Mobile Opportunistic Services for Experience Sharing

Via a NetInf Android Application

POURYA MORADINIA
and
IMAL SAKHI

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Mobile Opportunistic Services for Experience Sharing

Via a NetInf Android Application

Pourya Moradinia
pourya@kth.se

Imal Sakhi
imal@kth.se

2014-06-21

Master's Thesis

Examiner and Academic Supervisor
Professor Gerald Q. Maguire Jr.

Industrial advisors:
Dr. Anders Lindgren
Dr. Fehmi Ben Abdesslem

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

Information-Centric Networking (ICN) is a new research area concerning creating a new network architecture that would be more suitable for both current and the future's network. The MOSES (Mobile Opportunistic Services for Experience Sharing) project is part of this development. The project works with the development and demonstration of the Network of Information (NetInf) protocol, which is an implementation of the ICN concept.

This Master's thesis project is part of the MOSES project and aims to assist the MOSES project with the demonstration of a mobile opportunistic sharing concept based on the NetInf protocol. Demonstrating the MOSES concept in practice requires deep understanding of networking, localization, transport, and dissemination of digital content in an *ad hoc* network. This implementation requires an analysis of the previous work, development of new functionalities, and finally an analysis of a series of controlled experiments.

This Master's thesis project has designed, implemented, and evaluated an Android application within the MOSES framework by using the previously developed NetInf Android library. This prototype is used to demonstrate how mobile Android users can opportunistically share and disseminate content based on their location using the MOSES/ICN concept. The functionality and efficiency of the prototype Android application built during this thesis project has been analyzed and evaluated by conducting a series of controlled experiments under the supervision of MOSES researchers. The results of these controlled experiments has provided insight for MOSES researchers as well as explored the concept of using ICN (NetInf) for opportunistic content distribution. The experiment results aim at assisting MOSES researchers to extend and further develop the prototype application and the involved algorithms to create a fully functional mobile application for experience sharing services tailored to large-scale events.

Keywords

ICN, NetInf, MOSES, Android, sharing, content distribution, opportunistic, transport, networking

Sammanfattning

Information-Centric Networking (ICN) är ett nytt forskningsområde för att bygga en ny nätverksarkitektur mer passande för dagens och framtida nätverk. MOSES projektet är en del av denna utveckling och arbetar med utveckling och demonstration av Network of Information (NetInf) protokollet, som är en implementering av ICN konceptet.

Detta examensarbete är en del av MOSES (Mobile Opportunistic Services for Experience Sharing) projektet som syftar till att bistå MOSES projektet med demonstrationen av "mobile opportunistic sharing" konceptet som bygger på NetInf protokollet. Att demonstrera MOSES konceptet i praktiken krävs djupt förståelse om nätverk, lokalisering, transport och spridning av digitalt data i en "ad hoc" och infrastruktur miljö. Genomförandet av denna implementering kräver en analys av tidigare arbete, utveckling av nya funktioner och slutligen analys av genomförda experiment och resultaten.

Detta examensarbete har utformat, genomfört och utvärderat en Android applikation inom ramen för MOSES med hjälp av tidigare utvecklat NetInf Android bibliotek. Denna prototyp används för att visa hur mobila Android användare opportunistiskt kan dela och sprida innehåll baserat på deras plats med hjälp av MOSES/ICN konceptet. Funktionaliteten och effektiviteten av prototypen byggd under detta examensarbete har analyserats och utvärderats genom att utföra en serie kontrollerade experiment under ledning av MOSES forskare. Resultaten av dessa kontrollerade experiment har gett insikt åt MOSES forskare samt utforskat konceptet att använda ICN (NetInf) för opportunistisk distribution av innehåll. Experimentens resultat syftar till att hjälpa MOSES forskare att utöka och vidareutveckla prototypen och de involverade algoritmer för att skapa en fullt fungerande mobil applikation för "experience sharing services" anpassad för stora evenemang

Nyckelord

ICN NetInf MOSES Android dela data information sprida spridning opportunistiskt transport nätverk

Table of contents

List of Figures

List of Tables

List of acronyms and abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
CL	Convergence Layer
EIT	European Institute of Technology
geoTagged	geographically Tagged
GPS	Global Positioning System
GUI	Graphical User Interface
ICN	Information-Centric Networking
ICT	Information and Communication Technologies
MAC	Media Access and Control
MOSES	Mobile Opportunistic Services for Experience Sharing
NetInf	Network of Information
NDO	Named Data Object
NI	Named Information
NTP	Network Time Protocol
PKI	Public Key Infrastructure
RSSI	Received Signal Strength Indication
SICS	Swedish Institute of Computer Science
Sync	Synchronize
UAT	User Acceptance Testing
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier

1 Introduction

This chapter describes the specific problem that this thesis addresses, the context of the problem, the goals of this thesis project, and outlines the structure of the thesis.

1.1 General introduction to the area

Information-Centric Networking (ICN)[1] is a concept for an Internet architecture where the focus is on the *content*, rather than the communication between *hosts* as in today's networks. This new architecture intends to be more suitable for content distribution and to mitigate the impact of communication disruptions. In ICN architecture any content (e.g. image, music, text, etc.) is treated as an object which can be cached at different locations in the network. Unlike the current Internet architecture, servers or other devices that are capable of retaining a copy of an object can implement these caches. When a user requests a specific object from the network, the network will locate the object (with the help of locators) within one of the caches with a copy of the object and then will deliver a copy of the object to the requester. With this architecture, ICN attempts to decentralize content sharing and offload traffic from content servers. ICN aims for a future Internet that is scalable and well suited for content sharing and distribution.

Network of Information (NetInf)[2] is a protocol based on the ICN architecture. The NetInf protocol implements different features of the ICN architecture and allows creation and exchange of Named Data Objects (NDOs) between NetInf nodes. NDOs are ICN objects that are named with a standardized Uniform Resource Identifier (URI). An NDO could be any type of content (e.g. image, music, text). In a NetInf network a user will request an NDO and the NetInf network will retrieve the NDO from the nearest* cached location.

Mobile Opportunistic Services for Experience Sharing (MOSES) is a research project that “will adapt and extend the algorithms and software systems built in the carrier projects to create a specific mobile application for experience sharing between visitors (using live images, videos) of an entertainment venue such as Expo 2015. The resulting system will (i) reduce the load on the mobile network (offloading) and (ii) provide a platform allowing instant refinements, creation, and sharing of new apps.”[3] The prototype developed during this Master's thesis project may become an experiment platform for experimenting with the opportunistic sharing concept by MOSES partners or the research community.

1.2 Problem definition

The ICN/NetInf concept is a new approach for eliminating the flaws of the current Internet architecture, as well as optimizing the current functionalities of the Internet. Opportunistic content sharing is a concept built upon the ICN/NetInf architecture. This opportunistic content sharing intends to decentralize and offload traffic from the original content storage infrastructure for common and popular content. Users near each other who request the same content should retrieve it from each other, rather than from the original content-servers. This new architecture shifts the traffic due to requests by end-users trying to retrieve the same content from long geographical distance to short distances. This is especially valuable in a flash-crowd scenario.

The idea of distributed dissemination of user-content is to make relevant content easily available where it is most likely to be requested. In this approach, mobile users will mostly operate their wireless local area network interface in *ad hoc* mode, but will use infrastructure mode in order to share and synchronize location relevant content with NetInf-Servers for distribution purposes.

