-"*".pcap" >> "$SDIR/$SIMNAMEFORMATTED-packets.pcap"
cat coojatime.log >> "$SDIR/$SIMNAMEFORMATTED-coojaTime.log"


#Saving Log for generating graphs
cat uaperformance.txt >> "$GDIR/case-$i-uaperformance.log"
cat uaperformance-processed.txt >> "$GDIR/case-$i-uaperformance-processed.log"
cat daperformance.txt >> "$GDIR/case-$i-daperformance.log"
cat l-energy-all.dat >> "$GDIR/case-$i-energy-all.dat"
cat l-energy-ind.dat >> "$GDIR/case-$i-energy-ind.dat"
cat l-raw-energy-ind.dat >> "$GDIR/case-$i-raw-energy-ind.dat"
cat l-packet-all.dat >> "$GDIR/case-$i-packet-all.dat"
cat l-packet-ind.dat >> "$GDIR/case-$i-packet-ind.dat"
cat radio-packets.dat >> "$GDIR/case-$i-radio-packets.log"
cat "radiolog-"*".pcap" >> "$GDIR/case-$i-packets.pcap"


#Cleaning up before ending
rm *.testlog
rm *.txt
rm *.log
rm *.dat
rm *.pcap

done #End of multiple iterations of one Simulation
done #End of one Unique Simulation - Loop for all simulations
sudo service dropbox start
```

A.2 Script for 6LoWPAN data gathering

Each COOJA simulation requires some Javascript to produce results in log files with different level of details. This automates the process of result gathering in a specified format for each experiment, which aids the process of debugging, validation and measurements of different performance metrics for 6LoWPANs. Following is an example of such a Javascript:

```
importPackage(java.io);

// Function to record statistics after each time window
```

```
function
print_stats()
{
total_energy_consumption = total_cpu_energy_consumption + total_lpm_energy_consumption
            + total_listen_energy_consumption + total_transmit_energy_consumption;

    log.log("----------------Time = "+ time +"------------------------\n");
    log.log("Total Nodes = " + (nrNodes-1) +":\n");
    log.log("Total reports = " + total_reports +":\n");
    log.log("Total Update Messages =" + total_updates_msgs + "\n");
    log.log("Total Grouping Messages =" + total_grouping_msgs + "\n");

    log.log("Total CPU Consumption =" + Math.round (total_cpu_energy_consumption)
                    + " [" + Math.round ((total_cpu_energy_consumption/
                        total_energy_consumption)*100, 5) + "%]");

    log.log("Total LPM Consumption =" + Math.round (total_lpm_energy_consumption)
                    + " [" + Math.round ((total_lpm_energy_consumption/
                        total_energy_consumption)*100, 5) + "%]");

    log.log("Total LISTEN Consumption =" + Math.round (total_listen_energy_consumption)
                    + " [" + Math.round ((total_listen_energy_consumption/
                        total_energy_consumption)*100, 5) + "%]");

    log.log("Total TRANSMIT Consumption =" + Math.round (total_transmit_energy_consumption)
                        + " [" + Math.round ((total_transmit_energy_consumption/
                            total_energy_consumption)*100, 5) + "%]");

    log.log("Total Energy Consumption =" + (total_energy_consumption)/1000  + " Joules [milli:
        "+total_energy_consumption);
    log.log("-------------------------------------------------------------\n");

gtime = time/1000000;
remainder = gtime%60;

if(remainder<200){
gtime = gtime-remainder;
}

// File to save aggregated energy statistics: lAllEnergyDescfile
writeinfile = gtime +"  "+total_energy_consumption +"   "+total_cpu_energy_consumption+"    "+
    total_lpm_energy_consumption +"   "+total_listen_energy_consumption+" "+
    total_transmit_energy_consumption+"\n";
output1[lAllEnergyDescfile].write(writeinfile);
log.log("lAllEnergyDescfile:"+ writeinfile);

// File to save aggregated packets statistics: lAllPacketDescfile
writeinfile = gtime +"  "+ (total_updates_msgs + total_grouping_msgs) + "   "+
    total_updates_msgs + "   "+ total_grouping_msgs +"\n";
output3[lAllPacketDescfile].write(writeinfile);
log.log("lAllPacketDescfile:"+ writeinfile);

}


TIMEOUT(MAIN_TIMEOUT);
/* override simulation speed limit to specified one*/
sim.setSpeedLimit(RATIO_TO_CHANGE1);

log.log("--------------------------------------------------------------------------------\n");
/* Configurations at start */
nrNodes = 0;
node_reported = new Array();
cpu_value = new Array();
lpm_value = new Array();
listen_value = new Array();
transmit_value = new Array();
energy_value = new Array();
updates_value = new Array();
grouping_value = new Array();
data = new Array();
total_updates_msgs = 0;
total_grouping_msgs = 0;
total_cpu_energy_consumption = 0;
total_lpm_energy_consumption = 0;
total_listen_energy_consumption = 0;
total_transmit_energy_consumption = 0;
total_energy_consumption = 0;
total_reports = 0;
total_reports_needed = 0;
```

```
all_reported = false;

output1 = new Object();
output2 = new Object();
output3 = new Object();
output4 = new Object();
output5 = new Object();
saNodes = nrNodes-1;
lAllEnergyDescfile =  "l-energy-all.dat";
lIndEnergyDescfile =  "l-energy-ind.dat";
lIndRawEnergyDescfile =  "l-raw-energy-ind.dat";
lAllPacketDescfile =  "l-packet-all.dat";
lIndPacketDescfile =  "l-packet-ind.dat";

nodes_starting = true;
for(i = 2; i <= nrNodes; i++) {
    node_reported[i] = false;
    energy_value[i] = 0;
    cpu_value[i] = 0;
    lpm_value[i] = 0;
    listen_value[i] = 0;
    transmit_value[i] = 0;
    updates_value[i] = 0;
    grouping_value[i] = 0;
}

// This loop will run till the end and wait for different messages to record them
while (true) {

// Creating different log files
if(!output1[lAllEnergyDescfile]){
output1[lAllEnergyDescfile] = new FileWriter(lAllEnergyDescfile);
slinetowrite =  "#Time: calculated_energy_consumption   calculated_cpu_energy_consumption
    calculated_lpm_energy_consumption   calculated_listen_energy_consumption
    calculated_transmit_energy_consumption"+ "\n";
//output1[lAllEnergyDescfile].write(slinetowrite);
log.log(slinetowrite);
}

if(!output2[lIndEnergyDescfile]){
output2[lIndEnergyDescfile] = new FileWriter(lIndEnergyDescfile);
slinetowrite =   "#Time:    i   energy_value[i] cpu_value[i]    lpm_value[i]    listen_value[i
    ] transmit_value[i]"+ "\n";
output2[lIndEnergyDescfile].write(slinetowrite);
log.log(slinetowrite);
}

if(!output3[lAllPacketDescfile]){
output3[lAllPacketDescfile] = new FileWriter(lAllPacketDescfile);
slinetowrite =   "#Time:    Total Trendy Packets:   UPD GROUPING"+ "\n";
output3[lAllPacketDescfile].write(slinetowrite);
log.log(slinetowrite);
}

if(!output4[lIndPacketDescfile]){
output4[lIndPacketDescfile] = new FileWriter(lIndPacketDescfile);
slinetowrite =   "#Time:    Node-ID:    Total Trendy Packets:   UPD GROUPING"+ "\n";
output4[lIndPacketDescfile].write(slinetowrite);
log.log(slinetowrite);
}

if(!output5[lIndRawEnergyDescfile]){
output5[lIndRawEnergyDescfile] = new FileWriter(lIndRawEnergyDescfile);
slinetowrite =   "#Time: i  raw_energy_value[i] raw_cpu_value[i] raw_lpm_value[i]
    raw_listen_value[i] raw_transmit_value[i]"+ "\n";
log.log(slinetowrite);
}


try{
    YIELD();
}catch(e){
        output1[lAllEnergyDescfile].close();
        output2[lIndEnergyDescfile].close();
        output3[lAllPacketDescfile].close();
        output4[lIndPacketDescfile].close();
        output5[lIndRawEnergyDescfile].close();
        log_func();
        throw('test script killed');
    }
```

```
/* Enforcing simulation speed limit to specified one */
sim.setSpeedLimit(RATIO_TO_CHANGE2);

// Recording error messages
if(msg.contains("off-link")){
nmsg=msg.substr(msg.indexOf("slip-bridge"));
log.log( "\n" + id +" at "+ time +  ":" + nmsg + "\n");
continue;
}
log.log( "\n" + id +" at "+ time +  ":" + msg + "\n");

// Recording Statistics message
if(!msg.contains("S:")){continue;}

    node_reported[id] = true;
    log.log( id +" at "+ time +  ":" + msg + "\n");
    data = msg.split(":");
    updates_value[id] = Number(data[1]);
    grouping_value[id] = Number(data[2]);
    cpu_value[id]= Number(data[3])
    lpm_value[id]= Number(data[4]);
    listen_value[id]= Number(data[5]);
    transmit_value[id]= Number(data[6]);
    energy_value[id]= cpu_value[id]+lpm_value[id]+listen_value[id]+transmit_value[id];


// Time is rounded
gtime = time/1000000;
remainder = gtime%60;

if(remainder<20){
gtime = gtime-remainder;
}

// Synchronisation with the DA after each time window
syncWithDA = new Object();
if(!syncWithDA["sync"]){
syncWithDA["sync"] = new FileWriter("sync.dat");
log.log( "Sync File Created");
}
syncWithDA["sync"].write(""+gtime+"");
syncWithDA["sync"].close();

// #1: lIndEnergyDescfile
log.log( id + " Before Conversion:[CPU,LPM,LSN,TRN]=[" + cpu_value[id] +" "+ lpm_value[id]+" "
     + listen_value[id]+" " + transmit_value[id]+ "]\n");

writeinfile = gtime +"  "+id +" "+energy_value[id]+"    "+cpu_value[id]+"   "+lpm_value[id]+"
       "+listen_value[id]+"    "+transmit_value[id]+"\n";
    output5[lIndRawEnergyDescfile].write(writeinfile);
    log.log("lIndRawEnergyDescfile:"+ writeinfile);

            cpu_value[id] = ((cpu_value[id]*(0.5*3.9))*3)/32768;
            lpm_value[id] = ((lpm_value[id]*(0.0545))*3)/32768;
            listen_value[id] = ((listen_value[id]*19.7)*3)/32768;
            transmit_value[id] = ((transmit_value[id]*17.4)*3)/32768;
        energy_value[id] =  cpu_value[id] + lpm_value[id] + listen_value[id] + transmit_value[
            id];
log.log( id +" After Conversion :[CPU,LPM,LSN,TRN=TOTAL]=[" + cpu_value[id] +" "+ lpm_value[id
    ]+" " + listen_value[id]+" " + transmit_value[id]+" ="+ energy_value[id]+"]\n");

    writeinfile = gtime +"  "+id +" "+energy_value[id]+"    "+cpu_value[id]+"   "+lpm_value[id
        ]+"   "+listen_value[id]+"    "+transmit_value[id]+"\n";
    output2[lIndEnergyDescfile].write(writeinfile);
    log.log("lIndEnergyDescfile:"+ writeinfile);

// #2: lIndPacketDescfile
    writeinfile = gtime +"  "+id +" " +(updates_value[id]+grouping_value[id]) +"    "+
        updates_value[id]+"   "+grouping_value[id]+"\n";
    output4[lIndPacketDescfile].write(writeinfile);
    log.log("lIndPacketDescfile:"+ writeinfile);

    total_reports++;

// Recording details till and reset after each time window
if(total_reports >= total_reports_needed) {
            all_reported = true;
  }
```

```
    if(all_reported){
227         all_reported = false;

229         for(i = 2; i <= nrNodes; i++) {
                total_updates_msgs += updates_value[i];
231             total_grouping_msgs += grouping_value[i];
                total_cpu_energy_consumption += cpu_value[i];
233             total_lpm_energy_consumption += lpm_value[i];
                total_listen_energy_consumption += listen_value[i];
235             total_transmit_energy_consumption += transmit_value[i];
            }
237
            print_stats();
239         total_updates_msgs = 0;
            total_grouping_msgs = 0;
241         total_energy_consumption = 0;

243 if(sim.getSimulationTimeMillis()>TIME_OUT-FINALIZE){
            log.log( "Ended at "+ time +  "\n");
245         output1[lAllEnergyDescfile].close();
            output2[lIndEnergyDescfile].close();
247         output3[lAllPacketDescfile].close();
            output4[lIndPacketDescfile].close();
249 log_func();
    SCRIPT_TIMEOUT();
251 }

253         for(i = 2; i <= nrNodes; i++) {
                node_reported[i] = false;
255             updates_value[i] = 0;
                grouping_value[i] = 0;
257             cpu_value[i]= 0;
                lpm_value[i]= 0;
259             listen_value[i]= 0;
                transmit_value[i]= 0;
261             energy_value[i]= 0;
                total_reports = 0;
263             total_updates_msgs = 0;
                total_grouping_msgs = 0;
265             total_energy_consumption = 0;
                total_cpu_energy_consumption = 0;
267             total_lpm_energy_consumption = 0;
                total_listen_energy_consumption = 0;
269             total_transmit_energy_consumption = 0;
            }
271
    }
273
}
```

Appendix B

Logs for validation and debugging

This chapter describes the validation log generated by simulations and gives some examples with the snippets of some log files. Each simulation generates number of log files produced by the automation script process, COOJA, UA and the DA. All the log files are placed in a hierarchy of date and time in different folders, which makes the process of debugging and validation easier and manageable. During the testing, verification and validation of the results, these logs contains all the information required to confirm the origin of a bug. For example, the DA generates three different log files, which provides general perspective to a detailed one. Similarly, the 6LoWPAN log is generated using COOJA script, which also provides different level of detail in each file. These log files are used extensively during the implementation, testing and debugging phases.

B.1 Simulation Log

A separate log file is generated for each simulation iteration, which represents the more elaborative perspective of a simulation. Following is an example of such a log file:

Listing B.1: Log of automation script (covers all processes)

```
2  GIT ID: aa1da98-5f0397f-f1fae16 by VM#c1
   The Simulation Duration = 420*20
4  Total Nodes 35
    of 1 th of 2 iterations
6  DA last for 20 time windows each with 420 seconds
   UA will query 100 queries with interval of 10 seconds
8  UA will repeat above query set for 1 times each with 1 seconds interval

10 New Report time window: Now after 494117:

12 New Performance Analysis print interval = 70588:
   Mode=4[Basic SD: with timer and grouping]:TimeWindow{420} Grouping{true} Trendy{1,9} Adaptive{
        Enabled[true], threshold[1], step[2], Limit[1], RetainThreshold[1]} PPUB{ Enabled[false],
        Threshold[0], Limit[0], RetainThreshold[0]}Directory Agent listening on port 5683.
14 DA will run for more than 9882 seconds.
    WARN [main] (GUI.java:3152) - JAVA_HOME environment variable not set, Cooja motes may not
        compile
```