* Nearest in a network sense, not necessarily nearest in a geographical sense.

Caching and sharing NDOs consumes crucial resources (e.g. random-access memory and battery power) of the end-user's mobile device. Mobile devices should operate in a suitable manner to share and disseminate as many NDOs as possible, while using the minimum amount of the mobile device's *constrained* resources. The idea is to trade available local resources of a collection of user devices for a reduction in wide area network traffic.

1.3 Problem context

This Master's thesis project is part of the MOSES project within the EIT ICT Labs Future Networking Systems action line. The goal of the MOSES project is to develop an architecture and platform that exploits opportunistic communication among mobile users[4].

1.4 Goals

The main goal of this thesis is to assist the MOSES research community with a proof of the mobile opportunistic sharing concept by designing and implementing a prototype that will be evaluated during a series of controlled experiments.

Using the Python and Android NetInf libraries provided by the MOSES project, this Master's thesis project aims to design and implement an Android application which includes NetInf features for mobile end-users. This Android application will demonstrate how users can opportunistically share content based on their location(s). The contents shared by this Android application will be geographically tagged (geoTagged) with the coordinates of the location that the content is relevant to. Indoors, Bluetooth Beacons provide these coordinates as explained in Section 4.1. A use case for this application is that a user can opportunistically exchange location relevant content with other users in the *physical* vicinity of this user.

Implementing features of the NetInf protocol in this Android application will demonstrate some of the functionalities of the ICN/NetInf concept in practice. This implementation will mainly contribute to a prototype for conducting a series of controlled experiments that may point out key factors regarding the MOSES/ICN concept. The results of these controlled experiments are intended for researchers in the MOSES area to reproduce or further develop the scenarios of these experiments as necessary.

Besides its use in the experiments, this prototype will also demonstrate an application of MOSES for end users, which may generally be a contribution to the ICN research.

1.5 Research Methodology

ICN/NetInf is a new networking architecture, which is quite different from the current host-centric TCP/IP Internet architecture. In order to design, implement, and evaluate a prototype for this new Internet architecture, the Design Science[5] research methodology has been used in this thesis.

The Android application has been iteratively[6][7] designed and implemented based on the NetInf protocol and is expected to operate within the framework of the mobile opportunistic sharing concept. This Android application has followed the guidelines of the MOSES project and has been implemented according with the instructions of our respective supervisors, who are part of the MOSES research community.

The Android application is the prototype, which has been experimentally evaluated by a series of controlled experiments. The results of each controlled experiment is analyzed in terms of a set of performance metrics, the purpose of this analysis has been to attempt to identify anomalies, factors of efficiency, and factors of inefficiency in the opportunistic sharing concept in the context of MOSES.

1.6 Structure of this thesis

Chapter 1 provides background about the technologies and concepts used in this thesis. Chapter 3 describes the methodology used in this project for achieving the goals stated in Section 1.4. Chapter 3 presents in detail the actual implementation and its components. Chapter 5 presents the analysis of the proposed solution and the metrics for the evaluation of the results of the controlled experiments. Chapter 6 briefly describes some of material that was developed during this thesis project but which has not been discussed in this thesis, this material may be of use in future projects. Finally, Chapter 7 presents a conclusion and suggests potential future work.

2 Background

Chapter 2 starts by giving a background to ICN, followed by an introduction to the NetInf architecture and its messaging and naming scheme. Section 2.3 reviews the Python code and Android Library used and developed in this project. Section 2.4 discusses the localization of objects by geoTagging and the significance of geoTagged content in mobile opportunistic sharing. Section 2.5 presents a use case for this Android implementation and its contribution to the MOSES project. The chapter concludes with a brief comparison of different ICN models, including NetInf.

2.1 Information-Centric Networking

This section provides an overview of the ICN framework in general and the NetInf implementation of ICN in particular. In the context of this thesis the words “content”, “data”, and “information” are used interchangeably to refer to the same entity: an NDO.

As mentioned in Chapter 1, ICN is an alternative to today’s host-centric networking (one-to-one communication) model. There has been an increasing demand for highly efficient and scalable content distribution in recent times. This demand has led to the development of new Internet architectures. All that matters for many end users is rapid access to the content that they want *without* being limited to acquiring this content from a specific host (as is the case in the present Internet architecture). ICN is a research area that focuses primarily on fast retrieval of content, rather than focusing on which node to retrieve this content *from*. Ideally, the requested content could be provided by any node that has this data available as opposed to the current model where the content needs to be retrieved from a specific node. There have been several research projects regarding ICN both in Europe (PSIRP[8], 4WARD[8]) and in USA (CCN[8], DONA[9][10]). This Master’s thesis project is based on the NetInf implementation of ICN which is a result of FP7 project 4WARD*, later continued in the FP7 project SAIL†.

2.2 NetInf

As with all other ICN implementations, in NetInf‡ communication is driven by users requesting NDOs and the network satisfying the request by retrieving the NDO from any entity with a copy of the NDO. The main constituents of the NetInf protocol are NDOs, messages, and a convergence layer. The following subsections will describe each of these.

2.2.1 Named Data Objects (NDOs)

A common abstraction used by all ICN implementations is a naming scheme that enables the identification of information objects *independent* of their location. Unlike today’s Internet architecture where the *storage locations* of content are named (by a Uniform Resource Locator (URL) or the IP address of a server), in ICN the *content* itself is named. In NetInf an NDO is simply a block of information stored somewhere in the network. An NDO can be a text file, a web page, a picture, or a streaming video. There are two components to this NDO model: a naming scheme and metadata objects:

* http://www.netinf.org/4ward-netinf/ and http://www.4ward-project.eu/
† http://www.netinf.org/sail-netinf/ and http://www.sail-project.eu/
‡ http://www.netinf.org/

`Naming Scheme`

NetInf uses the Named Information (NI)[11] naming scheme for naming NDOs. A NI Uniform Resource Identifier (URI) has the following general format:
ni://[Authority]/<DigestAlgorithm>;<Digest Value>
The `Authority` field is optional and assists applications in finding a given NDO. The `Digest Algorithm` field is mandatory and specifies the digest algorithm that has been applied to the binary data of the NDO to generate the corresponding digest value. The `Digest Value` field is also mandatory and is the output of the specified digest algorithm. Two NDOs are equivalent if they have the same digest algorithm and same digest value. An NDO's name does not change depending on how or where it is stored.

`Metadata`

A metadata object is a useful component of an NDO, which will present convenient information about the respective NDO. A great deal of useful information may be included in the metadata to assist a NDO retriever by providing information about the content of the NDO. For example, this information can include content type, timestamp, URI, size, and search tokens. For example, a receiver could request the metadata of an object and get an overview of the object's content before retrieving the complete NDO. This could be very useful in a search scenario.

2.2.2 Message types

NetInf is a message-based protocol with requests and responses for publishing, getting, and searching for NDOs. NetInf consists of three basic requests (`GET`, `PUBLISH`, and `SEARCH`) and their corresponding response message types (`GET-RESPONSE`, `PUBLISH-RESPONSE`, and `SEARCH-RESPONSE`). All of these message types should contain a message identifier (`msg-id`). It is recommended that the `msg-id` should be chosen in a way to make collisions with another `msg-id` unlikely and that this message identifier must **not** contain any information that might be personally identifying, e.g. an IP address or username. It is recommended that a sufficiently long random string should be used for each `msg-id`. The details of each message type are:

`GET/GET-RESPONSE`

A GET message is sent to request an NDO from a NetInf network using the NI URI scheme. A node receiving a GET request would send a corresponding GET-RESPONSE with the same msg-id if it has an instance of the requested NDO. If the node does not have the NDO available, then it should forward the request and then generate a GET-RESPONSE after waiting a reasonable time for a response (depending on the implementation). The format of a GET-RESPONSE depends on whether the node returns both content octets of the requested NDO & its affiliated data* or only the affiliated data when the content is not available in node's cache. In the latter case, the GET-RESPONSE message would only contain affiliated data (such as locators as to where the NDO is available). The locator could be an IP address or a network interface's media access and control (MAC) address depending on the underlying convergence layer used.

* Affiliated data describes the "overall" data *excluding* the content octets of a NDO [12].

PUBLISH/PUBLISH-RESPONSE	As the name suggests, the PUBLISH message enables a node to announce to other nodes that it has a given NDO. In addition to the NDO name, the PUBLISH message contains a copy of the object octets and/or object meta-data. A node receiving the PUBLISH message may choose to add the metadata and content to its cache if it does not already have the NDO stored; while if the receiver already has a copy of the content, then it should merge the metadata. Ignoring extensions*, the corresponding PUBLISH-RESPONSE message contains only a status code and affiliated metadata.
SEARCH/SEARCH-RESPONSE	The SEARCH message enables a requester to find an NDO by specifying search keywords in the search message. In addition to the mandatory `msg-id` field the SEARCH message has another mandatory field known as the `tokens field` which is basically a query string specifying what is being searched for. As with GET messages, a SEARCH request may also be forwarded to other nodes. Upon receiving a SEARCH message, the receiving node carries out a search in its cache using the tokens and returns a SEARCH-RESPONSE message. The response message contains a list of NDOs matching the tokens and their affiliated meta-data corresponding to each NDO in order to assist the searcher to choose between the search results.

2.2.3 Convergence Layer

A Convergence Layer (CL) enables communication between NetInf nodes and provides framing and message integrity as its main services. Any protocol that allows NetInf messages to be passed without loss of information can be used as a NetInf CL. Initially two convergence layers were proposed[2][8]: HTTP CL and UDP CL. As the names suggest, they use HTTP and UDP services to transfer NetInf messages between nodes. E. Davis has described how to implement the NetInf Protocol with HTTP and DTN CLs[15]. A new Bluetooth CL has been proposed and introduced as a result of a previous student's Master's thesis[16]. A CL must fulfill at least the following requirements stated in the expired Internet draft draft-kutscher-icnrg-netinf-proto-01[2]:

- Unidirectional point-to-point transport of NetInf messages,
- Preservation of message boundaries,
- Reliable transmission of messages, and
- In-order delivery of messages to the destination node.

2.3 Python and Android NetInf library

The MOSES project provided the authors of this Master's thesis with the source code of the Python NetInf library v0.4[17]. This Python implementation may act as a NetInf Server and can do PUBLISH, GET, and SEARCH for NDOs both locally and via other NetInf Servers. Throughout this thesis, references are made to NetInf servers, these only refer to NetInf servers in the infrastructure and **not** to a NetInf server in an *ad hoc* network. In the context of MOSES, NetInf servers in the infrastructure may be used for content dissemination purposes. Users push/pull content to/from NetInf servers when possible.

The Android application developed in this Master's thesis has used the Python NetInf library v0.4 as the NetInf server in infrastructure mode, only for interoperability purposes in case of future development of this application; the focus of this thesis is on *ad hoc* networking and the Bluetooth CL. The message formats used by this Android application over the HTTP CL are according to the Python library v0.4 implementation. The Android application in this Master's thesis is based on this Python

* Some extensions are described in [13] and [14].

NetInf implementation and the NetInf Android library developed by Linus Sundé during a previous Master's thesis project[16].

2.4 GeoTagged objects and their coordinates

GeoTagging is the process of adding geographical information to an object. The tag provides a form of geospatial metadata and usually consists of latitude and longitude coordinates and/or other information, such as altitude, distance, proximity, etc. This geographical data might be obtained through GPS, cellular network, Wi-Fi, or Bluetooth signals. GeoTagging can provide location-specific information to end users.

The geographical location data used in geoTagging can be in the form of latitude/longitude-coordinate system either in decimal degrees (DD) format or in degrees, minutes, and seconds (DMS) format. When using Wi-Fi or Bluetooth devices for geoTagging, one can use a device's MAC addresses or Bluetooth ID to tag content. The prototype Android application senses the received signal strength of Bluetooth signals and tags the content with the ID of the Bluetooth device that has the strongest received signal strength.

2.5 Use case

ICN opens a broad area for new applications that might be infeasible with today's Internet architecture. One type of application that MOSES is investigating, is local content sharing *without* access to Internet (i.e. via an *ad hoc* network).

Assume a scenario where User A enters an art exhibition without any former knowledge about the art objects being exhibited. Fortunately, User B who is an art specialist, goes around the gallery and captures photos of the objects and comments upon them*. User A that is very interested in these art objects and is eager to know more about them. To get more information about these objects User A opportunistically attempts to synchronize with user B over Bluetooth. In this synchronization process User A fetches User B's locally generated (published) commented photos (as NDOs). User B's NDOs are geoTagged with indoor location obtained from localization beacons (explained in Section 4.1), therefore User A does not need to explain to User B what type of information he/she is interested in. User B will only give location relevant content to User A. Assuming that the application is smart enough, User A will receive User B's photos as a popup when passing by every art object in the gallery.

This use case is one of the many scenarios where applications of MOSES could be used. One of the ideas of MOSES is content sharing *without* the need to explicitly search for the material on the Internet, thus decentralizing content dissemination.

2.6 Comparison of alternative ICN models

This section provides a short comparison of several different ICN approaches. Although each implementation's approach to ICN differs in the details, the overall goal is to develop a network architecture that is better suited to efficiently accessing and distributing content. The comparison will consider the following four ICN approaches:

- Data-Oriented Network Architecture (DONA),
- Content-Centric Networking (CCN),
- Publish-Subscribe Internet Routing Paradigm (PSIRP), and
- Network of Information (NetInf).

* We assume for the purposes of this scenario that User B is permitted to take pictures of the art objects being exhibited and that they are permitted to communicate via Bluetooth with other nearby users.

Section 2.6.1 reviews the commonalities of these four ICN designs, while Section 2.6.2 discusses the basic differences between these four approaches.

2.6.1 Commonalities of Designs

Although each implementation has its own distinctive terminologies, they all share three fundamental principles: use a publish/subscribe paradigm, exploit universal caching, and utilize a content oriented security model.

2.6.1.1 Publish/Subscribe paradigm

All four ICN approaches make use of the publish/subscribe paradigm. PUBLISH enables nodes to advertise the availability of content, while SUBSCRIBE enables consumers to request content. Using the publish/subscribe paradigm decouples requests and responses both in time and space. The provider and requester of the content are not required to know the location of each other nor be online at the same time in order to publish or receive content. This decoupling mechanism is one of the most profound aspects of ICN architecture.