```
16  INFO [main] (GUI.java:1325) -> Starting Cooja
    INFO [main] (GUI.java:2839) - External tools default settings: /external_tools_linux.config
18  INFO [main] (GUI.java:2869) - External tools user settings: /root/.cooja.user.properties
    INFO [main] (Simulation.java:423) - Simulation random seed: -1948817365486295399
20  INFO [main] (CompileContiki.java:140) -> make border-router.sky TARGET=sky
    INFO [main] (CompileContiki.java:140) -> make group-member.sky TARGET=sky
22  INFO [main] (CompileContiki.java:140) -> make group-member.sky TARGET=sky
    INFO [main] (CompileContiki.java:140) -> make group-member.sky TARGET=sky
24  INFO [main] (CompileContiki.java:140) -> make group-member.sky TARGET=sky
    INFO [main] (CompileContiki.java:140) -> make group-member.sky TARGET=sky
26  INFO [main] (CompileContiki.java:140) -> make group-leader.sky TARGET=sky
    INFO [main] (CompileContiki.java:140) -> make group-leader.sky TARGET=sky
28  INFO [main] (CompileContiki.java:140) -> make group-leader.sky TARGET=sky
    INFO [main] (CompileContiki.java:140) -> make group-leader.sky TARGET=sky
30  INFO [main] (CompileContiki.java:140) -> make group-leader.sky TARGET=sky
*** Setting up f1611 IO!
32  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/
        examples/ipv6/rpl-border-router/border-router.sky
*** Setting up f1611 IO!
34  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gl/group-leader.sky
*** Setting up f1611 IO!
36  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gl group-2/group-leader.sky
*** Setting up f1611 IO!
38  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gl group-3/group-leader.sky
*** Setting up f1611 IO!
40  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gl group-4/group-leader.sky
*** Setting up f1611 IO!
42  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gl group-5/group-leader.sky
*** Setting up f1611 IO!
44  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm/group-member.sky
*** Setting up f1611 IO!
46  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm/group-member.sky
*** Setting up f1611 IO!
48  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm/group-member.sky
*** Setting up f1611 IO!
50  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm/group-member.sky
*** Setting up f1611 IO!
52  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm/group-member.sky
*** Setting up f1611 IO!
54  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm/group-member.sky
*** Setting up f1611 IO!
56  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-2/group-member.sky
*** Setting up f1611 IO!
58  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-2/group-member.sky
*** Setting up f1611 IO!
60  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-2/group-member.sky
*** Setting up f1611 IO!
62  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-2/group-member.sky
*** Setting up f1611 IO!
64  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-2/group-member.sky
*** Setting up f1611 IO!
66  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-2/group-member.sky
*** Setting up f1611 IO!
68  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-3/group-member.sky
*** Setting up f1611 IO!
70  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-3/group-member.sky
*** Setting up f1611 IO!
72  INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
        trendy-gm group-3/group-member.sky
*** Setting up f1611 IO!
```

```
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-3/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-3/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-3/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-4/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-4/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-4/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-4/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-4/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-4/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-5/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-5/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-5/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-5/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-5/group-member.sky
*** Setting up f1611 IO!
 INFO [main] (MspMote.java:216) - Loading firmware from: /home/talal/sensinode-contiki/work/
      trendy-gm group-5/group-member.sky
 INFO [main] (GUI.java:1768) - Visualized plugin was not started: class se.sics.cooja.plugins.
      SimControl
 INFO [main] (GUI.java:1768) - Visualized plugin was not started: class se.sics.cooja.plugins.
      LogListener
 INFO [main] (SerialSocketServer.java:119) - Listening on port: 60001
 INFO [main] (ScriptRunner.java:421) - Test script deactivated
 INFO [main] (LogScriptEngine.java:263) - Script timeout in 8600000 ms
 INFO [main] (ScriptRunner.java:378) - Test script activated
 INFO [main] (GUI.java:1768) - Visualized plugin was not started: class se.sics.cooja.plugins.
      Visualizer
 INFO [main] (GUI.java:1768) - Visualized plugin was not started: class se.sics.cooja.plugins.
      Visualizer
 INFO [Thread-2] (Simulation.java:252) - Simulation main loop started, system time:
      1362927241931
INFO: compiling with CoAP-07
TARGET not defined, using target 'native'
sudo ../../tools/tunslip6 -a 127.0.0.1 aaaa::1/64
ifconfig tun0 inet `hostname` up
 INFO [Thread-3] (SerialSocketServer.java:178) - Forwarder: socket -> serial port
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.0.1  P-t-P:127.0.0.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Server IPv6 addresses:
 aaaa::212:7401:1:101
route1: 0101
```

```
 fe80::212:7401:1:101
route2: 0101
Server IPv6 addresses:
 aaaa::212:7401:1:101
route1: 0101
 fe80::212:7401:1:101
route2: 0101
event-r-1: 03->03 -273
r-2: 05->05 -273
r-3: 0f->05 -265
r-4: 02->02 -272
r-5: 11->02 -257
r-6: 0a->05 -256
r-7: 04->04 -277
r-8: 1f->06 -271
r-9: 0c->02 -276
r-10: 08->03 -267
r-11: 06->06 -276
r-12: 1a->06 -256
r-13: 18->04 -267
r-14: 1e->05 -264
r-15: 15->06 -270
r-16: 21->03 -266
r-17: 1c->03 -267
r-18: 07->02 -257
r-19: 20->02 -257
r-20: 13->04 -267
r-21: 19->05 -267
r-22: 1b->02 -256
r-23: 17->03 -265
r-24: 14->05 -257
r-25: 10->06 -269
r-26: 0e->04 -267
r-27: 23->05 -264
r-28: 0b->06 -269
r-29: 12->03 -266
r-30: 09->04 -267
r-31: 24->06 -270
r-32: 0d->03 -268
r-33: 22->04 -265
r-34: 1d->04 -266
r-35: 16->02 -264
Incoming request from /trendy/rep/trendy/rep
Added Resource: URL[trendy/gl];[,]; Cache[ for 0 minutes at 1362927409057]
************************************************

        Analysis of Registry for Grouping:

************************************************
New GL: so checking
Node Node{URI[/trendy/rep], glIP[/aaaa:0:0:0:212:7405:5:505] Active[true(1)]  trendy[1/1]
     GlChanged[false] location[jmf04] batteryLevel[0] currentNumberOfResources[1] Active Since
     [1362927409045]}

************************************************
Incoming request from /trendy/rep/trendy/rep
Added Resource: URL[trendy/gl];[,]; Cache[ for 0 minutes at 1362927412931]
************************************************

        Analysis of Registry for Grouping:

************************************************
New GL: so checking
Node Node{URI[/trendy/rep], glIP[/aaaa:0:0:0:212:7404:4:404] Active[true(1)]  trendy[1/1]
     GlChanged[false] location[rut03] batteryLevel[0] currentNumberOfResources[1] Active Since
     [1362927412922]}

************************************************
Incoming request from /trendy/rep/trendy/rep
Added Resource: URL[trendy/gl];[,]; Cache[ for 0 minutes at 1362927417711]
************************************************
```

B.2 COOJA Log

The COOJA generates following log files for a 6LoWPAN running in its environment. This section explains the log files briefly and presents few examples.

B.2.1 Case-x-packet.pcap

This file holds all the network traffic sniffed in a 6LoWPAN. The network traffic validation is done by checking different details of the sniffed traffic using Wireshark [98]. Figure B.1 shows a snapshot to illustrate the coarse detail of each packet in a Wireshark GUI.

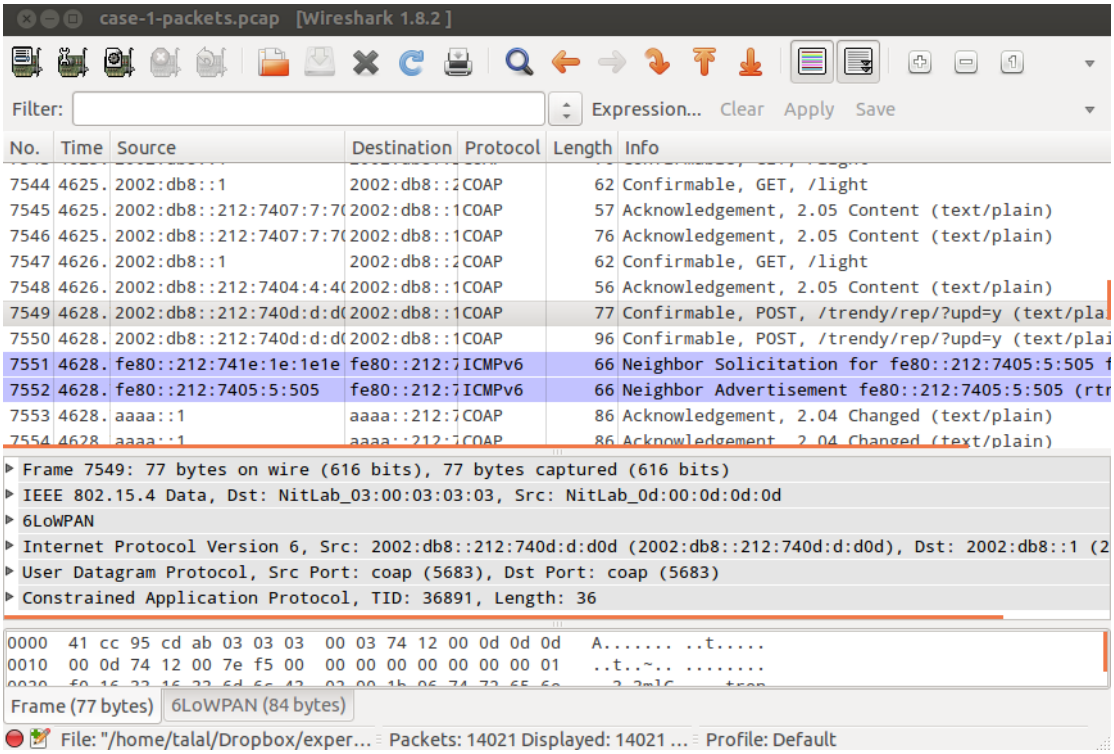


Figure B.1: Sniffed traffic of a 6LoWPAN

B.2.2 Lowpan-detail.log

This file holds all the log details generated by each SA for debugging and validation. An example is shown in Listing B.2.

Listing B.2: 6LoWPAN detailed log carrying all output of 35 SAs for Validation

```
Random seed: -1948817365486295399
2 ------------------------------------
#Time: calculated_energy_consumption calculated_cpu_energy_consumption
    calculated_lpm_energy_consumption calculated_listen_energy_consumption
    calculated_transmit_energy_consumption
4 #Time: i energy_value[i] cpu_value[i] lpm_value[i] listen_value[i] transmit_value[i]
#Time: Total Trendy Packets: UPD GROUPING
6 #Time: Node-ID: Total Trendy Packets: UPD GROUPING
```

```
 8  2 at 281872:Rime started with address 0.18.116.2.0.2.2.2
10  6 at 283119:Rime started with address 0.18.116.6.0.6.6.6
12  19 at 291251:MAC 00:12:74:13:00:13:13:13 Contiki 2.6 started. Node id is set to 19.
14  2 at 292046:MAC 00:12:74:02:00:02:02:02 Contiki 2.6 started. Node id is set to 2.
16  6 at 293291:MAC 00:12:74:06:00:06:06:06 Contiki 2.6 started. Node id is set to 6.
18  27 at 293331:Rime started with address 0.18.116.27.0.27.27.27
20  19 at 298739:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
22  2 at 299467:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
24  6 at 300714:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
26  27 at 305099:MAC 00:12:74:1b:00:1b:1b:1b Contiki 2.6 started. Node id is set to 27.
28  27 at 312586:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
30  2 at 313978:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7402:0002:0202
32  19 at 314543:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7413:0013:1313
34  2 at 314981:Starting 'GL'
36  14 at 315067:Rime started with address 0.18.116.14.0.14.14.14
38  6 at 315225:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7406:0006:0606
40  19 at 315546:Starting 'GM'
42  6 at 316228:Starting 'GL'
44  14 at 325603:MAC 00:12:74:0e:00:0e:0e:0e Contiki 2.6 started. Node id is set to 14.
46  27 at 328394:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:741b:001b:1b1b
48  27 at 329397:Starting 'GM'
50  14 at 333090:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
52  36 at 337112:Rime started with address 0.18.116.36.0.36.36.36
54  14 at 347678:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:740e:000e:0e0e
56  14 at 348681:Starting 'GM'
58  36 at 348876:MAC 00:12:74:24:00:24:24:24 Contiki 2.6 started. Node id is set to 36.
60  36 at 356363:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
62  8 at 361327:Rime started with address 0.18.116.8.0.8.8.8
64  12 at 361714:Rime started with address 0.18.116.12.0.12.12.12
66  8 at 371503:MAC 00:12:74:08:00:08:08:08 Contiki 2.6 started. Node id is set to 8.
68  36 at 372165:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7424:0024:2424
70  12 at 372259:MAC 00:12:74:0c:00:0c:0c:0c Contiki 2.6 started. Node id is set to 12.
72  36 at 373172:Starting 'GM'
74  8 at 378924:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
76  12 at 379747:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
78  34 at 390495:Rime started with address 0.18.116.34.0.34.34.34
80  8 at 393437:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7408:0008:0808
82  12 at 394338:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:740c:000c:0c0c
84  8 at 394440:Starting 'GM'
86  12 at 395341:Starting 'GM'
```

```
88  34 at 402255:MAC 00:12:74:22:00:22:22:22 Contiki 2.6 started. Node id is set to 34.

90  34 at 409742:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26

92  34 at 425547:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7422:0022:2222

94  34 at 426551:Starting 'GM'

96  13 at 432660:Rime started with address 0.18.116.13.0.13.13.13

98  13 at 443208:MAC 00:12:74:0d:00:0d:0d:0d Contiki 2.6 started. Node id is set to 13.

100 13 at 450695:CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26

102 7 at 453583:Rime started with address 0.18.116.7.0.7.7.7

104 7 at 463762:MAC 00:12:74:07:00:07:07:07 Contiki 2.6 started. Node id is set to 7.