2.6.1.2 Universal Caching

In ICN architecture when a node receives a request for content, the node has to perform one of the following two actions:

1. If the content is available in its own cache, then respond directly with the content.
2. If the content is not available in its local cache, then the node can request the content from its peer(s). When it receives the content then the node can send a response back to the original requester with the content and the node can cache this content for later use.

The caching mechanism applies to content carried by any protocol, and generated by any user, thus the content is not limited to content from a specific content provider or carried by any specific protocol. This decentralizes content distribution and delivery. In all four ICNs caching is implemented by all ICN nodes, rather than only by a few specialized caches.

2.6.1.3 Content-oriented Security Model

In an ICN, the content can be fetched from any network element rather than the originating server and thus the current security model of securing a *connection* between end points is of no use. Instead, in ICN the content itself needs to be secured. The authenticity of the content is established by having the original content provider sign the content. Now all of the receiving network elements can verify the authenticity of the content by verifying this signature. This content-oriented security model is shared by all four ICN approaches and is regarded as name-data integrity. The confidentiality part of this model is unclear[18], therefore for now only data integrity is provided.

2.6.2 Fundamental Differences in Design

Besides the commonalities described in the previous subsection, there are some fundamental differences. These differences are described further in this section in terms of the difference between naming, inter-domain name-based routing, and transport.

2.6.2.1 Naming

As stated previously, in ICN, consumers request objects by name rather than from a specific network node. There are two main naming systems proposed for the ICN architecture. The first naming scheme uses hierarchical human-readable names, similar to today's DNS system. In this naming scheme, a variety of techniques can be used to allow a user to learn a public key, but this usually requires a globally-agreed-upon public key infrastructure (PKI) to bind names to keys in order to achieve name-data integrity.

The second naming system uses self-certifying names. In this case, the key is bound to the name itself, removing the need for a PKI. The disadvantage is that these names are not human-readable, so consumers have to use other techniques to determine the name of the content that they desire.

DONA uses a flat namespace in the form P:L, where P is a globally unique `principal` field containing the cryptographic hash of the publisher's public key, and L is a unique `object label`. PSIRP also uses a flat namespace, but has a slightly different approach to it than DONA. NetInf also uses a flat namespace and includes the hash digests in the name. NetInf can use a number of different hashing schemes. CCN uses a hierarchical namespace in order to achieve better routing scalability through name-prefix aggregation.

2.6.2.2 Inter-domain Name-Based Routing

In order to distribute content efficiently, ICN systems have to route requests. Two main routing approaches are used in ICN architectures:

Name resolution	This is similar to today's DNS system. In this model, a resolution service is queried for a given name and then one or more lower-layer locators are returned. These locators are then queried to retrieve the content, using a protocol such as HTTP or directly via another IP based protocol.
Name-based routing	An alternative approach to name resolution routing is to directly forward the request to an object copy in the network based on the object's name, without initially resolving the object name into some lower-layer locator(s). This approach is often referred to as name-based routing.

DONA and CCN use name-based routing to route queries, while PSIRP uses a name resolution model where the resolver is called the rendezvous point. NetInf supports both name resolution and name-based routing for retrieving data objects.

2.6.2.3 Transport

The transport layer is responsible for forwarding requests and responses as well providing basic transport layer functionalities (congestion control, flow control, and reliability).

The DONA architecture relies on existing transport protocols such as TCP and UDP. The CCN architecture defines two different transport layer packet types: interest packets and data packets. A node that sends an interest packet via an interface to a (or a set of) neighboring node expects to receive a corresponding data packet in response.

The NetInf architecture uses a CL for its transport layer. The CL can be HTTP, TCP, UDP, Bluetooth, etc. as long as this CL provides reliable and in-order delivery of requests and response messages. There could be multiple hops involved in forwarding such request and response messages, and each hop can potentially use a different convergence layer.

3 Method

One of the main phases of this thesis project was to design and implement a prototype. This prototype was evaluated in the second phase of this thesis project in a series of controlled experiments. An iterative[6][7] approach (shown in Figure 3-1) was chosen to design and implement this prototype. A literature review concerning ICN, NetInf, and MOSES was performed to gain a clear understanding of the concepts underlying such a prototype. This review examined the basic goals of MOSES and the architecture of ICN. The NetInf protocol inherited this architecture. A review was made of an Android NetInf library implemented during a previous Master's thesis project. This review assisted in pinpointing functions that could be utilized in our prototype.

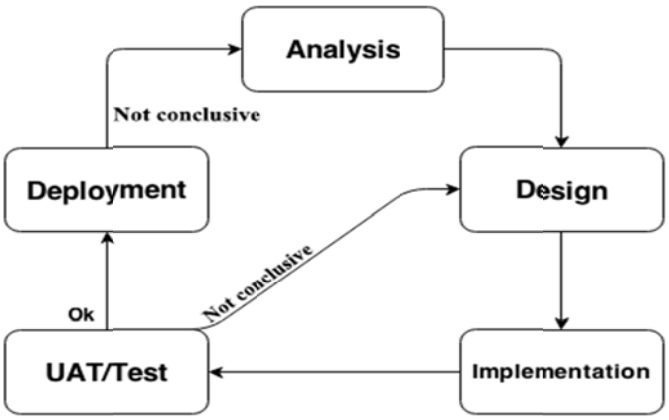


Figure 3-1: Iterative design approach

After clarifying the basic architecture of the prototype, the prototype underwent a cyclic process of design, implementation, and User Acceptance Testing (UAT). In each iteration of this cyclic process, the UAT was demonstrated to our supervisors who are part of the MOSES research community. Each UAT was demonstrated both by observations of the prototype's behavior in the field and by observations of logs. The final UAT was an uncontrolled experiment. The results of this UAT were evaluated in terms of a set of performance metrics. The results of this uncontrolled experiment will *not* be present in this thesis since the intentions of this thesis project were to generally evaluate the mobile opportunistic sharing concept and its corresponding deployed CL, rather than focusing on an evaluation of the prototype itself.

In the final phase of this thesis project, the prototype was deployed in the Swedish Institute of Computer Science (SICS) office in Kista in order to conduct a series of controlled experiments. The results of these experiments were analyzed and evaluated in terms of a set of performance metrics. Chapter 5 presents this analysis.

4 Implementation of a NetInf Android Application

In order to demonstrate an application of MOSES in practice an Android application was designed from scratch. A prototype of this application was then implemented and used in a series of controlled experiments. The main idea behind the design of this application was to demonstrate the basic functionalities that are of interest for MOSES (i.e., geoTagging, GET, PUBLISH, and *ad hoc* networking). For this reason, a simple application was designed to allow users to capture images, name them, and share them over a short-ranged wireless channel, specifically Bluetooth, by using the NetInf protocol. In order to respect the mobile opportunistic sharing concept, these images are geoTagged by an indoor localization service.

MOSES intends these types of applications be used by end users, therefore a user-friendly Graphical User Interface (GUI) was developed in order to make this application more interesting for non-technical users. For the same reason, the GUI avoids using NetInf technical terms, such as GET, PUBLISH, or NDO. In order to make the GUI even more interesting for an end user, an indoor localization service uses the GUI to show the user's approximate indoor position in real time. Furthermore, hard coding of experimental options, such as enabling/disabling services and modifying timeouts/intervals has been avoided - as this prototype application was to be used and experimented with mainly by MOSES researchers, rather than real end users.

Using the Android NetInf library[16] an Android application was developed to implement the NetInf features of GET, SEARCH, and PUBLISH in the Android environment. This Android application was designed to operate over a wireless local area network interface in both *ad hoc* and infrastructure mode. The Python NetInf Server code v0.4_Elwyn[17] was slightly altered to make it compatible with this Android application. Note that this application was primarily designed for *ad hoc* networks using the Bluetooth CL, the HTTP implementation is simply a service directly provided by the NetInf library to facilitate further development of this application.

GeoTagging, PUBLISH, GET, SEARCH, NDOs, metadata, and opportunistic sharing are implemented in this application to demonstrate both their functionalities and to contribute to a prototype demonstrating the mobile opportunistic sharing concept.

4.1 GeoTagging

The NetInf Android application can use GPS, cellular network, Wi-Fi, or Bluetooth signals for geoTagging content. However, the current prototype uses Bluetooth transmitters known as “Estimote Beacons” both for localization purposes and for geoTagging content. An Estimote Beacon (shown in Figure 4-1) is a small, wireless device that periodically broadcasts low power radio signals. Each Estimote Beacon consists of a 32-bit ARM Cortex M0 CPU with 256kB flash memory, accelerometer, temperature sensor, and a 2.4 GHz Bluetooth 4.0 Smart (also known as Bluetooth Low Energy, BLE) radio transceiver. A soft Silicone casing covers the hardware and has a sticky backside that can be used to attach the Beacon to any flat surface (be it wood, concrete, or glass).

Figure 4-1: Estimote Beacon. The community manager of Estimote granted permission (by E-Mail) for use of this image in this thesis[19].

The “Estimote” Beacon periodically advertises the following information every 200ms (this advertising interval can be changed):

- Fixed beacons containing a universally unique identifier (UUID): 0xB9407F30F5F8466EAFF925556B57FE6D,
- A two byte long “Major” value,
- A two byte long “Minor” value, and
- A one byte long measured Received Signal Strength Indication (RSSI) value.

The major and minor values are integer values that can be used to divide an area into different zones and sub-zones. These values can be manually changed by using the proprietary “Estimote” application (available at Google Play). For the NetInf demonstration application, 10 Estimote Beacon transmitters are used. These transmitter were placed at different locations around the SICS Kista office as specified in Figure 4-2. For purposes of the demonstration, the SICS office was divided into two zones, thus two different “Major” values were used. Each Estimote Beacon was configured with a unique “Minor” value associated with the corresponding “Major” value depending on where it is placed.

The Android application continuously listens for Estimote Beacon advertisements. Once a user has captured an image using their device’s camera, given this image a title, and pushed the “publish” button, the image is locally published after being tagged with the nearest (based upon with strongest RSSI value) beacon’s major and minor values. This geoTagging information is published as part of the metadata of the NDO along with a timestamp and other relevant information. The timestamps are based upon the local device's clock that has been automatically synchronized by different Network Time Protocol (NTP) servers provided by device's firmware.

4.2 Convergence Layers

Currently three CLs have been developed for the NetInf protocol: HTTP, UDP, and Bluetooth. The NetInf Android application uses HTTP and Bluetooth CLs for transport purposes. The Bluetooth CL is used only for *ad hoc* communication (i.e., directly between mobile devices), while the HTTP CL is used both between mobile devices (through an access point) and infrastructure attached servers (i.e., between a mobile device and a central NetInf server). A user can choose to enable either one or both of these CLs at a time. For the Bluetooth CL to work the mobile devices has to be paired in advance with each potential Bluetooth peer. The end user can type in individual Bluetooth device names or choose to communicate with all paired devices in the settings menu. In contrast, for HTTP CL to work the end user has to manually type in the IP addresses of all of the other devices/servers via the settings menu. For the demonstration application, the server’s IP address and Bluetooth paired devices settings have been pre-chosen by default, but these settings can be altered by the end-user. If both CLs are enabled, then PUBLISH, GET, or SEARCH requests for an NDO are sent over both CLs to their respective configured devices. Even though both CLs are functional in this application for demonstration of the mobile opportunistic sharing concept (i.e. *ad hoc* network), only the Bluetooth CL was used for the experiments and evaluation of this prototype. Even though MOSES may time to time use the infrastructure for disseminating content to the Internet, an infrastructure mode was not realized in this thesis project and could be incrementally added for optimization and further development of this application in the future.

4.3 Graphical User Interface (GUI)

The subject group for this Android application is end-users, therefore during the design phase end-users were the focus of the GUI. Factors such as simple access to functions, smooth graphic-flow, choice of color, and layout were considered when developing what was expected to be a user-friendly GUI.

In the scenario, illustrated in Figure 4-2, the application is being used in SICS's Kista office. The Estimote Bluetooth Beacons, as explained in Section 4.1, provide indoor localization. The Estimote Beacon transmitters (marked with blue filled circles in Figure 4-2) are placed at fixed positions in the SICS office. The coverage areas of all these beacons purposely overlap each other for indoor localization purposes. The Beacons were set to have a transmitting range of 70 meters, but in practice they covered ~10-15 meters due interference in the 2.4GHz channel and signal loss due to walls and other material in the environment. The application can sense up to five beacons depending on the user's position within the office.

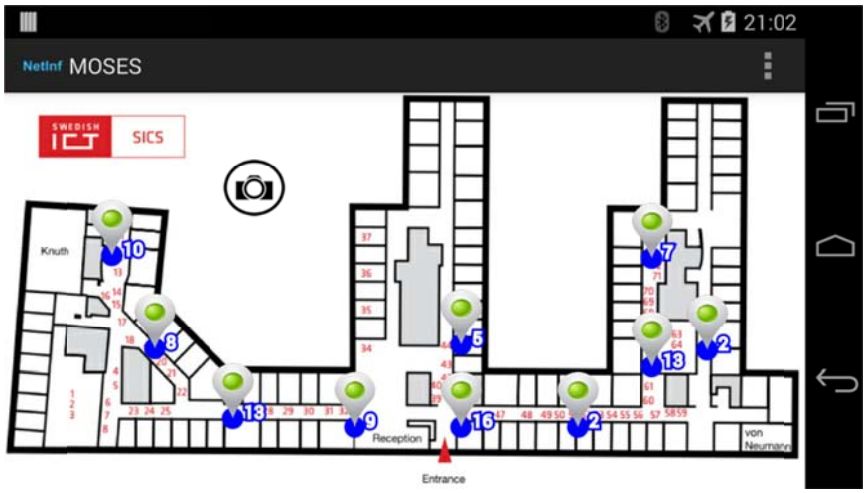


Figure 4-2: Android application GUI

The "Camera" button of the application allows the user to capture an image and assign this image a title. The captured image uses the PUBLISH function of the NetInf protocol to publish the corresponding image locally via the end-user's device together with metadata (including the title, timestamp, and other protocol-based information such as content-type, object-size, etc.). The metadata also includes information about the Estimote Beacon with the strongest RSSI. As a result, the captured images are geoTagged with the label of the closest Beacon. GeoTagging of NDOs in the context of this project is mandatory, therefore the application tries up to 5 times to listen for Estimote Beacons within range, if no Beacons are sensed after 5 tries then no NDOs are published. The user is informed that the image publish operation failed and is asked to retry again within 10-20 seconds (this failure is likely to be due to Android/Estimote Application Programming Interface (API) misbehavior and could be improved in future API updates).