106 13 at 465283:Tentative link-local IPv6 address fe80:0000:0000:0000:0212:740d:000d:0d0d

108 3 at 466279:Rime started with address 0.18.116.3.0.3.3.3

110 13 at 466286:Starting 'GM'
    .
112 .
    .
114 1 at 109235400:Server IPv6 addresses:

116 1 at 109241677: aaaa::212:7401:1:101

118 1 at 109243638:route1: 0101

120 1 at 109249930: fe80::212:7401:1:101

122 1 at 109251891:route2: 0101

124 1 at 109255367:event-r-1: 03->03 -273

126 1 at 109258519:r-2: 05->05 -273

128 1 at 109261588:r-3: 0f->05 -265

130 1 at 109264739:r-4: 02->02 -272

132 1 at 109268113:r-5: 11->02 -257

134 1 at 109271183:r-6: 0a->05 -256

136 1 at 109274263:r-7: 04->04 -277

138 1 at 109277639:r-8: 1f->06 -271

140 1 at 109280791:r-9: 0c->02 -276

142 1 at 109284228:r-10: 08->03 -267

144 1 at 109287748:r-11: 06->06 -276

146 1 at 109291492:r-12: 1a->06 -256

148 1 at 109295317:r-13: 18->04 -267

150 1 at 109299061:r-14: 1e->05 -264

152 1 at 109302826:r-15: 15->06 -270

154 1 at 109306568:r-16: 21->03 -266

156 1 at 109310312:r-17: 1c->03 -267

158 1 at 109313749:r-18: 07->02 -257

160 1 at 109317491:r-19: 20->02 -257

162 1 at 109321315:r-20: 13->04 -267

164 1 at 109325058:r-21: 19->05 -267

166 1 at 109328839:r-22: 1b->02 -256
```

```
168 1 at 109332582:r-23: 17->03 -265

170 1 at 109336406:r-24: 14->05 -257

172 1 at 109340149:r-25: 10->06 -269

174 1 at 109343669:r-26: 0e->04 -267

176 1 at 109347412:r-27: 23->05 -264

178 1 at 109350932:r-28: 0b->06 -269

180 1 at 109354674:r-29: 12->03 -266

182 1 at 109358194:r-30: 09->04 -267

184 1 at 109361939:r-31: 24->06 -270

186 1 at 109365416:r-32: 0d->03 -268

188 1 at 109369159:r-33: 22->04 -265

190 1 at 109372904:r-34: 1d->04 -266

192 1 at 109376664:r-35: 16->02 -264

194 5 at 128798244:p(l=JMF04,b=0,mgm=12,r=trendy/gl)l(30/30) to

196 5 at 129033677:t=420

198 5 at 129034883:t420

200 5 at 129036884:newrep 211|1

202 4 at 131968960:p(l=RUT03,b=0,mgm=12,r=trendy/gl)l(30/30) to

204 4 at 132204991:t=420

206 4 at 132206192:t420

208 4 at 132208193:newrep 335|1

210 2 at 135317971:p(l=INB01,b=0,mgm=12,r=trendy/gl)l(30/30) to

212 2 at 135676994:t=420

214 2 at 135678194:t420

216 2 at 135680267:newrep 415|1

218 6 at 141319263:p(l=SSB05,b=0,mgm=12,r=trendy/gl)l(30/30) to

220 6 at 141553669:t=420
    .
222 .
    .
224 ----------Time = 4201308746----------
    Total Nodes = 35:
226 Total reports = 35:
    Total Update Messages =184
228 Total Grouping Messages =60
    Total CPU Consumption =2239 [4%]Total LPM Consumption =1135 [2%]Total LISTEN Consumption
        =58981 [93%]Total TRANSMIT Consumption =1405 [2%]Total Energy Consumption
        =63.76076319272461 Joules [milli:63760.76319272461----------
230 lAllEnergyDescfile:4200 63760.76319272461   2239.430923461914   1134.8447906249999
        58981.22314453125   1405.2643341064456
    lAllPacketDescfile:4200 244 184 60
```

B.2.3 l-energy-all.log

It contains aggregated energy values (transmit, listening, Low Power listening and CPU) for 35 SAs at the end of each time window. An example is shown in Listing B.3.

Listing B.3: 6LoWPAN log for aggregated energy consumption of 35 SAs in each time window

```
1
Format (separated by tabs):
3 Seconds: 420
Total Energy Consumption: 8105 milli joules
5 Energy spent by CPU: 316
Energy spent during LPM:    112
7 Energy spent during Listening: 6933
Energy spent during Transmitting: 742
9 ----------------------------------------
420 8105.504205926513   316.25917053222656  112.99217894897458  6933.3306884765625
    742.9221679687498
11 840 14572.623792242432  552.2787322998047   226.4046973937988   12910.039672851562
    883.9006896972656
1260    20905.71028399658   771.89208984375 339.91303363037105  18786.626586914062
    1007.2785736083985
13 1680    26942.38112759399   975.0123138427734   453.5077663879394   24477.330322265625
    1036.5307250976562
2100    33135.47928929443   1187.9664459228516  567.0529831420898   30265.517578125
    1114.9422821044918
15 2520    39211.80095956421   1393.7349243164062  680.6337140808105   35980.294189453125
    1157.1381317138673
2940    45349.58579152222   1603.0055694580078  794.1971989929197   41737.81494140625
    1214.5680816650388
17 3360    51494.22161099853   1815.6357879638672  907.7424514526366   47486.21887207031
    1284.6244995117188
3780    57521.531957537845  2019.6591796875 1021.3347224304201  53172.27355957031
    1308.2644958496094
19 4200    63760.76319272461   2239.430923461914   1134.8447906249999  58981.22314453125
    1405.2643341064456
4620    69760.76371766967   2442.6050720214844  1248.4407440368655  64643.37890625
    1426.3389953613282
21 5040    75846.91823616944   2651.5826568603516  1362.00530892334    70358.22875976562
    1475.1015106201176
5460    82036.11023864135   2869.404739379883   1475.5217346130373  76140.12817382812
    1551.055590820312
23 5880    87991.41860117798   3071.8158416748047  1589.1219367492677  81769.37255859375
    1561.108264160156
6300    94287.65106833496   3297.1075744628906  1702.601230078125   87620.30456542969
    1667.637698364258
25 6720    100253.06200737305  3501.946060180664   1816.1882463867187  93252.38159179688
    1682.546109008789
7140    106425.43900129394  3719.647018432617   1929.7102995117184  99021.96350097656
    1754.118182373047
27 7560    112524.26618941041  3934.2019958496094  2043.2941953918455  104740.17700195312
    1806.5929962158207
7980    118534.7614795166   4143.127258300781   2156.8605884033204  110407.45422363281
    1827.3194091796877
29 8400    124783.54844002075  4365.155868530273   2270.360759967041   116223.25927734375
    1924.7725341796872
```

B.2.4 l-energy-ind.log

It carrys the individual energy values for each of the 35 SAs at the end of each time window. An example is shown in Listing B.4.

Listing B.4: 6LoWPAN log for individual energy for 35 SAs in each time window

```
1
Similar format like l-energy-all.log
3 Only second column represents node number here
----------------------------------------
5 420 2   294.65434200439455  12.94683837890625   3.2083703247070314  246.6943359375
    31.804797363281253
420 6   286.28680442504884  12.582366943359375  3.2101168029785154  243.065185546875
    27.429135131835935
7 420 19  221.25385037841795  8.291336059570312   3.232118811035156   190.9954833984375
    18.734912109375003
420 27  217.24769946899414  8.09454345703125    3.233157843017578   187.7215576171875
    18.198440551757812
```

```
 9  420 14  217.15902416381834  8.29083251953125    3.2321442810058594  186.024169921875
        19.61187744140625
    420 36  215.18124539794923  8.013427734375  3.2335708374023433  187.3297119140625
        16.604534912109376
11  420 8   214.82246350708007  8.112579345703125   3.233053106689453   185.9600830078125
        17.516748046875
    420 12  218.85444497680663  7.95574951171875    3.233885284423828   190.777587890625
        16.887222290039062
13  420 34  220.88778381958008  8.424636840820312   3.2314392150878906  188.6004638671875
        20.631243896484378
    420 13  222.65375192871096  8.373733520507812   3.2317212890625 191.220703125
        19.827593994140628
15  420 7   212.47223981323242  7.7936553955078125  3.2349931091308592  185.2606201171875
        16.182971191406253
    420 3   284.5271376708984   12.743820190429688  3.2092696289062497  239.794921875
        28.779125976562497
17  420 32  228.00281817626953  8.849990844726562   3.229228088378906   193.79150390625
        22.132095336914062
    420 21  227.58744158935545  8.330429077148438   3.2319179077148434  199.8724365234375
        16.15265808105469
19  420 11  218.19248501586915  8.262222290039062   3.2322951965332027  186.895751953125
        19.802215576171875
    420 22  218.1124487915039   8.476593017578125   3.231827453613281   183.7664794921875
        22.637548828125
21  420 30  229.02147634277344  8.974868774414062   3.2285106445312497  193.319091796875
        23.499005126953126
    420 35  222.21827649536135  8.667205810546875   3.230791754150391   185.8026123046875
        24.51766662597656
23  420 5   287.69299608764646  12.948394775390625  3.208373895263672   240.6866455078125
        30.84958190917969
    420 31  216.99980755004884  8.057418823242188   3.2333463684082027  187.7545166015625
        17.954525756835938
25  420 16  219.37035444946290  8.466888427734375   3.231212603759766   186.1541748046875
        21.51807861328125
    420 20  214.85796419677735  7.9071807861328125  3.234111657714844   188.34228515625
        15.374386596679688
27  420 24  219.11335708007815  8.179092407226562   3.232714379882812   189.708251953125
        17.99329833984375
    420 4   290.04208698120120  13.089889526367188  3.207461737060547   243.343505859375
        30.401229858398438
29  420 28  218.65594456787110  8.349884033203125   3.2318112670898436  186.9195556640625
        20.154693603515625
    420 10  225.17402451782226  8.395111083984375   3.231602032470703   192.5445556640625
        21.00275573730469
31  420 23  217.70547147216797  8.06744384765625    3.2332851928710933  189.5379638671875
        16.866778564453128
    420 15  235.88801070556642  8.603622436523438   3.230512536621094   206.180419921875
        17.873455810546876
33  420 33  230.94909391479490  9.13165283203125    3.227752972412109   194.410400390625
        24.17928771972656
    420 26  216.23162774047850  8.088180541992188   3.2331749816894533  186.9635009765625
        17.946771240234376
35  420 17  237.27476727905272  9.469985961914062   3.2259238952636715  199.3414306640625
        25.2374267578125
    420 29  218.02615773925783  8.253067016601562   3.23231923828125    186.8865966796875
        19.6541748046875
37  420 25  226.89658725585937  8.563613891601562   3.2307120117187496  192.7276611328125
        22.374600219726563
    420 9   241.71109421997070  9.192169189453125   3.2274363830566406  205.8270263671875
        23.464462280273438
39  420 18  219.77912620239258  8.310745239257812   3.232016217041015   189.1094970703125
        19.12686767578125
```

B.2.5 l-packet-all.log

It holds the total TRENDY packets sent by all the 35 SAs for every time window. An example is shown in Listing B.5.

Listing B.5: 6LoWPAN log for total TRENDY messages for all 35 SAs in each time window

```
1
Format (separated by tabs):
```

```
Seconds: 420
Total TRENDY messages:  97
Total UPDs: 37
Total Grouping: 60
-----------------------------------------
420 97  37  60
840 130 70  60
1260    151 91  60
1680    170 110 60
2100    184 124 60
2520    201 141 60
2940    213 153 60
3360    222 162 60
3780    234 174 60
4200    244 184 60
4620    254 194 60
5040    266 206 60
5460    276 216 60
5880    281 221 60
6300    293 233 60
6720    300 240 60
7140    307 247 60
7560    314 254 60
7980    324 264 60
8400    329 269 60
```

B.2.6 l-packet-ind.log

It stores the TRENDY packets information separately for each SA. An example is shown in Listing B.6.

Listing B.6: 6LoWPAN log for total TRENDY messages for each 35 SAs in each time window

```

Similar format like l-packet-all.log
Only second column represents node number here
---------------------------------------
420 2   13  1   12
420 6   14  2   12
420 19  1   1   0
420 27  1   1   0
420 14  1   1   0
420 36  1   1   0
420 8   1   1   0
420 12  1   1   0
420 34  1   1   0
420 13  1   1   0
420 7   1   1   0
420 3   13  1   12
420 32  1   1   0
420 21  1   1   0
420 11  1   1   0
420 22  1   1   0
420 30  1   1   0
420 35  1   1   0
420 5   14  2   12
420 31  1   1   0
420 16  1   1   0
420 20  1   1   0
420 24  1   1   0
420 4   13  1   12
420 28  1   1   0
420 10  1   1   0
420 23  1   1   0
420 15  1   1   0
420 33  1   1   0
420 26  1   1   0
420 17  1   1   0
420 29  1   1   0
420 25  1   1   0
420 9   1   1   0
420 18  1   1   0
```

B.3 DA Log

The DA generates few log files to record different level of details. This section explains the log files briefly and presents few examples.

B.3.1 daperformance.log

It contains very general information about the results, which is used for graphs after some processing. Following is an example of such a log file. An example is shown in Listing B.7.

Listing B.7: DA Log for graph generation after further processing

```
1
  Format (separated by tabs):
3 Seconds: 420
  Total UPD received: 37
5 Total grouping messages sent: 30
  Total nodes registered: 35
7 Total GLs: 5
  Nodes in groups: 30
9 ----------------------------------------
  420 37  30  0   35  5   30
11 840 40  30  0   35  5   30
  1260    43  30  0   35  5   30
13 1680    45  30  0   35  5   30
  2100    48  30  0   35  5   30
15 2520    50  30  0   35  5   30
  2940    50  30  0   35  5   30
17 3360    54  30  0   35  5   30
  3780    55  30  0   35  5   30
19 4200    55  30  0   35  5   30
  4620    56  30  0   35  5   30
21 5040    59  30  0   35  5   30
  5460    60  30  0   35  5   30
23 5880    60  30  0   35  5   30
  6300    61  30  0   35  5   30
25 6720    63  30  0   35  5   30
  7140    64  30  0   35  5   30
27 7560    65  30  0   35  5   30
  7980    65  30  0   35  5   30
29 8400    65  30  0   35  5   30
```

B.3.2 da-detail.log

This file holds some details about the status of the DA including number of nodes registers at certain time (every minute) etc. Following is an example of this log file. An example is shown in Listing B.8.

Listing B.8: Some of the DA detail

```
1 Only Showing some of the log
  ------------------------------
3 2013/03/10 14:53:59
  Time:0,Updates:0,Grouping Messages:0,Trendy Change Messages:0,Total Nodes:0,Total GLs:0,Nodes
      in groups:0
5
  Mode=4[Basic SD: with timer and grouping]:TimeWindow{420} Grouping{true} Trendy{1,9} Adaptive{
      Enabled[true], threshold[1], step[2], Limit[1], RetainThreshold[1]} PPUB{ Enabled[false],
      Threshold[0], Limit[0], RetainThreshold[0]}
7 Time:70,Updates:0,Grouping Messages:0,Trendy Change Messages:0,Total Nodes:0,Total GLs:0,Nodes
      in groups:0
```

```
Time:141,Updates:0,Grouping Messages:0,Trendy Change Messages:0,Total Nodes:0,Total GLs:0,
    Nodes in groups:0

Time:211,Updates:10,Grouping Messages:5,Trendy Change Messages:0,Total Nodes:10,Total GLs:5,
    Nodes in groups:5

Time:282,Updates:28,Grouping Messages:23,Trendy Change Messages:0,Total Nodes:28,Total GLs:5,
    Nodes in groups:23

Time:352,Updates:35,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:423,Updates:35,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:494,Updates:36,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Mode=4[Basic SD: with timer and grouping]:TimeWindow{420} Grouping{true} Trendy{1,9} Adaptive{
    Enabled[true], threshold[1], step[2], Limit[1], RetainThreshold[1]} PPUB{ Enabled[false],
    Threshold[0], Limit[0], RetainThreshold[0]}

Time:420,Updates:37,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:564,Updates:37,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:635,Updates:39,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:705,Updates:40,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:776,Updates:40,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:847,Updates:40,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:917,Updates:40,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:988,Updates:40,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Mode=4[Basic SD: with timer and grouping]:TimeWindow{420} Grouping{true} Trendy{1,9} Adaptive{
    Enabled[true], threshold[1], step[2], Limit[1], RetainThreshold[1]} PPUB{ Enabled[false],
    Threshold[0], Limit[0], RetainThreshold[0]}

GL[aaaa:0:0:0:212:7403:3:303/aaaa:0:0:0:212:7403:3:303]: 6 GMs Active for GL:1th[/aaaa
    :0:0:0:212:740d:d:d0d], 2th[/aaaa:0:0:0:212:7408:8:808], 3th[/aaaa
    :0:0:0:212:7417:17:1717], 4th[/aaaa:0:0:0:212:7412:12:1212], 5th[/aaaa:0:0:0:212:741c:1c:1
    c1c], 6th[/aaaa:0:0:0:212:7421:21:2121],

GL[aaaa:0:0:0:212:7405:5:505/aaaa:0:0:0:212:7405:5:505]: 6 GMs Active for GL:1th[/aaaa
    :0:0:0:212:741e:1e:1e1e], 2th[/aaaa:0:0:0:212:7419:19:1919], 3th[/aaaa
    :0:0:0:212:7423:23:2323], 4th[/aaaa:0:0:0:212:740a:a:a0a], 5th[/aaaa
    :0:0:0:212:7414:14:1414], 6th[/aaaa:0:0:0:212:740f:f:f0f],

GL[aaaa:0:0:0:212:7406:6:606/aaaa:0:0:0:212:7406:6:606]: 6 GMs Active for GL:1th[/aaaa
    :0:0:0:212:741f:1f:1f1f], 2th[/aaaa:0:0:0:212:741a:1a:1a1a], 3th[/aaaa
    :0:0:0:212:7424:24:2424], 4th[/aaaa:0:0:0:212:7410:10:1010], 5th[/aaaa:0:0:0:212:740b:b:
    b0b], 6th[/aaaa:0:0:0:212:7415:15:1515],

GL[aaaa:0:0:0:212:7404:4:404/aaaa:0:0:0:212:7404:4:404]: 6 GMs Active for GL:1th[/aaaa
    :0:0:0:212:740e:e:e0e], 2th[/aaaa:0:0:0:212:7418:18:1818], 3th[/aaaa
    :0:0:0:212:7413:13:1313], 4th[/aaaa:0:0:0:212:7422:22:2222], 5th[/aaaa:0:0:0:212:741d:1d:1
    d1d], 6th[/aaaa:0:0:0:212:7409:9:909],

GL[aaaa:0:0:0:212:7402:2:202/aaaa:0:0:0:212:7402:2:202]: 6 GMs Active for GL:1th[/aaaa
    :0:0:0:212:7407:7:707], 2th[/aaaa:0:0:0:212:7411:11:1111], 3th[/aaaa:0:0:0:212:740c:c:c0c
    ], 4th[/aaaa:0:0:0:212:741b:1b:1b1b], 5th[/aaaa:0:0:0:212:7416:16:1616], 6th[/aaaa
    :0:0:0:212:7420:20:2020],

Time:840,Updates:40,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30

Time:1058,Updates:41,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
    Nodes in groups:30
```

```
55 Time:1129,Updates:41,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
       Nodes in groups:30

57 Time:1200,Updates:41,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
       Nodes in groups:30

59 Time:1270,Updates:41,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
       Nodes in groups:30

61 Time:1341,Updates:42,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
       Nodes in groups:30

63 Time:1411,Updates:43,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
       Nodes in groups:30

65 Time:1482,Updates:43,Grouping Messages:30,Trendy Change Messages:0,Total Nodes:35,Total GLs:5,
       Nodes in groups:30

67 Mode=4[Basic SD: with timer and grouping]:TimeWindow{420} Grouping{true} Trendy{1,9} Adaptive{
       Enabled[true], threshold[1], step[2], Limit[1], RetainThreshold[1]} PPUB{ Enabled[false],
       Threshold[0], Limit[0], RetainThreshold[0]}

69 GL[aaaa:0:0:0:212:7403:3:303/aaaa:0:0:0:212:7403:3:303]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:740d:d:d0d], 2th[/aaaa:0:0:0:212:7408:8:808], 3th[/aaaa
       :0:0:0:212:7417:17:1717], 4th[/aaaa:0:0:0:212:7412:12:1212], 5th[/aaaa:0:0:0:212:741c:1c:1
       c1c], 6th[/aaaa:0:0:0:212:7421:21:2121],

71 GL[aaaa:0:0:0:212:7405:5:505/aaaa:0:0:0:212:7405:5:505]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:741e:1e:1e1e], 2th[/aaaa:0:0:0:212:7419:19:1919], 3th[/aaaa
       :0:0:0:212:7423:23:2323], 4th[/aaaa:0:0:0:212:740a:a:a0a], 5th[/aaaa
       :0:0:0:212:7414:14:1414], 6th[/aaaa:0:0:0:212:740f:f:f0f],

73 GL[aaaa:0:0:0:212:7406:6:606/aaaa:0:0:0:212:7406:6:606]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:741f:1f:1f1f], 2th[/aaaa:0:0:0:212:741a:1a:1a1a], 3th[/aaaa
       :0:0:0:212:7424:24:2424], 4th[/aaaa:0:0:0:212:7410:10:1010], 5th[/aaaa:0:0:0:212:740b:b:
       b0b], 6th[/aaaa:0:0:0:212:7415:15:1515],

75 GL[aaaa:0:0:0:212:7404:4:404/aaaa:0:0:0:212:7404:4:404]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:740e:e:e0e], 2th[/aaaa:0:0:0:212:7418:18:1818], 3th[/aaaa
       :0:0:0:212:7413:13:1313], 4th[/aaaa:0:0:0:212:7422:22:2222], 5th[/aaaa:0:0:0:212:741d:1d:1
       d1d], 6th[/aaaa:0:0:0:212:7409:9:909],

77 GL[aaaa:0:0:0:212:7402:2:202/aaaa:0:0:0:212:7402:2:202]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:7407:7:707], 2th[/aaaa:0:0:0:212:7411:11:1111], 3th[/aaaa:0:0:0:212:740c:c:c0c
       ], 4th[/aaaa:0:0:0:212:741b:1b:1b1b], 5th[/aaaa:0:0:0:212:7416:16:1616], 6th[/aaaa
       :0:0:0:212:7420:20:2020],

79 GL[aaaa:0:0:0:212:7403:3:303/aaaa:0:0:0:212:7403:3:303]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:740d:d:d0d], 2th[/aaaa:0:0:0:212:7408:8:808], 3th[/aaaa
       :0:0:0:212:7417:17:1717], 4th[/aaaa:0:0:0:212:7412:12:1212], 5th[/aaaa:0:0:0:212:741c:1c:1
       c1c], 6th[/aaaa:0:0:0:212:7421:21:2121],

81 GL[aaaa:0:0:0:212:7405:5:505/aaaa:0:0:0:212:7405:5:505]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:741e:1e:1e1e], 2th[/aaaa:0:0:0:212:7419:19:1919], 3th[/aaaa
       :0:0:0:212:7423:23:2323], 4th[/aaaa:0:0:0:212:740a:a:a0a], 5th[/aaaa
       :0:0:0:212:7414:14:1414], 6th[/aaaa:0:0:0:212:740f:f:f0f],

83 GL[aaaa:0:0:0:212:7406:6:606/aaaa:0:0:0:212:7406:6:606]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:741f:1f:1f1f], 2th[/aaaa:0:0:0:212:741a:1a:1a1a], 3th[/aaaa
       :0:0:0:212:7424:24:2424], 4th[/aaaa:0:0:0:212:7410:10:1010], 5th[/aaaa:0:0:0:212:740b:b:
       b0b], 6th[/aaaa:0:0:0:212:7415:15:1515],

85 GL[aaaa:0:0:0:212:7404:4:404/aaaa:0:0:0:212:7404:4:404]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:740e:e:e0e], 2th[/aaaa:0:0:0:212:7418:18:1818], 3th[/aaaa
       :0:0:0:212:7413:13:1313], 4th[/aaaa:0:0:0:212:7422:22:2222], 5th[/aaaa:0:0:0:212:741d:1d:1
       d1d], 6th[/aaaa:0:0:0:212:7409:9:909],

87 GL[aaaa:0:0:0:212:7402:2:202/aaaa:0:0:0:212:7402:2:202]: 6 GMs Active for GL:1th[/aaaa
       :0:0:0:212:7407:7:707], 2th[/aaaa:0:0:0:212:7411:11:1111], 3th[/aaaa:0:0:0:212:740c:c:c0c
       ], 4th[/aaaa:0:0:0:212:741b:1b:1b1b], 5th[/aaaa:0:0:0:212:7416:16:1616], 6th[/aaaa
       :0:0:0:212:7420:20:2020],
```

B.3.3 daFullDetail.log

This file contains the most of the log of different classes in the DA. It is used for the debugging and validation. Following code snippet shows an example of this log file. An example is shown in Listing B.9.

Listing B.9: Full log generated by DA classes

```
Log for all of the DA classes. Again just showing the one page as an example
_____________________________________
2013-03-10 14:53:59 [util.Log] INFO - ==[ START-UP
    ]=========================================================
2013-03-10 14:56:49 [layers.TokenLayer] INFO - Incoming request: [aaaa
    :0:0:0:212:7405:5:505]:5683#--
2013-03-10 14:56:49 [endpoint.LocalEndpoint] INFO - Dispatching execution: /trendy/rep
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - String payload [l=JMF04,b=0,mgm=12,r=
    trendy/gl], size (30)
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Context[JMF04]
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Battery[0]
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - gm[12]
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Total Registry Nodes = 1
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - DA Registry Nodes:
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Location(JMF04) has 1 records:
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Byte[0]: 164
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Byte[1]: 1
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - DEBUG chk Timer bytes to int = 420

2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Resource: URL[trendy/gl];[,]; Cache[ for
    0 minutes at 1362927409057]
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Making it GL
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - New GL: GL{ IP[/aaaa
    :0:0:0:212:7405:5:505], location[jmf04], maxGM[12], confirmed GM[0], capability[122.0],
    rank[0.0]

2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - as Node: Node{URI[/trendy/rep], glIP[/
    aaaa:0:0:0:212:7405:5:505] Active[true(0)]  trendy[1/1] GlChanged[false] location[jmf04]
    batteryLevel[0] currentNumberOfResources[1] Active Since[1362927409045]}

2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Node's UPD Message # 1 of 1
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - trendy = 1

2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Sending code =65
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - New GL added
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - Asking for Normal behaviour = 0
2013-03-10 14:56:49 [trendy.da.ReportResource] INFO - payload([B@11c55bb) of 3bytes
2013-03-10 14:56:49 [layers.TokenLayer] INFO - Responding request: [aaaa
    :0:0:0:212:7405:5:505]:5683#--
2013-03-10 14:56:52 [layers.TokenLayer] INFO - Incoming request: [aaaa
    :0:0:0:212:7404:4:404]:5683#--
2013-03-10 14:56:52 [endpoint.LocalEndpoint] INFO - Dispatching execution: /trendy/rep
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - String payload [l=RUT03,b=0,mgm=12,r=
    trendy/gl], size (30)
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Context[RUT03]
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Battery[0]
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - gm[12]
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Total Registry Nodes = 2
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - DA Registry Nodes:
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Location(RUT03) has 1 records:
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Byte[0]: 164
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Byte[1]: 1
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - DEBUG chk Timer bytes to int = 420

2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Resource: URL[trendy/gl];[,]; Cache[ for
    0 minutes at 1362927412931]
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Making it GL
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - New GL: GL{ IP[/aaaa
    :0:0:0:212:7404:4:404], location[rut03], maxGM[12], confirmed GM[0], capability[122.0],
    rank[0.0]

2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - as Node: Node{URI[/trendy/rep], glIP[/
    aaaa:0:0:0:212:7404:4:404] Active[true(0)]  trendy[1/1] GlChanged[false] location[rut03]
    batteryLevel[0] currentNumberOfResources[1] Active Since[1362927412922]}

2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Node's UPD Message # 1 of 2
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - trendy = 1
```

```
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Sending code =65
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - New GL added
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - Asking for Normal behaviour = 0
2013-03-10 14:56:52 [trendy.da.ReportResource] INFO - payload([B@1d31859) of 3bytes
2013-03-10 14:56:52 [layers.TokenLayer] INFO - Responding request: [aaaa
    :0:0:0:212:7404:4:404]:5683#--
2013-03-10 14:56:57 [layers.TokenLayer] INFO - Incoming request: [aaaa
    :0:0:0:212:7402:2:202]:5683#--
2013-03-10 14:56:57 [endpoint.LocalEndpoint] INFO - Dispatching execution: /trendy/rep
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - String payload [l=INB01,b=0,mgm=12,r=
    trendy/gl], size (30)
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Context[INB01]
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Battery[0]
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - gm[12]
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Total Registry Nodes = 3
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - DA Registry Nodes:
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Location(INB01) has 1 records:
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Byte[0]: 164
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Byte[1]: 1
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - DEBUG chk Timer bytes to int = 420

2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Resource: URL[trendy/gl];[,]; Cache[ for
     0 minutes at 1362927417711]
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Making it GL
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - New GL: GL{ IP[/aaaa
    :0:0:0:212:7402:2:202], location[inb01], maxGM[12], confirmed GM[0], capability[122.0],
    rank[0.0]

2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - as Node: Node{URI[/trendy/rep], glIP[/
    aaaa:0:0:0:212:7402:2:202] Active[true(0)]  trendy[1/1] GlChanged[false] location[inb01]
    batteryLevel[0] currentNumberOfResources[1] Active Since[1362927417703]}

2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Node's UPD Message # 1 of 3
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - trendy = 1

2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Sending code =65
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - New GL added
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - Asking for Normal behaviour = 0
2013-03-10 14:56:57 [trendy.da.ReportResource] INFO - payload([B@15a07bf) of 3bytes
2013-03-10 14:56:57 [layers.TokenLayer] INFO - Responding request: [aaaa
    :0:0:0:212:7402:2:202]:5683#--
2013-03-10 14:57:05 [layers.TokenLayer] INFO - Incoming request: [aaaa
    :0:0:0:212:7406:6:606]:5683#--
2013-03-10 14:57:05 [endpoint.LocalEndpoint] INFO - Dispatching execution: /trendy/rep
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - String payload [l=SSB05,b=0,mgm=12,r=
    trendy/gl], size (30)
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Context[SSB05]
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Battery[0]
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - gm[12]
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Total Registry Nodes = 4
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - DA Registry Nodes:
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Location(SSB05) has 1 records:
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Byte[0]: 164
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Byte[1]: 1
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - DEBUG chk Timer bytes to int = 420

2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Resource: URL[trendy/gl];[,]; Cache[ for
     0 minutes at 1362927425199]
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Making it GL
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - New GL: GL{ IP[/aaaa
    :0:0:0:212:7406:6:606], location[ssb05], maxGM[12], confirmed GM[0], capability[122.0],
    rank[0.0]

2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - as Node: Node{URI[/trendy/rep], glIP[/
    aaaa:0:0:0:212:7406:6:606] Active[true(0)]  trendy[1/1] GlChanged[false] location[ssb05]
    batteryLevel[0] currentNumberOfResources[1] Active Since[1362927425189]}

2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Node's UPD Message # 1 of 4
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - trendy = 1

2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Sending code =65
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - New GL added
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - Asking for Normal behaviour = 0
2013-03-10 14:57:05 [trendy.da.ReportResource] INFO - payload([B@8f3d27) of 3bytes
2013-03-10 14:57:05 [layers.TokenLayer] INFO - Responding request: [aaaa
    :0:0:0:212:7406:6:606]:5683#--
2013-03-10 14:57:14 [layers.TokenLayer] INFO - Incoming request: [aaaa
    :0:0:0:212:7403:3:303]:5683#--
2013-03-10 14:57:14 [endpoint.LocalEndpoint] INFO - Dispatching execution: /trendy/rep
```

```
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - String payload [l=HSG02,b=0,mgm=12,r=
    trendy/gl], size (30)
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Context[HSG02]
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Battery[0]
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - gm[12]
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Total Registry Nodes = 5
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - DA Registry Nodes:
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Location(HSG02) has 1 records:
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Byte[0]: 164
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Byte[1]: 1
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - DEBUG chk Timer bytes to int = 420

2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Resource: URL[trendy/gl];[,]; Cache[ for
    0 minutes at 1362927434372]
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Making it GL
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - New GL: GL{ IP[/aaaa
    :0:0:0:212:7403:3:303], location[hsg02], maxGM[12], confirmed GM[0], capability[122.0],
    rank[0.0]

2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - as Node: Node{URI[/trendy/rep], glIP[/
    aaaa:0:0:0:212:7403:3:303] Active[true(0)]  trendy[1/1] GlChanged[false] location[hsg02]
    batteryLevel[0] currentNumberOfResources[1] Active Since[1362927434363]}

2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Node's UPD Message # 1 of 5
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - trendy = 1