The Beacons on the floor plan in Figure 4-2 are created as an instance of the FloorObject class, which has the same coordinate system as its parent class. This allows the application to be aware of the coordinates of each object/Beacon on the floor plan. In this way, the application can use the tagged Beacon information in the NDO's metadata to map each NDO in the database to the corresponding Beacon on the floor plan. As shown in Figure 4-3 by pressing on a Beacon displayed by the application, the user can view information about the corresponding Beacon and the images tagged with its location. By pressing on the "View images" item in the Beacon popup window, the user will see a list of NDOs organized by their title and the time they were published. The user can then select an NDO and view the image.

It is convenient for the user to be able to view the number of available NDOs at each location in an overview manner. As shown in Figure 4-3 a function in the application shows the number of available NDOs at each location every time an NDO is added or removed from the database. This database is an Android SQLite database provided by the NetInf library. The database holds all the components (e.g. metadata, NI, locator(s), etc) of an NDO except for the actual octets of the NDO which is separately saved in the file system in a folder called "ndo_cache".

Figure 4-3: GUI Beacon information

In order for the application to be user friendly, preferences can be selected in the NetInf library as well as the main application (see Figure 4-4) so that the user can at any time adjust the parameters of the different functions. Most of the NetInf settings are set to a default value (this is transparent to users). However, since this application is still in its early phases of development, many settings exist solely for testing purposes or used by advanced users (i.e. MOSES researchers).

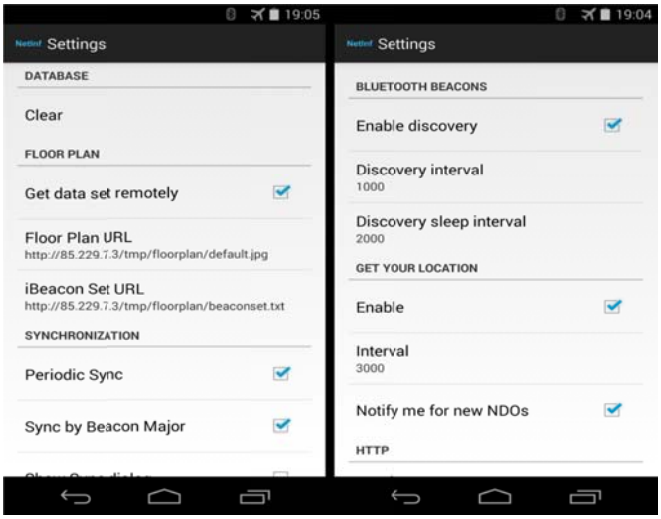


Figure 4-4: GUI user settings

The Estimote Beacons are positioned at fixed locations in the SICS office leading to overlapping neighboring Beacons. The intention was to allow the application to triangulate the user's approximate position in the office. This functionality makes the application more user friendly as the user is aware of their approximate position at all times. The reason for estimating an approximate position rather than an exact position is because in this test scenario a limited number of Estimote Beacons were deployed. For better triangulation, three or more Beacon signals are required at all times and in this scenario there were spots where the user had access to only one Beacon. The application considers the three best signals that were sensed and then calculates the average of the corresponding Beacon's coordinates. As shown in Figure 4-5 the large blue circle indicates user's approximate position. The accuracy of the position computation produced by this application has *not* been evaluated due to the reason that the triangulation function of this application is more of a bonus functionality and not a requirement of MOSES. However, during repetitive observations (by human eye) of this triangulation function during the whole period of this thesis project, no major false results were observed.

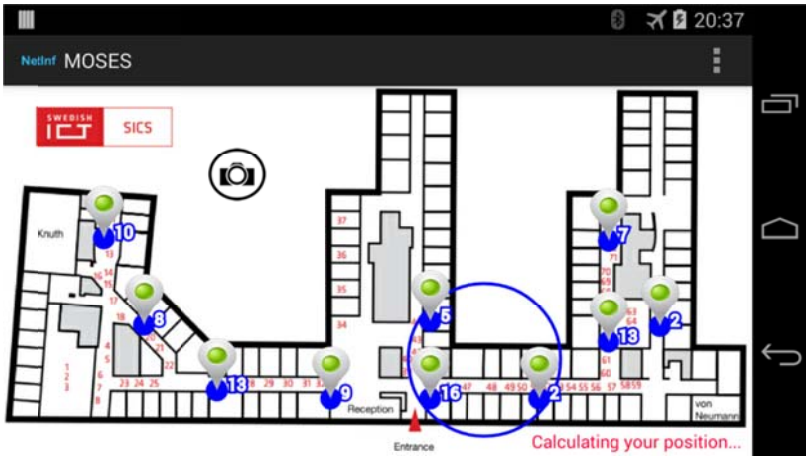


Figure 4-5: Triangulation

As illustrated in Figure 4-5 there is a small debugging line output in red text at the lower right corner of the screen to inform the user in simple words about background activities. This function is informative for the user in case of performance delays or misbehavior of the application. User friendly outputs such as "Sending Photos...", "Received Photo!" and similar simple terms are used to inform the end user of more or less what the application is doing.

Furthermore, an alert function has been implemented in the application to notify the user about the new unseen NDOs when the user arrives in the vicinity of a given Beacon. The alert function will change the corresponding Beacon's color to green on the floor map, vibrate the phone for 2 seconds, and show a popup window with information about the new NDOs available for this location.

4.4 Synchronization

In the context of this Master's thesis project, synchronization is one of the essential functions and is rather very complex. Who to share with? How to share in a fairly manner? How many devices to synchronize with simultaneously? When to synchronize? And many more questions need to be addressed when it comes to efficient synchronization. The main objective of this thesis project was to contribute to a prototype, in which basic synchronization is performed.

In this thesis project a synchronization (Sync) process has been designed from scratch and the focus has been to demonstrate simple NetInf tasks, such as SEARCH and GET. The PUBLISH function of NetInf has been ignored in this design since forcing NDOs to a remote user was not to be the main task for this prototype. For this reason, the Sync process is unidirectional, meaning the user who initiates the Sync process will be synchronized with the remote user, while the remote user is left unsynchronized.

As mentioned in the goals of this thesis project, this application was primarily meant to be used for experimental purposes, therefore the design of this Sync process is mainly meant to fulfill the requirements of the controlled experiments. The Sync process and its components will be fully described and explored in Chapter 5, where each controlled experiment will be discussed. However, from an overview perspective the Sync process sends a SEARCH and then based on the SEARCH-RESPONSE calculates those NDOs that do *not* exist in device's local cache and then performs a GET for each NDO that the device does not (yet) have.