2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Sending code =65
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - New GL added
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - Asking for Normal behaviour = 0
2013-03-10 14:57:14 [trendy.da.ReportResource] INFO - payload([B@1e6f0ef) of 3bytes
2013-03-10 14:57:14 [layers.TokenLayer] INFO - Responding request: [aaaa
    :0:0:0:212:7403:3:303]:5683#--
2013-03-10 14:57:20 [layers.TokenLayer] INFO - Incoming request: [aaaa:0:0:0:212:740e:e:e0e
    ]:5683#--
2013-03-10 14:57:20 [endpoint.LocalEndpoint] INFO - Dispatching execution: /trendy/rep
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - String payload [l=RUT03,b=0,r=light;
    humid;temp;], size (31)
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - Context[RUT03]
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - Battery[0]
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - Total Registry Nodes = 5
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - DA Registry Nodes:
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - Location(RUT03) has 2 records:
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - Byte[0]: 164
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - Byte[1]: 1
2013-03-10 14:57:20 [trendy.da.ReportResource] INFO - DEBUG chk Timer bytes to int = 420
```

B.4 UA Log

The UA generates following log files:

B.4.1 uaperformance-processed.log

It contains very general information about the results, which is used for graphs after some processing. An example is shown in Listing B.10.

Listing B.10: UA log for graph generation after further processing

```
Format:
Query#
Service Discovery delay:
Service invocation delay:
Optimal Service selection:
Group query:
------------------------------
1   21.507398   576.7593153500001   1   0
2   19.485132   500.78959975    1   0
3   14.615253   687.337319  1   0
4   17.641199   319.83119405    1   0
5   16.978385   527.99205795    1   0
```

```
13  6   20.059484   347.14612509999995  1   0
    7   22.514677   1287.6573075    1   0
15  8   14.014749   1084.5305371499999  1   0
    9   23.075649   572.03808725    1   0
17  10  23.555236   428.37224069999996  1   0
```

B.4.2 ua-detail.log

This file holds some details about the status of the UA including the details of queries and responses. An example is shown in Listing B.11.

Listing B.11: Full UA log including detail of queries and responses

```
2   2013/03/10 14:58:59
    Query[location=INB01&type=humid]:Response#1:aaaa:0:0:0:212:7416:16:1616;/humid;rt=;info=;,
4
    Query[location=INB01&type=humid]:Response#1:{237:795}30
6
    Time:1,Interval:1 secs,Query#:1,INB01,Service Discovery Delay:21.507398,Service Invocation
        Delay:678.540371,Optimal Service Selection:true,Group Service Selection:false
8
    Query[location=INB01&info=light]:Response#2:aaaa:0:0:0:212:7416:16:1616;/light;rt=;info=;,
10
    Query[location=INB01&info=light]:Response#2:{71:262}10
12
    Time:9,Interval:7 secs,Query#:2,INB01,Service Discovery Delay:19.485132,Service Invocation
        Delay:589.164235,Optimal Service Selection:true,Group Service Selection:false
14
    Query[location=INB01&info=light]:Response#3:aaaa:0:0:0:212:7416:16:1616;/light;rt=;info=;,
16
    Query[location=INB01&info=light]:Response#3:{328:1025}10
18
    Time:17,Interval:7 secs,Query#:3,INB01,Service Discovery Delay:14.615253,Service Invocation
        Delay:808.63214,Optimal Service Selection:true,Group Service Selection:false
20
    Query[location=HSG02&info=humid]:Response#4:aaaa:0:0:0:212:7421:21:2121;/humid;rt=;info=;,
22
    Query[location=HSG02&info=humid]:Response#4:{257:795}30
24
    Time:23,Interval:6 secs,Query#:4,HSG02,Service Discovery Delay:17.641199,Service Invocation
        Delay:376.271993,Optimal Service Selection:true,Group Service Selection:false
26
    Query[location=RUT03&type=l&info=light]:Response#5:aaaa:0:0:0:212:7409:9:909;/light;rt=;info
        =;,
28
    Query[location=RUT03&type=l&info=light]:Response#5:{280:869}10
```

Appendix C

Automation for statistics processing and graph generation

This chapter covers the detail of automation of raw statistics processing and the generation of graphs.

C.1 Data Processing

A bash script is written to process all the collected raw log files generated by the simulations to calculate the means and standard deviations for a set of experiments. Listing C.1 presents the code.

Listing C.1: Bash Script to process generated log files for mean and standard deviation values

```
#!/bin/bash

noOfArgumentsReqd=3;
if [ "$#" -ne "$noOfArgumentsReqd" ]; then
  echo "Usage: Need $noOfArgumentsReqd arguments"
  exit 1
fi

totalSimTime=$1
topologies=$2
numOfQueryVariations=$3

if [ "$4" == "" ]; then
numOfCases=4
  echo "Number of Cases: $numOfCases"
else
numOfCases=$4
  echo "Number of Cases: $numOfCases"
fi

if [ "$5" == "" ]; then
bFDIR=$PWD
  echo "Directory for: $bFDIR"
else
bFDIR=$5
  echo "Directory for: $bFDIR"
fi

cleanAllResults="";
cleanIndResults="";
cleanUaResults="";
```

```
cleanDaResults="";

#Configuring attributes
for ((i = 1; i <= $topologies; i++)); do
case "$i" in

    "1")
        tFDIR=$bFDIR/"top1";
        ;;
    "2")
        tFDIR=$bFDIR/"top2";
        ;;
    "3")
        tFDIR=$bFDIR/"top3";
        ;;
    *)
        echo "Ending"
        ;;
esac
for ((j = 1; j <= $numOfQueryVariations; j++)); do

case "$j" in

    "1")
        qFDIR=$tFDIR/"q100";
        numberOfQueries=100;
        ;;
    "2")
        qFDIR=$tFDIR/"q1000";
        numberOfQueries=1000;
        ;;
    *)
        echo "Ending"
        ;;
esac

#--------------------Processing for each case--------------------------
for ((k = 1; k <= $numOfCases; k++)); do

NUMBEROFNODES=35
SIMNAMEFORMATTED="case-$k"

gFDIR=$qFDIR;
cd $gFDIR
echo "Directory: $gFDIR case-$k : Presse enter to continue"; #read line

# Creating directories
gnuresults=$gFDIR"/gnu-files"
mkdir -p $gnuresults

allgnuresults=$gnuresults"/all"
mkdir -p $allgnuresults

indgnuresults=$gnuresults"/ind"
mkdir -p $indgnuresults

uagnuresults=$gnuresults"/ua"
mkdir -p $uagnuresults

dagnuresults=$gnuresults"/da"
mkdir -p $dagnuresults

aresults=$gFDIR"/all"
mkdir -p $aresults
echo "Created:"$aresults
cleanAllResults="$aresults";

amresults=$aresults"/mean"
mkdir -p $amresults

iresults=$gFDIR"/ind"
mkdir -p $iresults
cleanIndResults="$iresults";

idresults=$iresults"/indi"
mkdir -p $idresults

idsresults=$iresults"/selected"
mkdir -p $idsresults
```

```
imrresults=$iresults"/mean-raw"
mkdir -p $imrresults

imresults=$iresults"/mean"
mkdir -p $imresults

iagmrresults=$iresults"/mean-aggr"
mkdir -p $iagmrresults

uaresults=$gFDIR"/ua"
mkdir -p $uaresults
cleanUaResults="$uaresults";

uamrresults=$uaresults"/mean-raw"
mkdir -p $uamrresults

uamresults=$uaresults"/mean"
mkdir -p $uamresults

uamaresults=$uaresults"/mean-aggr"
mkdir -p $uamaresults

daresults=$gFDIR"/da"
mkdir -p $daresults
cleanDaResults="$daresults";

damresults=$daresults"/mean"
mkdir -p $damresults

damaresults=$daresults"/mean-aggr"
mkdir -p $damaresults

#----------------Processing started----------------

LOGENERGYALL=$SIMNAMEFORMATTED"-energy-all"
LOGPACKETALL=$SIMNAMEFORMATTED"-packet-all"
LOGENERGYIND=$SIMNAMEFORMATTED"-energy-ind"
LOGPACKETIND=$SIMNAMEFORMATTED"-packet-ind"
LOGUA=$SIMNAMEFORMATTED"-uaperformance-processed"
LOGDA=$SIMNAMEFORMATTED"-daperformance"

echo "Separating files w.r.t. time: Presse enter to continue";

awk -v "leall=$LOGENERGYALL" -F "    " '{close(f);f=$1}{print > "all/"f"-" leall ".log"}'
    $LOGENERGYALL'.dat'
awk -v "lpall=$LOGPACKETALL" -F "    " '{close(f);f=$1}{print > "all/"f"-" lpall ".log"}'
    $LOGPACKETALL'.dat'
awk -v "leind=$LOGENERGYIND" -F "    " '{close(f);f=$1}{print > "ind/"f"-" leind ".log"}'
    $LOGENERGYIND'.dat'
awk -v "lpind=$LOGPACKETIND" -F "    " '{close(f);f=$1}{print > "ind/"f"-" lpind ".log"}'
    $LOGPACKETIND'.dat'

#echo "Have you selected individual files?: Presse enter to continue"; read line
cp -f $iresults"/$totalSimTime-"*".log"  $idsresults

awk -v "leind=$LOGENERGYIND" -F "    " '{close(f);f=$2;t=$1}{print > "ind/indi/" t "-" leind "-
    "f ".log"}' 'ind/selected/'*'-'$LOGENERGYIND'.log'
awk -v "lpind=$LOGPACKETIND" -F "    " '{close(f);f=$2;t=$1}{print > "ind/indi/" t "-" lpind "-
    "f  ".log"}' 'ind/selected/'*'-'$LOGPACKETIND'.log'

awk -v "logua=$LOGUA" -F "   " '{close(f);f=$1}{print > "ua/" logua "-" f ".log"}' $LOGUA'.log'
awk -v "logda=$LOGDA" -F "   " '{close(f);f=$1}{print > "da/" logda "-" f ".log"}' $LOGDA'.log'

#----------------Aggregated energy and packet statistics----------------
FILES=$aresults/*"case-$k"*
for fileName in $FILES
do

file=$fileName
s=$fileName
filec=${s##*/}
file=${filec%.log}

echo "Going to process:"$file

awk -v "simnamef=$SIMNAMEFORMATTED -v "filetoprocess=$file" -v "isgrouping=$isgrouping" '{

cVal=$2;

arr[NR]= cVal;
```

```
186           if(NR==1) {min=max=cVal}
              if(cVal>max)   {max=cVal}
188           if(cVal<min)   {min=cVal}
              tot+=cVal;
190
    asort(arr)
192           mid= (NR/2)+0.5;
          if((mid%1)!=0) {
194           mid= int(mid);
              med=(arr[mid]+arr[mid+1])/2
196       }else{
                med=arr[mid]
198       }

200   for(i=1; i<=NF; i++){
          allnodesvalues[i]= ($i); #populate the array data
202       sum[i]=sum[i] + ($i);

204       }
      for(i=1; i<=NF; i++){
206       mean[i]=sum[i]/(NR);
          }
208   for(i=1; i<=NF; i++){
          sumsq[i]=sumsq[i] + (($i)-mean[i])^2 ;
210       }
      } END {
212
       print simnamef"\t"mean[2]"\t"sqrt(sumsq[2]/(NR))"\t"min"\t"max"\t"med   "\t"mean[3]"\t"mean[4]
            "\t"mean[5]"\t"mean[6]"\t"mean[7]"\t"NR > "all/mean/"filetoprocess"-mean-stdev.log";
214
       }' 'all/'${file}'.log'
216

218   #*-all-energy-mean.log: [1]Time: [2,3]Total-E-mean :stdev [4,5]E-CPU :stdev [6,7]E-LPM :stdev
            [8,9]E-Listen :stdev [10,11]E-Transmit :stdev
      #*-all-packet-mean.logTime: [2,3]Total-Packets-mean :stdev [4,5]UPD :stdev [6,7]GRP :stdev
220
      done #all
222
      #----------------Individual node energy and packet statistics----------------
224   FILES=$idresults/*"$totalSimTime"*"case-$k"*
      for fileName in $FILES
226   do

228   file=$fileName
      s=$fileName
230   filec=${s##*/}
      file=${filec%.log}
232
      echo "Going to process:"$file
234   echo "Presse enter to continue";
      thisFILENAME=$file
236
      awk -v "simnamef"=$SIMNAMEFORMATTED -v "filetoprocess=$thisFILENAME" -v "isgrouping=
           $isgrouping"  '{
238
      cVal=$3;
240
      arr[NR]= cVal;
242           if(NR==1) {min=max=cVal}
              if(cVal>max)   {max=cVal}
244           if(cVal<min)   {min=cVal}
              tot+=cVal;
246
      asort(arr)
248           mid= (NR/2)+0.5;
          if((mid%1)!=0) {
250           mid= int(mid);
              med=(arr[mid]+arr[mid+1])/2
252       }else{
                med=arr[mid]
254       }

256   for(i=1; i<=NF; i++){
          sum[i]=sum[i] + ($i);
258       }
      for(i=1; i<=NF; i++){
260       mean[i]=sum[i]/(NR);
          }
262   for(i=1; i<=NF; i++){
```

```
    sumsq[i]=sumsq[i] + ($i-mean[i])^2 ;
    }
} END {

 print mean[1]"\t"mean[2]"\t"mean[3]"\t"sqrt(sumsq[3]/(NR))"\t"min"\t"max"\t"med "\t"mean[4]"\t"mean[5]"\t"mean[6]"\t"mean[7]"\t"mean[1] > "ind/mean-raw/"filetoprocess"-mean-stdev.log";

}' 'ind/indi/'${thisFILENAME}'.log'

done #ind

# Combining files
caseEnergyFileName="$totalSimTime-case-$k-ind-energy";
casePacketFileName="$totalSimTime-case-$k-ind-packet";
paste -s "ind/mean-raw/"*"$totalSimTime"*"case-$k"*"energy"* > "ind/mean/$caseEnergyFileName.log"
paste -s "ind/mean-raw/"*"$totalSimTime"*"case-$k"*"packet"* > "ind/mean/$casePacketFileName.log"

sort -r -n -k3 "ind/mean/$caseEnergyFileName.log" -o "ind/mean/$caseEnergyFileName.log"

#----------------UA statistics----------------
awk -v "simnamef"=$SIMNAMEFORMATTED -v "filetoprocess=$newUAFileName" '{
if(NR<6){
 print simnamef"\t"$2"\t"$3"\t"$4 > "gnu-files/ind/" simnamef"-busiest-top-5.log";
}
}' "ind/mean/$caseEnergyFileName.log"

#cat "$qFDIR/gnu-files/ind/all-cases-busiest-top-5.log"
#'echo "DEBUG: sorted file $qFDIR/gnu-files/ind/all-cases-busiest-top-5.log"; read line

farray=("$caseEnergyFileName" "$casePacketFileName")

for fname in ${!farray[*]}
do
    printf "   %s\n" "${farray[$fname]}"

awk -v "simnamef"=$SIMNAMEFORMATTED -v "filetoprocess=${farray[$fname]}" '{

cVal=$3;

arr[NR]= cVal;
         if(NR==1) {min=max=cVal}
         if(cVal>max)  {max=cVal}
         if(cVal<min)  {min=cVal}
         tot+=cVal;

asort(arr)
         mid= (NR/2)+0.5;
    if((mid%1)!=0) {
         mid= int(mid);
         med=(arr[mid]+arr[mid+1])/2
    }else{
           med=arr[mid]
    }

for(i=1; i<=NF; i++){
    sum[i]=sum[i] + ($i);
    }
for(i=1; i<=NF; i++){
    mean[i]=sum[i]/(NR);
    }
for(i=1; i<=NF; i++){
    sumsq[i]=sumsq[i] + ($i-mean[i])^2 ;
    }
} END {

 print simnamef"\t"mean[1]"\t"mean[3]"\t"sqrt(sumsq[3]/(NR))"\t"min"\t"max"\t"med "\t"mean[3]"\t"mean[4]"\t"mean[5]"\t"mean[6]"\t"mean[7]"\t"NR  > "ind/mean-aggr/"filetoprocess".log";}' "ind/mean/${farray[$fname]}.log"
done
#echo "DEBUG: Presse enter to continue"; read line

awk -v "simnamef"=$SIMNAMEFORMATTED -v "gnuresults"=$gnuresults -v "numberOfQueries"=$numberOfQueries '{
 if($3==0) {cVal+=1}
} END {
if (length(cVal) == 0){cVal=0}
```

```
 print simnamef"\t"cVal"\t"cVal/(NR/numberOfQueries)"\t"numberOfQueries"\t"NR > gnuresults"/ua
      /"simnamef"-cache-hits.log";
336
}' ${qFDIR}'/'${SIMNAMEFORMATTED}'-uaperformance-processed.log'
338

340 FILES=$uaresults/*"case-$k"*
for fileName in $FILES
342 do

344 file=$fileName
s=$fileName
346 filec=${s##*/}
file=${filec%.log}
348
thisFILENAME=$file
350 echo "Going to process:"$thisFILENAME

352 awk -v "simnamef"=$SIMNAMEFORMATTED -v "filetoprocess=$thisFILENAME" -v "isgrouping=
     $isgrouping" '{

354 cVal=$2;

356 arr[NR]= cVal;
          if(NR==1) {min=max=cVal}
358       if(cVal>max)  {max=cVal}
          if(cVal<min)  {min=cVal}
360       tot+=cVal;

362 asort(arr)
          mid= (NR/2)+0.5;
364     if((mid%1)!=0) {
          mid= int(mid);
366       med=(arr[mid]+arr[mid+1])/2
      }else{
368           med=arr[mid]
      }
370
cVal2=$3;
372
arr2[NR]= cVal2;
374       if(NR==1) {min2=max2=cVal2}
          if(cVal2>max2)  {max2=cVal2}
376       if(cVal2<min2)  {min2=cVal2}
          tot2+=cVal2;
378
asort(arr2)
380       mid2= (NR/2)+0.5;
      if((mid2%1)!=0) {
382       mid2= int(mid2);
          med2=(arr2[mid2]+arr2[mid2+1])/2
384   }else{
            med2=arr2[mid2]
386   }

388 for(i=1; i<=NF; i++){
    sum[i]=sum[i] + ($i);
390     }
    for(i=1; i<=NF; i++){
392     mean[i]=sum[i]/(NR);
        }
394 for(i=1; i<=NF; i++){
    sumsq[i]=sumsq[i] + ($i-mean[i])^2 ;
396     }
} END {
398
 print mean[1]"\t"mean[2]"\t"sqrt(sumsq[2]/(NR))"\t"min"\t"max"\t"med "\t"mean[3]"\t"sqrt(
      sumsq[3]/(NR))"\t"min2"\t"max2"\t"med2"\t"mean[4]"\t"mean[5]"\t"$6 > "ua/mean-raw/"
       filetoprocess"mean-stdev.log";
400
}' 'ua/'${thisFILENAME}'.log'
402
done #End of INDividual files mean and stddev calculations"ind/mean/$newIndsFileName.log"
404
newUAFileName="case-$k-uaperformance-processed";
406 paste -s "ua/mean-raw/"*"case-$k"* > "ua/mean/$newUAFileName.log"

408 awk -v "simnamef"=$SIMNAMEFORMATTED -v "filetoprocess=$newUAFileName" '{

410 cVal=$2;
```

```
412 arr[NR]= cVal;
            if(NR==1) {min=max=cVal}
414         if(cVal>max)  {max=cVal}
            if(cVal<min)  {min=cVal}
416         tot+=cVal;

418 asort(arr)
            mid= (NR/2)+0.5;
420     if((mid%1)!=0) {
            mid= int(mid);
422         med=(arr[mid]+arr[mid+1])/2
    }else{
424           med=arr[mid]
    }
426
cVal2=$7;
428 if(cVal2==0){
cachehit = cachehit+1;
430 print "cache HIT";
}
432 arr2[NR]= cVal2;
            if(NR==1) {min2=max2=cVal2}
434         if(cVal2>max2)  {max2=cVal2}
            if(cVal2<min2)  {min2=cVal2}
436         tot2+=cVal2;

438 asort(arr2)
            mid2= (NR/2)+0.5;
440     if((mid2%1)!=0) {
            mid2= int(mid2);
442         med2=(arr2[mid2]+arr2[mid2+1])/2
    }else{
444           med2=arr2[mid2]
    }
446
for(i=1; i<=NF; i++){
448     sum[i]=sum[i] + ($i);
    }
450 for(i=1; i<=NF; i++){
    mean[i]=sum[i]/(NR);
452 }
for(i=1; i<=NF; i++){
454     sumsq[i]=sumsq[i] + ($i-mean[i])^2 ;
    }
456 } END {
if(cachehit==""){
458 cachehit=0;
}
460  print simnamef"\t"mean[2]"\t"sqrt(sumsq[2]/(NR))"\t"min"\t"max"\t"med "\t"mean[7]"\t"sqrt(
      sumsq[7]/(NR))"\t"min2"\t"max2"\t"med2"\t"cachehit > "ua/mean-aggr/"filetoprocess".log"
      ;}' "ua/mean/$newUAFileName.log"

462 #----------------DA statistics----------------
FILES=$daresults/*"case-$k"*
464 for fileName in $FILES
do
466
file=$fileName
468 s=$fileName
filec=${s##*/}
470 file=${filec%.log}

472 echo "Going to process:"$file

474 awk -v "simnamef"=$SIMNAMEFORMATTED -v "filetoprocess=$file" -v "isgrouping=$isgrouping" '{

476 cVal=$2;

478 arr[NR]= cVal;
            if(NR==1) {min=max=cVal}
480         if(cVal>max)  {max=cVal}
            if(cVal<min)  {min=cVal}
482         tot+=cVal;

484 asort(arr)
            mid= (NR/2)+0.5;
486     if((mid%1)!=0) {
            mid= int(mid);
488         med=(arr[mid]+arr[mid+1])/2
```

```
    }else{
490             med=arr[mid]
    }
492
for(i=1; i<=NF; i++){
494     allnodesvalues[i]= ($i); #populate the array data
    sum[i]=sum[i] + ($i);
496
    }
498 for(i=1; i<=NF; i++){
    mean[i]=sum[i]/(NR);
500     }
    for(i=1; i<=NF; i++){
502     sumsq[i]=sumsq[i] + (($i)-mean[i])^2 ;
    }
504 } END {

506  print simnamef"\t"mean[1]"\t"mean[2]"\t"sqrt(sumsq[2]/(NR))"\t"min"\t"max"\t"med  "\t"mean[3]
       "\t"mean[4]"\t"mean[5]"\t"mean[6]"\t"mean[7]"\t"NR > "da/mean/"filetoprocess"mean-stdev.
       log";
 }' 'da/'${file}'.log'
508 done #all da

510 dafile="case-$k-daperformance"

512 newDAFileName="case-$k-da-performance";
    paste -s "da/mean/"*"case-$k"* > "da/mean-aggr/case-$k-da-performance.log"
514
    paste -s "all/mean/"*"case-$k"*"-energy"*".log" > $allgnuresults"/all-case-$k-energy-mean.log"
516 paste -s "all/mean/"*"case-$k"*"-packet"*".log" > $allgnuresults"/all-case-$k-packet-mean.log"
    sort -n -k2 "ind/mean/$totalSimTime-case-$k-ind-energy.log" > $indgnuresults"/$totalSimTime-
        ind-case-$k-energy-mean.log"
518 sort -n -k2 "ind/mean/$totalSimTime-case-$k-ind-packet.log" > $indgnuresults"/$totalSimTime-
        ind-case-$k-packet-mean.log"
    paste -s "da/mean/case-$k"*""*".log" > $dagnuresults"/case-$k-daperformance-mean.log"
520 sort -n +0 -1 "ua/mean/case-$k"*""*".log" > $uagnuresults"/case-$k-uaperformance-mean.log"

522 done #End of one Unique Simulation - Loop for all simulations

524 #------------Combining statistics files for graph automation----------------
    #For energy and packet details of a all cases
526 paste -s "all/mean/$totalSimTime-case"*"-energy"*".log" > $allgnuresults"/$totalSimTime-all-
        cases-all-energy-mean.log"
    paste -s "all/mean/$totalSimTime-case"*"-packet"*".log" > $allgnuresults"/$totalSimTime-all-
        cases-all-packet-mean.log"
528 sort -n +1 -1  "$iagmrresults/$totalSimTime-case"*"-energy.log" > $indgnuresults"/
        $totalSimTime-all-cases-ind-avg-node-energy-mean.log"
    sort -n +1 -1  "$iagmrresults/$totalSimTime-case"*"-packet.log" > $indgnuresults"/
        $totalSimTime-all-cases-ind-avg-node-packet-mean.log"
530
    paste -s "$iagmrresults/case"*"-energy"*".log" > $indgnuresults"/all-cases-ind-avg-node-energy
        -mean.log"
532 paste -s "$iagmrresults/case"*"-packet"*".log" > $indgnuresults"/all-cases-ind-avg-node-packet
        -mean.log"

534 sort -n +1 -1 $dagnuresults"/case-"*".log" > $dagnuresults"/all-cases-daperformance-aggr-mean.
        log"
    paste -s $uamaresults"/case-"*".log" > $uagnuresults"/all-cases-uaperformance-aggr-mean.log"
536 paste -s $uagnuresults"/case-"*"-cache-hits.log" > $uagnuresults"/all-cases-cache-hits-mean.
        log"

538 awk -v "dir=$uagnuresults" -F " " '{close(f);f=$14}{print > dir "/" f "-case-1-uaperformance-
        mean.log"}' $uagnuresults'/case-1-uaperformance-mean.log'
    awk -v "dir=$uagnuresults" -F " " '{close(f);f=$14}{print > dir "/" f "-case-2-uaperformance-
        mean.log"}' $uagnuresults'/case-2-uaperformance-mean.log'
540 awk -v "dir=$uagnuresults" -F " " '{close(f);f=$14}{print > dir "/" f "-case-3-uaperformance-
        mean.log"}' $uagnuresults'/case-3-uaperformance-mean.log'
    awk -v "dir=$uagnuresults" -F " " '{close(f);f=$14}{print > dir "/" f "-case-4-uaperformance-
        mean.log"}' $uagnuresults'/case-4-uaperformance-mean.log'
542 awk -v "dir=$uagnuresults" -F " " '{close(f);f=$14}{print > dir "/" f "-case-5-uaperformance-
        mean.log"}' $uagnuresults'/case-5-uaperformance-mean.log'
    awk -v "dir=$uagnuresults" -F " " '{close(f);f=$14}{print > dir "/" f "-case-6-uaperformance-
        mean.log"}' $uagnuresults'/case-6-uaperformance-mean.log'
544
    #Delete all unneeded files
546 rm -rf $cleanAllResults;
    rm -rf $cleanIndResults;
548 rm -rf $cleanUaResults;
    rm -rf $cleanDaResults; #read line
550 done
```

```
552
done
```

C.2 GNUPLOT graphs generation

The GNUPLOT scripts are used to automate the graph generation from the processed statistics data. Listings C.2 and C.3 show two example scripts used to generate graphs for one set of experiments.

Listing C.2: Gnuplot Code to generate histogram graphs

```
#Configurations
set term post eps size 6,3 enhanced color
set style histogram errorbars linewidth 1
set grid y
set style histogram errorbars
set style data histograms
set style fill solid 0.3 # Make the bars semi-transparent so that the errorbars are easier to see.
set key width -1 outside below center vertical maxrows 3

# Attributes
top1q100="Topo1 - 100"
top2q100="Topo2 - 100"
top3q100="Topo3 - 100"
top1q1000="Topo1 - 1000"
top2q1000="Topo2 - 1000"
top3q1000="Topo3 - 1000"
dtop1q100="Topology 1 - 100 queries"
dtop2q100="Topology 2 - 100 queries"
dtop3q100="Topology 3 - 100 queries"
dtop1q1000="Topology 1 - 1000 queries"
dtop2q1000="Topology 2 - 1000 queries"
dtop3q1000="Topology 3 - 1000 queries"

#-------------------Energy-------------------------------------
set ylabel "Energy Consumption in Joules (J)"
set yrange [0:*]
set key outside below center horizontal
set xlabel "Scenarios"
set title ""
set output "all-tops-8400-energy-all-mean-cases-all.eps"
plot 'top1/q100/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($2/1000):($3/1000):xtic(1)  title top1q100 fs pattern 2 lc rgb "forest-green", \
    'top2/q100/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($2/1000):($3/1000):xtic(1)  title top2q100 fs pattern 2 lc rgb "blue", \
    'top3/q100/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($2/1000):($3/1000):xtic(1)  title top3q100 fs pattern 2 lc rgb "red", \
    'top1/q1000/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($2/1000):($3/1000):xtic(1)  title top1q1000 fs pattern 3 lc rgb "forest-green", \
    'top2/q1000/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($2/1000):($3/1000):xtic(1)  title top2q1000 fs pattern 3 lc rgb "blue", \
    'top3/q1000/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($2/1000):($3/1000):xtic(1)  title top3q1000 fs pattern 3 lc rgb "red";

set output "all-tops-8400-energy-ind-mean-cases-all.eps"
plot    'top1/q100/gnu-files/ind/8400-all-cases-ind-avg-node-energy-mean.log' using 3:4:xtic(1)  title top1q100 fs pattern 2 lc rgb "forest-green", \
        'top2/q100/gnu-files/ind/8400-all-cases-ind-avg-node-energy-mean.log' using 3:4:xtic(1)  title top2q100 fs pattern 2 lc rgb "blue", \
    'top3/q100/gnu-files/ind/8400-all-cases-ind-avg-node-energy-mean.log' using 3:4:xtic(1)  title top3q100 fs pattern 2 lc rgb "red", \
    'top1/q1000/gnu-files/ind/8400-all-cases-ind-avg-node-energy-mean.log' using 3:4:xtic(1)  title top1q1000 fs pattern 3 lc rgb "forest-green",\
    'top2/q1000/gnu-files/ind/8400-all-cases-ind-avg-node-energy-mean.log' using 3:4:xtic(1)  title top2q1000 fs pattern 3 lc rgb "blue",\
    'top3/q1000/gnu-files/ind/8400-all-cases-ind-avg-node-energy-mean.log' using 3:4:xtic(1)  title top3q1000 fs pattern 3 lc rgb "red";

#-------------------Service Invocation-------------------------------------
set key width -1 outside below center vertical maxrows 3
```