In Section 4.1 when describing the Estimote Beacons, it was mentioned that in this application, the SICS office has been virtually divided into two zones with two different Beacon Major numbers as shown in Figure 4-6. The idea behind this separation was to allow the users of this application to synchronize location relevant content. This implementation allows users to choose from the preferences menu whether to synchronize only location relevant content or fetch everything a remote device has. In this scenario location relevant means content belonging to zone A or Zone B, e.g. if a

user namely User A finds his/her self in zone A and meets another user, namely User B in zone B or zone A, then User A will only receive content for zone A from User B. Technically in terms of NetInf this is done by sending a SEARCH either with token "*" or "Beacon Major 5001/5002". Token "*" tells the remote device to respond with a SEARCH-RESPONSE with all NDOs (without the actual octets of the object, which is called Fullput in NetInf) existing in device's entire cache. Token "Beacon Major 5002/5001" is the Beacon Major (advertised by the Estimote Beacons in the office) that the device (which initialized the SEARCH request) is sensing, which tells the remote device to respond with a SEARCH-RESPONSE with all NDOs (rather than Fullput) that are tagged with this Beacon Major value.

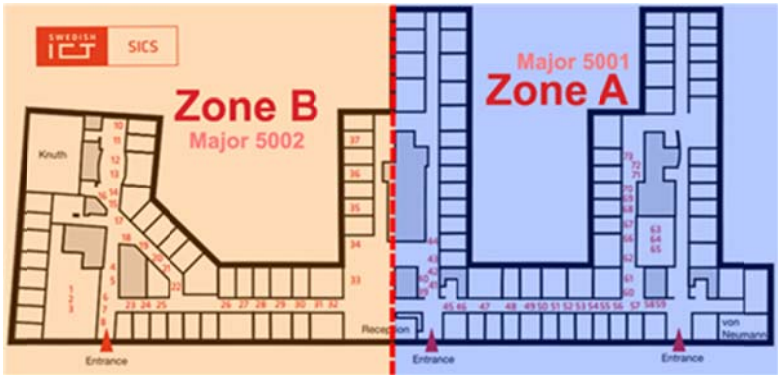


Figure 4-6: Estimote Beacon Zones in the SICS office

One may wonder, in the context of MOSES/ICN, how content is disseminated in a scenario such as the SICS office scenario as implemented in the GUI of this application? If this application were to be further developed, the two brown filled circles shown in Figure 4-7 would indicate the availability of a HTTP NetInf-Server. This would allow users in the vicinity of these NetInf-Servers, to perform infrastructure synchronization over their device's Wi-Fi interface. Assuming that the range of these Wi-Fi access points were set to be very short, then the mobile users would most of the time, opportunistically synchronize with other NetInf enabled Android devices over Bluetooth in *ad hoc* mode. The idea behind the scenario of this application was mainly to emulate a real life scenario where these types of MOSES applications are used. After all, MOSES is attempting to focus on opportunistic *ad hoc* communication, rather than communication through the infrastructure (i.e. offloading traffic from the content servers). Moreover, the infrastructure is mainly used for content dissemination to the Internet. This type of scenario attempts to be a base for larger outdoor scenarios such as a "National Park" scenario, where a user may have other neighboring users in a radius of few kilometers and come across relatively few (relative to the size of the park) short-ranged access points (i.e. local Wi-Fi network).

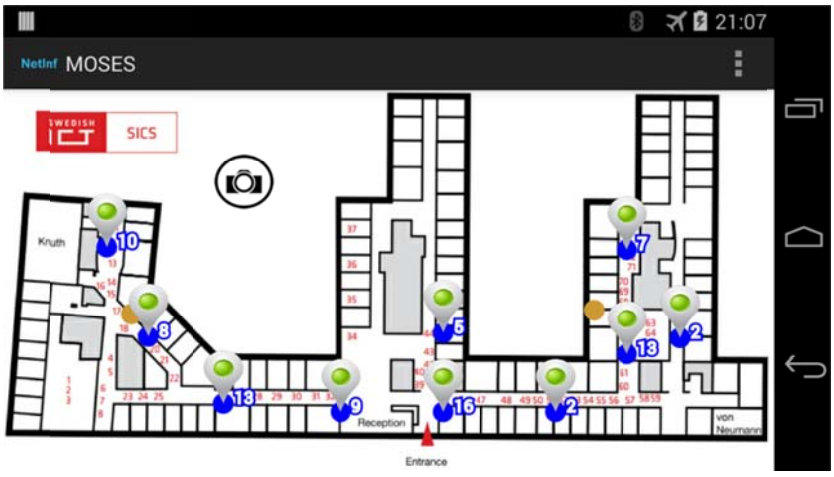


Figure 4-7: Synchronization points

4.5 Logging

A logging system was implemented to support debugging and troubleshooting. The application utilizes Comma Separated Values (CSV) format for logging data. As illustrated in Figure 4-8 the application logs the following fields: *timestamp, local device's Bluetooth name, message type, remote device's Bluetooth name, message ID, convergence layer, or the function executing the line.*

```
27-March-2014 12:37:29.005,TPA-4,NEW INSTANCE
27-March-2014
12:37:29.311,TPA-4,SEARCH,TPA-1,MOSVAbAsFddVMYDgzxeT,BT
27-March-2014 12:37:29.452,TPA-4,SEARCH-
RESPONSE,TPA-1,MOSVAbAsFddVMYDgzxeT,BT
27-March-2014 12:37:29.510,TPA-4,GET,TPA-1,ni:///sha-256;nCD-
glKpb5jAdgh45Z46-f8SgeW_ET9CvTbOLToItjY,ENiMTtcbIIOYrllACsah,BT
27-March-2014 12:37:30.161,TPA-4,GET-
RESPONSE,TPA-1,ni:///sha-256;nCD-glKpb5jAdgh45Z46-
f8SgeW_ET9CvTbOLToItjY,ENiMTtcbIIOYrllACsah,BT
27-March-2014
12:37:30.233,TPA-4,GET,TPA-1,ni:///sha-256;8HsjCvznh3n8rLxN-9tayL
4kV2KAnnkrD92JP5_6Yhg,mDnonPf7sRgTM3j2617J,BT
27-March-2014 12:37:30.740,TPA-4,GET-
RESPONSE,TPA-1,ni:///sha-256;8HsjCvznh3n8rLxN-9tayL4kV2KAnnkrD92J
P5_6Yhg,mDnonPf7sRgTM3j2617J,BT
27-March-2014
12:37:30.796,TPA-4,GET,TPA-1,ni:///sha-256;XAJhWK84s3I50AEWItbC7v
sntWbhZDkUbOJL9VQgqK0,CUEmy2wKc7fJ24rhjMwB,BT
27-March-2014 12:37:31.058,TPA-4,GET-
RESPONSE,TPA-1,ni:///sha-256;XAJhWK84s3I50AEWItbC7vsntWbhZDkUbOJL
9VQgqK0,CUEmy2wKc7fJ24rhjMwB,BT
27-March-2014 12:37:44.079,TPA-4,iBEACON,1002
27-March-2014 12:42:45.600,TPA-4,PUBLISH,{"Time":"12:42:45
pm","iBeaconMajor":"5001","iBeaconID":"b9407f30-f5f8-466e-
aff9-25556b57fe6d","charset":"UTF-8","iBeaconMinor":"1002","Title
":"vid tagen","meta":"publish","ct":"img"},MAIN
27-March-2014 12:46:12.589,TPA-4,SEARCH-
RESPONSE,API,ViIkx8plRSakGZuE3vg9,BT
```

Figure 4-8: Example of an application log

Depending on the message type, the value of the fields may change (e.g. every time the application starts/restarts there will be a line logged saying "NEW INSTANCE" or when a new image is captured and published locally then the PUBLISH line will be followed by the metadata). There may be times where the line is tagged with “API”, such as the last line in Figure 4-8, this indicates that a remote device has caused the function to be called (e.g. in this case the device is replying to a SEARCH message with a SEARCH-RESPONSE).