```
set ylabel "Service Invocation Delay in milli seconds"
set xlabel "Scenarios"
set title ""
set output "all-tops-ua-si-delay-mean-all-cases.eps"
plot     'top1/q100/gnu-files/ua/all-cases-uaperformance-aggr-mean.log' using 7:8:xtic(1)
     title dtop1q100 fs pattern 2 lc rgb "forest-green", \
     'top2/q100/gnu-files/ua/all-cases-uaperformance-aggr-mean.log' using 7:8:xtic(1)  title
          dtop2q100 fs pattern 2 lc rgb "blue", \
     'top3/q100/gnu-files/ua/all-cases-uaperformance-aggr-mean.log' using 7:8:xtic(1)  title
          dtop3q100 fs pattern 2 lc rgb "red", \
     'top1/q1000/gnu-files/ua/all-cases-uaperformance-aggr-mean.log' using 7:8:xtic(1)  title
          dtop1q1000 fs pattern 3 lc rgb "forest-green", \
     'top2/q1000/gnu-files/ua/all-cases-uaperformance-aggr-mean.log' using 7:8:xtic(1)  title
          dtop2q1000 fs pattern 3 lc rgb "blue", \
     'top3/q1000/gnu-files/ua/all-cases-uaperformance-aggr-mean.log' using 7:8:xtic(1)  title
          dtop3q1000 fs pattern 3 lc rgb "red";

#-------------------Packets-------------------------------------
set ylabel "Total number of packets"
set xlabel "Scenarios"
set title ""
set output "all-tops-da-8400-mean-all-cases.eps"
plot     "< awk '{if ($2==8400) {print $0}}' top1/q100/gnu-files/da/all-cases-daperformance-
     aggr-mean.log" using 3:4:xtic(1)  title dtop1q100 fs pattern 2 lc rgb "forest-green", \
     "< awk '{if ($2==8400) {print $0}}' top2/q100/gnu-files/da/all-cases-daperformance-aggr-
          mean.log" using 3:4:xtic(1)  title dtop2q100 fs pattern 2 lc rgb "blue", \
     "< awk '{if ($2==8400) {print $0}}' top3/q100/gnu-files/da/all-cases-daperformance-aggr-
          mean.log" using 3:4:xtic(1)  title dtop3q100 fs pattern 2 lc rgb "red", \
     "< awk '{if ($2==8400) {print $0}}' top1/q1000/gnu-files/da/all-cases-daperformance-aggr-
          mean.log" using 3:4:xtic(1)  title dtop1q1000 fs pattern 3 lc rgb "forest-green", \
     "< awk '{if ($2==8400) {print $0}}' top2/q1000/gnu-files/da/all-cases-daperformance-aggr-
          mean.log" using 3:4:xtic(1)  title dtop2q1000 fs pattern 3 lc rgb "blue", \
     "< awk '{if ($2==8400) {print $0}}' top3/q1000/gnu-files/da/all-cases-daperformance-aggr-
          mean.log" using 3:4:xtic(1)  title dtop3q1000 fs pattern 3 lc rgb "red";

set output "all-tops-8400-packet-ind-avg-node-mean-cases-all.eps"
plot     'top1/q100/gnu-files/ind/8400-all-cases-ind-avg-node-packet-mean.log' using 3:4:xtic
     (1)  title top1q100 fs pattern 2 lc rgb "forest-green", \
     'top2/q100/gnu-files/ind/8400-all-cases-ind-avg-node-packet-mean.log' using 3:4:xtic(1)
          title top2q100 fs pattern 2 lc rgb "blue", \
     'top3/q100/gnu-files/ind/8400-all-cases-ind-avg-node-packet-mean.log' using 3:4:xtic(1)
          title top3q100 fs pattern 2 lc rgb "red", \
     'top1/q1000/gnu-files/ind/8400-all-cases-ind-avg-node-packet-mean.log' using 3:4:xtic(1)
          title top1q1000 fs pattern 3 lc rgb "forest-green", \
     'top2/q1000/gnu-files/ind/8400-all-cases-ind-avg-node-packet-mean.log' using 3:4:xtic(1)
          title top2q1000 fs pattern 3 lc rgb "blue", \
     'top3/q1000/gnu-files/ind/8400-all-cases-ind-avg-node-packet-mean.log' using 3:4:xtic(1)
          title top3q1000 fs pattern 3 lc rgb "red";

set output "all-tops-8400-packet-all-mean-cases-all.eps"
plot     'top1/q100/gnu-files/all/8400-all-cases-all-packet-mean.log' using 2:3:xtic(1)  title
     dtop1q100 fs pattern 2 lc rgb "forest-green", \
     'top2/q100/gnu-files/all/8400-all-cases-all-packet-mean.log' using 2:3:xtic(1)  title
          dtop2q100 fs pattern 2 lc rgb "blue", \
     'top3/q100/gnu-files/all/8400-all-cases-all-packet-mean.log' using 2:3:xtic(1)  title
          dtop3q100 fs pattern 2 lc rgb "red", \
     'top1/q1000/gnu-files/all/8400-all-cases-all-packet-mean.log' using 2:3:xtic(1)  title
          dtop1q1000 fs pattern 3 lc rgb "forest-green", \
     'top2/q1000/gnu-files/all/8400-all-cases-all-packet-mean.log' using 2:3:xtic(1)  title
          dtop2q1000 fs pattern 3 lc rgb "blue", \
     'top3/q1000/gnu-files/all/8400-all-cases-all-packet-mean.log' using 2:3:xtic(1)  title
          dtop3q1000 fs pattern 3 lc rgb "red";

#-------------------Cache hits-------------------------------------
set style histogram

set ylabel "Number of Cache hits"
set xlabel ""
set title ""
set output "all-tops-cache-hits-all-cases-all-mean.eps"
plot     'top1/q100/gnu-files/ua/all-cases-cache-hits-mean.log' using 3:xtic(1)  title
     dtop1q100 fs pattern 2 lc rgb "forest-green", \
     'top2/q100/gnu-files/ua/all-cases-cache-hits-mean.log' using 3:xtic(1)  title dtop2q100 fs
          pattern 2 lc rgb "blue", \
     'top3/q100/gnu-files/ua/all-cases-cache-hits-mean.log' using 3:xtic(1)  title dtop3q100 fs
          pattern 2 lc rgb "red", \
     'top1/q1000/gnu-files/ua/all-cases-cache-hits-mean.log' using 3:xtic(1)  title dtop1q1000
          fs pattern 3 lc rgb "forest-green", \
     'top2/q1000/gnu-files/ua/all-cases-cache-hits-mean.log' using 3:xtic(1)  title dtop2q1000
          fs pattern 3 lc rgb "blue", \
```

```
    'top3/q1000/gnu-files/ua/all-cases-cache-hits-mean.log' using 3:xtic(1)  title dtop3q1000
        fs pattern 3 lc rgb "red";

#-------------------Further details of energy and packets---------------------------------
set style data histogram
set style histogram rowstack gap 1
set boxwidth 0.75 absolute
set style fill solid 1.00 border -1
set ylabel "Energy Consumption in Joules (J)"
set yrange [0:*]
set key outside below center horizontal
set xtics nomirror rotate by -45 scale 1
set xlabel " " offset 0,-2

set title ""
set output "all-tops-8400-energy-details-all-cases-all-mean.eps"
plot for [i=1:20] newhistogram top1q100 lt 1, \
'top1/q100/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($9/1000) fs pattern 3 t "
    Listen + Receive" , \
                              '' using ($10/1000) fs pattern 2 t "Transmit", \
                              '' using ($7/1000) fs pattern 1 t "CPU",  \
                              '' using ($8/1000):xtic(1)  fs pattern 4 t "LPM", \
     newhistogram top2q100 lt 1, \
'top2/q100/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($9/1000) fs pattern 3
    notitle , \
                              '' using ($10/1000) fs pattern 2 notitle, \
                              '' using ($7/1000) fs pattern 1 notitle,  \
                              '' using ($8/1000):xtic(1)  fs pattern 4 notitle, \
     newhistogram top3q100 lt 1, \
'top3/q100/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($9/1000) fs pattern 3
    notitle , \
                              '' using ($10/1000) fs pattern 2 notitle, \
                              '' using ($7/1000) fs pattern 1 notitle,  \
                              '' using ($8/1000):xtic(1)  fs pattern 4 notitle, \
     newhistogram top1q1000 lt 1, \
'top1/q1000/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($9/1000) fs pattern 3
    notitle , \
                              '' using ($10/1000) fs pattern 2 notitle, \
                              '' using ($7/1000) fs pattern 1 notitle,  \
                              '' using ($8/1000):xtic(1)  fs pattern 4 notitle, \
     newhistogram top2q1000 lt 1, \
'top2/q1000/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($9/1000) fs pattern 3
    notitle , \
                              '' using ($10/1000) fs pattern 2 notitle, \
                              '' using ($7/1000) fs pattern 1 notitle,  \
                              '' using ($8/1000):xtic(1)  fs pattern 4 notitle, \
     newhistogram top3q1000 lt 1, \
'top3/q1000/gnu-files/all/8400-all-cases-all-energy-mean.log' using ($9/1000) fs pattern 3
    notitle , \
                              '' using ($10/1000) fs pattern 2 notitle, \
                              '' using ($7/1000) fs pattern 1 notitle,  \
                              '' using ($8/1000):xtic(1)  fs pattern 4 notitle;

set key autotitle columnheader
set key outside below center horizontal
set ylabel "Number of Packets"
set title ""
set output "all-tops-8400-packet-details-all-cases-all-mean.eps"
plot newhistogram top1q100 lt 1, \
'top1/q100/gnu-files/all/8400-all-cases-all-packet-mean.log' using 7 fs pattern 3 t "Updates"
    , \
                              '' using 8:xtic(1)  fs pattern 1 t "Grouping", \
     newhistogram top2q100 lt 1, \
'top2/q100/gnu-files/all/8400-all-cases-all-packet-mean.log' using 7 fs pattern 3 notitle , \
                              '' using 8:xtic(1)  fs pattern 1 notitle, \
     newhistogram top3q100 lt 1, \
'top3/q100/gnu-files/all/8400-all-cases-all-packet-mean.log' using 7 fs pattern 3 notitle , \
                              '' using 8:xtic(1)  fs pattern 1 notitle, \
     newhistogram top1q1000 lt 1, \
'top1/q1000/gnu-files/all/8400-all-cases-all-packet-mean.log' using 7 fs pattern 3 notitle , \
                              '' using 8:xtic(1)  fs pattern 1 notitle, \
     newhistogram top2q1000 lt 1, \
'top2/q1000/gnu-files/all/8400-all-cases-all-packet-mean.log' using 7 fs pattern 3 notitle , \
                              '' using 8:xtic(1)  fs pattern 1 notitle, \
     newhistogram top3q1000 lt 1, \
'top3/q1000/gnu-files/all/8400-all-cases-all-packet-mean.log' using 7 fs pattern 3 notitle , \
                              '' using 8:xtic(1)  fs pattern 1 notitle;

#-------------------Energy: Top five-----------------------------------
set style histogram clustered gap 1 title offset 2,0.25
```

```
set style data histograms
set style histogram errorbars
set style fill solid 0.7
set boxwidth 2
set xlabel " " offset 0,-2
set ytics
set auto y
set ylabel "Energy Consumption (mJ)"
set key outside below center horizontal
set xtics ("100 queries" 9, "1000 queries" 29)
set output "all-tops-all-cases-ind-top-5.eps"
plot newhistogram "Case-1" lt 1, \
     'top1/q100/gnu-files/ind/case-1-busiest-top-5.log' u 3:4 t "Topology #1", \
     'top2/q100/gnu-files/ind/case-1-busiest-top-5.log' u 3:4 t "Topology #2", \
     'top3/q100/gnu-files/ind/case-1-busiest-top-5.log' u 3:4 t "Topology #3", \
     newhistogram "Case-2" lt 1, \
     'top1/q100/gnu-files/ind/case-2-busiest-top-5.log' u 3:4 notitle, \
     'top2/q100/gnu-files/ind/case-2-busiest-top-5.log' u 3:4 notitle, \
     'top3/q100/gnu-files/ind/case-2-busiest-top-5.log' u 3:4 notitle, \
     newhistogram "Case-3" lt 1, \
     'top1/q100/gnu-files/ind/case-3-busiest-top-5.log' u 3:4 notitle, \
     'top2/q100/gnu-files/ind/case-3-busiest-top-5.log' u 3:4 notitle, \
     'top3/q100/gnu-files/ind/case-3-busiest-top-5.log' u 3:4 notitle, \
     newhistogram "Case-4" lt 1, \
     'top1/q100/gnu-files/ind/case-4-busiest-top-5.log' u 3:4 notitle, \
     'top2/q100/gnu-files/ind/case-4-busiest-top-5.log' u 3:4 notitle, \
     'top3/q100/gnu-files/ind/case-4-busiest-top-5.log' u 3:4 notitle, \
     newhistogram "Case-1" lt 7, \
     'top1/q1000/gnu-files/ind/case-1-busiest-top-5.log' u 3:4 notitle, \
     'top2/q1000/gnu-files/ind/case-1-busiest-top-5.log' u 3:4 notitle, \
     'top3/q1000/gnu-files/ind/case-1-busiest-top-5.log' u 3:4 notitle, \
     newhistogram "Case-2" lt 7, \
     'top1/q1000/gnu-files/ind/case-2-busiest-top-5.log' u 3:4 notitle, \
     'top2/q1000/gnu-files/ind/case-2-busiest-top-5.log' u 3:4 notitle, \
     'top3/q1000/gnu-files/ind/case-2-busiest-top-5.log' u 3:4 notitle, \
    newhistogram "Case-3" lt 7, \
     'top1/q1000/gnu-files/ind/case-3-busiest-top-5.log' u 3:4 notitle, \
     'top2/q1000/gnu-files/ind/case-3-busiest-top-5.log' u 3:4 notitle, \
     'top3/q1000/gnu-files/ind/case-3-busiest-top-5.log' u 3:4 notitle,\
    newhistogram "Case-4" lt 7, \
     'top1/q1000/gnu-files/ind/case-4-busiest-top-5.log' u 3:4 notitle,\
     'top2/q1000/gnu-files/ind/case-4-busiest-top-5.log' u 3:4 notitle,\
     'top3/q1000/gnu-files/ind/case-4-busiest-top-5.log' u 3:4 notitle;
```

Listing C.3: Gnuplot Code to generate Linepoint graphs

```
#Configurations
set term post eps size 6,3 enhanced color
set grid y

set style line 1 lt 1 pt 1 lc 1 lw 2
set style line 2 lt 1 pt 1 lc rgb "forest-green" lw 2
set style line 3 lt 1 pt 1 lc 3 lw 2
set style line 4 lt 1 pt 1 lc rgb "dark-magenta" lw 2
set style line 5 lt 2 pt 6 lc 1 lw 2
set style line 6 lt 2 pt 6 lc rgb "forest-green" lw 2
set style line 7 lt 2 pt 6 lc 3 lw 2
set style line 8 lt 2 pt 6 lc rgb "dark-magenta" lw 2
set style line 9 lt 3 pt 3 lc 1 lw 2
set style line 10 lt 3 pt 3 lc rgb "forest-green" lw 2
set style line 11 lt 3 pt 3 lc 3 lw 2
set style line 12 lt 3 pt 3 lc rgb "dark-magenta" lw 2

set style line 16 lc rgb '#808080' lt 0 lw 1
set grid back ls 16

#Attributes
top1c1="Topology 1 - case-1"
top1c2="Topology 1 - case-2"
top1c3="Topology 1 - case-3"
top1c4="Topology 1 - case-4"
top2c1="Topology 2 - case-1"
top2c2="Topology 2 - case-2"
top2c3="Topology 2 - case-3"
top2c4="Topology 2 - case-4"
top3c1="Topology 3 - case-1"
top3c2="Topology 3 - case-2"
top3c3="Topology 3 - case-3"
top3c4="Topology 3 - case-4"
```

```
35
set key outside below center horizontal
37 set style fill solid 1.00 border 1
set key width -2 outside below center vertical maxrows 3
39
#--------------------Energy--------------------------
41 set ylabel ""
set xlabel "Energy consumption in milli Joules (mJ)"
43
set output "8400-energy-tops-case-all-q1000-ind-cdf.eps"
45 plot "top1/q1000/gnu-files/ind/8400-ind-case-1-energy-mean.log"  using ($3):(1./35.) title
     top1c1 with linespoints ls 1 smooth cumulative, \
   "top2/q1000/gnu-files/ind/8400-ind-case-1-energy-mean.log"  using ($3):(1./35.) title top2c1
        with linespoints ls 5 smooth cumulative, \
47  "top3/q1000/gnu-files/ind/8400-ind-case-1-energy-mean.log"  using ($3):(1./35.) title top3c1
        with linespoints ls 9 smooth cumulative, \
   "top1/q1000/gnu-files/ind/8400-ind-case-2-energy-mean.log" using ($3):(1./35.) title top1c2
        with linespoints ls 2 smooth cumulative, \
49  "top2/q1000/gnu-files/ind/8400-ind-case-2-energy-mean.log" using ($3):(1./35.) title top2c2
        with linespoints ls 6 smooth cumulative, \
   "top3/q1000/gnu-files/ind/8400-ind-case-2-energy-mean.log" using ($3):(1./35.) title top3c2
        with linespoints ls 10  smooth cumulative, \
51  "top1/q1000/gnu-files/ind/8400-ind-case-3-energy-mean.log" using ($3):(1./35.) title top1c3
        with linespoints ls 3 smooth cumulative, \
   "top2/q1000/gnu-files/ind/8400-ind-case-3-energy-mean.log" using ($3):(1./35.) title top2c3
        with linespoints ls 7 smooth cumulative, \
53  "top3/q1000/gnu-files/ind/8400-ind-case-3-energy-mean.log" using ($3):(1./35.) title top3c3
        with linespoints ls 11 smooth cumulative, \
   "top1/q1000/gnu-files/ind/8400-ind-case-4-energy-mean.log" using ($3):(1./35.) title top1c4
        with linespoints ls 4 smooth cumulative,\
55  "top2/q1000/gnu-files/ind/8400-ind-case-4-energy-mean.log" using ($3):(1./35.) title top2c4
        with linespoints ls 8 smooth cumulative,\
   "top3/q1000/gnu-files/ind/8400-ind-case-4-energy-mean.log" using ($3):(1./35.) title top3c4
        with linespoints ls 12 smooth cumulative
57
set output "8400-energy-tops-case-all-q100-ind-cdf.eps"
59 plot "top1/q100/gnu-files/ind/8400-ind-case-1-energy-mean.log"  using ($3):(1./35.) title
     top1c1 with linespoints ls 1 smooth cumulative, \
   "top2/q100/gnu-files/ind/8400-ind-case-1-energy-mean.log"  using ($3):(1./35.) title top2c1
        with linespoints ls 5 smooth cumulative, \
61  "top3/q100/gnu-files/ind/8400-ind-case-1-energy-mean.log"  using ($3):(1./35.) title top3c1
        with linespoints ls 9 smooth cumulative, \
   "top1/q100/gnu-files/ind/8400-ind-case-2-energy-mean.log" using ($3):(1./35.) title top1c2
        with linespoints ls 2 smooth cumulative, \
63  "top2/q100/gnu-files/ind/8400-ind-case-2-energy-mean.log" using ($3):(1./35.) title top2c2
        with linespoints ls 6 smooth cumulative, \
   "top3/q100/gnu-files/ind/8400-ind-case-2-energy-mean.log" using ($3):(1./35.) title top3c2
        with linespoints ls 10  smooth cumulative, \
65  "top1/q100/gnu-files/ind/8400-ind-case-3-energy-mean.log" using ($3):(1./35.) title top1c3
        with linespoints ls 3 smooth cumulative, \
   "top2/q100/gnu-files/ind/8400-ind-case-3-energy-mean.log" using ($3):(1./35.) title top2c3
        with linespoints ls 7 smooth cumulative, \
67  "top3/q100/gnu-files/ind/8400-ind-case-3-energy-mean.log" using ($3):(1./35.) title top3c3
        with linespoints ls 11 smooth cumulative, \
   "top1/q100/gnu-files/ind/8400-ind-case-4-energy-mean.log" using ($3):(1./35.) title top1c4
        with linespoints ls 4 smooth cumulative,\
69  "top2/q100/gnu-files/ind/8400-ind-case-4-energy-mean.log" using ($3):(1./35.) title top2c4
        with linespoints ls 8 smooth cumulative,\
   "top3/q100/gnu-files/ind/8400-ind-case-4-energy-mean.log" using ($3):(1./35.) title top3c4
        with linespoints ls 12 smooth cumulative
71
#--------------------Packets--------------------------
73 set ylabel ""
set xlabel "Number of packets"
75 set title ""

77 set output "8400-packet-tops-case-all-q1000-ind-cdf.eps"
plot "top1/q1000/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title
     top1c1 with linespoints ls 1 smooth cumulative, \
79  "top2/q1000/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title top2c1
        with linespoints ls 5 smooth cumulative, \
   "top3/q1000/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title top3c1
        with linespoints ls 9 smooth cumulative, \
81  "top1/q1000/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top1c2
        with linespoints ls 2 smooth cumulative, \
   "top2/q1000/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top2c2
        with linespoints ls 6 smooth cumulative, \
83  "top3/q1000/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top3c2
        with linespoints ls 10  smooth cumulative, \
```

```
  "top1/q1000/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top1c3
       with linespoints ls 3 smooth cumulative, \
  "top2/q1000/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top2c3
       with linespoints ls 7 smooth cumulative, \
  "top3/q1000/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top3c3
       with linespoints ls 11 smooth cumulative, \
  "top1/q1000/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top1c4
       with linespoints ls 4 smooth cumulative,\
  "top2/q1000/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top2c4
       with linespoints ls 8 smooth cumulative,\
  "top3/q1000/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top3c4
       with linespoints ls 12 smooth cumulative

set output "8400-packet-tops-case-all-q100-ind-cdf.eps"
plot "top1/q100/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title
    top1c1 with linespoints ls 1 smooth cumulative, \
  "top2/q100/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title top2c1
       with linespoints ls 5 smooth cumulative, \
  "top3/q100/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title top3c1
       with linespoints ls 9 smooth cumulative, \
  "top1/q100/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top1c2
       with linespoints ls 2 smooth cumulative, \
  "top2/q100/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top2c2
       with linespoints ls 6 smooth cumulative, \
  "top3/q100/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top3c2
       with linespoints ls 10  smooth cumulative, \
  "top1/q100/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top1c3
       with linespoints ls 3 smooth cumulative, \
  "top2/q100/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top2c3
       with linespoints ls 7 smooth cumulative, \
  "top3/q100/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top3c3
       with linespoints ls 11 smooth cumulative, \
  "top1/q100/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top1c4
       with linespoints ls 4 smooth cumulative,\
  "top2/q100/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top2c4
       with linespoints ls 8 smooth cumulative,\
  "top3/q100/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top3c4
       with linespoints ls 12 smooth cumulative

#-------------------Service Invocation ------------------------
set ylabel ""
set xlabel "Service Invocation (SI) delay in milli seconds"

set output "8400-packet-tops-case-all-q100-ind-cdf.eps"
plot "top1/q100/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title
    top1c1 with linespoints ls 1 smooth cumulative, \
  "top1/q100/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top1c2
       with linespoints ls 2 smooth cumulative, \
  "top1/q100/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top1c3
       with linespoints ls 3 smooth cumulative, \
  "top1/q100/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top1c4
       with linespoints ls 4 smooth cumulative,\
"top2/q100/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title top2c1
    with linespoints ls 5 smooth cumulative, \
  "top2/q100/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top2c2
       with linespoints ls 6 smooth cumulative, \
  "top2/q100/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top2c3
       with linespoints ls 7 smooth cumulative, \
  "top2/q100/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top2c4
       with linespoints ls 8 smooth cumulative,\
"top3/q100/gnu-files/ind/8400-ind-case-1-packet-mean.log"  using ($3):(1./35.) title top3c1
    with linespoints ls 9 smooth cumulative, \
  "top3/q100/gnu-files/ind/8400-ind-case-2-packet-mean.log" using ($3):(1./35.) title top3c2
       with linespoints ls 10  smooth cumulative, \
  "top3/q100/gnu-files/ind/8400-ind-case-3-packet-mean.log" using ($3):(1./35.) title top3c3
       with linespoints ls 11 smooth cumulative, \
  "top3/q100/gnu-files/ind/8400-ind-case-4-packet-mean.log" using ($3):(1./35.) title top3c4
       with linespoints ls 12 smooth cumulative


set key width -1 outside below center vertical maxrows 3
set title ""
set ylabel ""
set xlabel "Service Invocation (SI) delay in milli seconds"
set output "all-tops-ua-si-delay-all-cases-1000-queries-cdf.eps"
plot    "< awk '{if ($7<=3000) {print $0}}' top1/q1000/gnu-files/ua/case-1-uaperformance-mean.
    log"  using ($7):(1./1000.) title top1c1 with linespoints ls 1 smooth cumulative, \
   "< awk '{if ($7<=3000) {print $0}}' top2/q1000/gnu-files/ua/case-1-uaperformance-mean.log"
         using ($7):(1./1000.) title top2c1 with linespoints ls 5 smooth cumulative, \
```

```
131     "< awk '{if ($7<=3000) {print $0}}' top3/q1000/gnu-files/ua/case-1-uaperformance-mean.log"
            using ($7):(1./1000.) title top3c1 with linespoints ls 9 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top1/q1000/gnu-files/ua/case-2-uaperformance-mean.log"
            using ($7):(1./1000.) title top1c2 with linespoints ls 2 smooth cumulative, \
133     "< awk '{if ($7<=3000) {print $0}}' top2/q1000/gnu-files/ua/case-2-uaperformance-mean.log"
            using ($7):(1./1000.) title top2c2 with linespoints ls 6 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top3/q1000/gnu-files/ua/case-2-uaperformance-mean.log"
            using ($7):(1./1000.) title top3c2 with linespoints ls 10 smooth cumulative, \
135     "< awk '{if ($7<=3000) {print $0}}' top1/q1000/gnu-files/ua/case-3-uaperformance-mean.log"
            using ($7):(1./1000.) title top1c3 with linespoints ls 3 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top2/q1000/gnu-files/ua/case-3-uaperformance-mean.log"
            using ($7):(1./1000.) title top2c3 with linespoints ls 7 smooth cumulative,\
137     "< awk '{if ($7<=3000) {print $0}}' top3/q1000/gnu-files/ua/case-3-uaperformance-mean.log"
            using ($7):(1./1000.) title top3c3 with linespoints ls 11 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top1/q1000/gnu-files/ua/case-4-uaperformance-mean.log"
            using ($7):(1./1000.) title top1c4 with linespoints ls 4 smooth cumulative, \
139     "< awk '{if ($7<=3000) {print $0}}' top2/q1000/gnu-files/ua/case-4-uaperformance-mean.log"
            using ($7):(1./1000.) title top2c4 with linespoints ls 8 smooth cumulative,\
        "< awk '{if ($7<=3000) {print $0}}' top3/q1000/gnu-files/ua/case-4-uaperformance-mean.log"
            using ($7):(1./1000.) title top3c4 with linespoints ls 12 smooth cumulative
141
    set output "all-tops-ua-si-delay-all-cases-100-queries-cdf.eps"
143 plot     "< awk '{if ($7<=3000) {print $0}}' top1/q100/gnu-files/ua/case-1-uaperformance-mean.
    log"  using ($7):(1./100.) title top1c1 with linespoints ls 1 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top2/q100/gnu-files/ua/case-1-uaperformance-mean.log"
            using ($7):(1./100.) title top2c1 with linespoints ls 5 smooth cumulative, \
145     "< awk '{if ($7<=3000) {print $0}}' top3/q100/gnu-files/ua/case-1-uaperformance-mean.log"
            using ($7):(1./100.) title top3c1 with linespoints ls 9 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top1/q100/gnu-files/ua/case-2-uaperformance-mean.log"
            using ($7):(1./100.) title top1c2 with linespoints ls 2 smooth cumulative, \
147     "< awk '{if ($7<=3000) {print $0}}' top2/q100/gnu-files/ua/case-2-uaperformance-mean.log"
            using ($7):(1./100.) title top2c2 with linespoints ls 6 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top3/q100/gnu-files/ua/case-2-uaperformance-mean.log"
            using ($7):(1./100.) title top3c2 with linespoints ls 10 smooth cumulative, \
149     "< awk '{if ($7<=3000) {print $0}}' top1/q100/gnu-files/ua/case-3-uaperformance-mean.log"
            using ($7):(1./100.) title top1c3 with linespoints ls 3 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top2/q100/gnu-files/ua/case-3-uaperformance-mean.log"
            using ($7):(1./100.) title top2c3 with linespoints ls 7 smooth cumulative,\
151     "< awk '{if ($7<=3000) {print $0}}' top3/q100/gnu-files/ua/case-3-uaperformance-mean.log"
            using ($7):(1./100.) title top3c3 with linespoints ls 11 smooth cumulative, \
        "< awk '{if ($7<=3000) {print $0}}' top1/q100/gnu-files/ua/case-4-uaperformance-mean.log"
            using ($7):(1./100.) title top1c4 with linespoints ls 4 smooth cumulative, \
153     "< awk '{if ($7<=3000) {print $0}}' top2/q100/gnu-files/ua/case-4-uaperformance-mean.log"
            using ($7):(1./100.) title top2c4 with linespoints ls 8 smooth cumulative,\
        "< awk '{if ($7<=3000) {print $0}}' top3/q100/gnu-files/ua/case-4-uaperformance-mean.log"
            using ($7):(1./100.) title top3c4 with linespoints ls 12 smooth cumulative
155

157 #-------------------DA-------------------------
    set xtics 1000
159 set xlabel "Time"
    set ylabel "Number of Packets"
161
    set output "all-tops-da-all-cases-q1000-line.eps"
163 plot "top1/q1000/gnu-files/da/case-1-daperformance-mean.log"  using 2:3:4 title top1c1 with
    linespoints ls 1, \
     "top2/q1000/gnu-files/da/case-1-daperformance-mean.log"  using 2:3:4 title top2c1 with
         linespoints ls 5, \
165  "top3/q1000/gnu-files/da/case-1-daperformance-mean.log"  using 2:3:4 title top3c1 with
         linespoints ls 9, \
     "top1/q1000/gnu-files/da/case-2-daperformance-mean.log"  using 2:3:4 title top1c2 with
         linespoints ls 2, \
167  "top2/q1000/gnu-files/da/case-2-daperformance-mean.log"  using 2:3:4 title top2c2 with
         linespoints ls 6, \
     "top3/q1000/gnu-files/da/case-2-daperformance-mean.log"  using 2:3:4 title top3c2 with
         linespoints ls 10, \
169  "top1/q1000/gnu-files/da/case-3-daperformance-mean.log"  using 2:3:4 title top1c3 with
         linespoints ls 3, \
     "top2/q1000/gnu-files/da/case-3-daperformance-mean.log"  using 2:3:4 title top2c3 with
         linespoints ls 7, \
171  "top3/q1000/gnu-files/da/case-3-daperformance-mean.log"  using 2:3:4 title top3c3 with
         linespoints ls 11, \
     "top1/q1000/gnu-files/da/case-4-daperformance-mean.log"  using 2:3:4 title top1c4 with
         linespoints ls 4,\
173  "top2/q1000/gnu-files/da/case-4-daperformance-mean.log"  using 2:3:4 title top2c4 with
         linespoints ls 8,\
     "top3/q1000/gnu-files/da/case-4-daperformance-mean.log"  using 2:3:4 title top3c4 with
         linespoints ls 12
```