Every now and then, there will be a line logged with message type “iBEACON”, which contains the major and minor values of the closest Estimote Beacon that the device is sensing. For troubleshooting purposes, this information is very helpful as it enables us to know the user's position at the moment of the event. This information also helps to recreate a scenario and know how the user was moving around.

The results of the controlled experiments presented in Chapter 5 are based on this logging system. Therefore, any necessary or unnecessary information that may be of interest after a conducted experiment has been included in these logs. While extensive logging may cause delay in the performance of the application for experimental reasons these logs are an essential tool for analysis and evaluation of the implementation itself and the protocol in general.

5 Analysis

After design and implementation of the prototype, we observed the behavior of the MOSES/ICN concept and the NetInf protocol, by conducting a series of controlled experiments. The upcoming subsections will describe the three controlled experiments conducted at the SICS office. We analyze the results of each of these experiments using a set of individual metrics (described in sections 5.1.4, 5.2.9, and 5.3.1).

The very first experiment was an uncontrolled experiment during an Open House event hosted by SICS. During this experiment five cell phones running an instance of the application were given to random users attending the Open House. These users voluntarily participated in this experiment and had no prior knowledge of ICN or MOSES. These participants went around the SICS office and captured images (creating NDOs) of objects that were interesting to them. When users were in (Bluetooth) range of each other, synchronization would start via *ad hoc* mode to disseminate all the NDOs. This early experiment mainly assisted in pointing out flaws in the implementation and a lack of information in the logs, thus contributing knowledge for the controlled experiments presented in this thesis.

Three controlled experiments were conducted in the SICS Kista office under the guidance of MOSES researchers. These experiments were designed to collect data to allow an analysis and evaluation of the MOSES/ICN concept using the prototype. The main reasons for this analysis were to identify key factors of efficiency or inefficiency in the MOSES/ICN concept and the Bluetooth CL. Since MOSES inherits the NetInf protocol from the ICN concept, factors of efficiency or inefficiency may raise questions about the architecture of both concepts. The basic idea behind the scenarios of these controlled experiments was to analyze basic MOSES scenarios where users opportunistically meet in a location and without any Internet search; they receive location relevant content from each other.

We perform a series of experiments that are incrementally more complex. The first controlled experiment starts by analyzing a simple point-to-point scenario with only two users. The second controlled experiment will complicate the scenario a bit more by involving 3 more users, thus a total of 5 devices, but in order to keep it simple, the users communicate with each other in a multi-hop manner (i.e. all of the users cannot directly hear each other). In the last controlled experiment, all 5 users may *almost* sense each other (i.e. a semi-mesh topology). These configurations are illustrated in Figure 5-1.

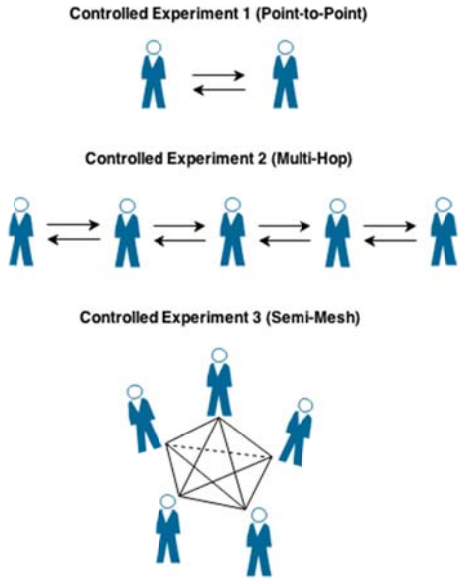


Figure 5-1: Controlled Experiments

5.1 Controlled Experiment 1: Point-to-Point

In this section, we will describe controlled experiment 1 in terms of the experimental conditions, scenario, synchronization process, the selected metrics, and results. Following this is a discussion of this experiment.

5.1.1 Experiment Conditions

This experiment used two Moto G handsets. Each of these handsets was running Android version 4.4.2, with the 176.44.1 Falcon system. Each of the handsets batteries were at more than 80% charge (as a low remaining battery charge has been observed to contribute to higher transmission time via Bluetooth. This transmission property has not been confirmed by any Android documentation or the device's manuals, but has only been observed in the logs from our experiments.)

The screens are turned on during the whole experiment. We chose this configuration as we observed in the logs that some threads go to sleep when the screen is off. We do not yet know the reason for this occurrence nor has any Android documentation or the device's manuals confirmed it.

Version 0.9 of NetInf application/library code was used together with the Bluetooth CL. The network was configured to operate in *ad hoc* mode. The Bluetooth devices only paired with each other.

There were 10 iterations per position leading to $10 \cdot 4 = 40$ iterations in total. Each iteration took around 30 seconds and the experiment was conducted at 5 predefined positions (one phone was located at a fixed stationary location). The total duration of the experiment was $30\,seconds \cdot 10 \cdot 4 \cong$ 20 minutes.

Fifteen unique NDOs were created per device. The NDO sizes ranged from 20 KB to 45 KB with an average size of around 29 KB. Synchronization was invoked manually.

Generally, mobile devices use Bluetooth class 2 radios which operate with a range of between 10 to 100 meters. However, this range is limited by several factors, such as interference from other devices operating in the 2.4 GHz ISM band and large physical obstructions, such as thick walls. The offices at SICS have very large glass-walls/windows, which contribute to better Bluetooth signal reception in the offices.

During the experiment, there was no direct line of sight propagation between the mobile devices and there were other wireless devices present in the experimental area operating in 2.4 GHz. These other devices may negatively affect the overall Bluetooth range and transmission rate.

5.1.2 Scenario

In this experiment, two mobile devices (named Device-1 and Device-2) wish to synchronize NDOs with each other at each of 5 predefined positions. The purpose was to emulate a typical scenario, which might arise during a mobile opportunistic situation, when nearby users exchange location-relevant content over a short-ranged wireless CL in an *ad hoc* network without accessing the Internet.

In addition to the smoothness of the GUI, what other factors contribute to a good user experience? One factor may be rapid exchange of information, thus low delay and exchanging as many NDO as possible is desirable. While the algorithms of the application might be improved, the main goal of this scenario and experiment was to measure the transmission-delay at different distances and with other possible impairments (such as closed doors and walls) when using the Bluetooth CL.

Each test was repeated 10 times, in order to see if there is some deviation in the results and perhaps to identify anomalies. The results of these tests did not show high variation; therefore, the tests were concluded after 10 iterations. This experiment could be reproduced by someone (especially MOSES researchers) who is interested in more detailed results.

This experiment took place in the SICS Kista office near the Von Neumann conference room (marked with a red circle in Figure 5-2). The experiment was conducted at five predefined positions (marked A, B, C, D, and E in Figure 5-2). Table 5-1 summarizes how each device was positioned at each location and what that position might represent in a real life scenario. All the controlled experiments in

this thesis were conducted in the same location and at the same predefined positions. The reason for doing so was to keep most of the experiment parameters the same, while only changing the number of devices and their connectivity in the different experiments.

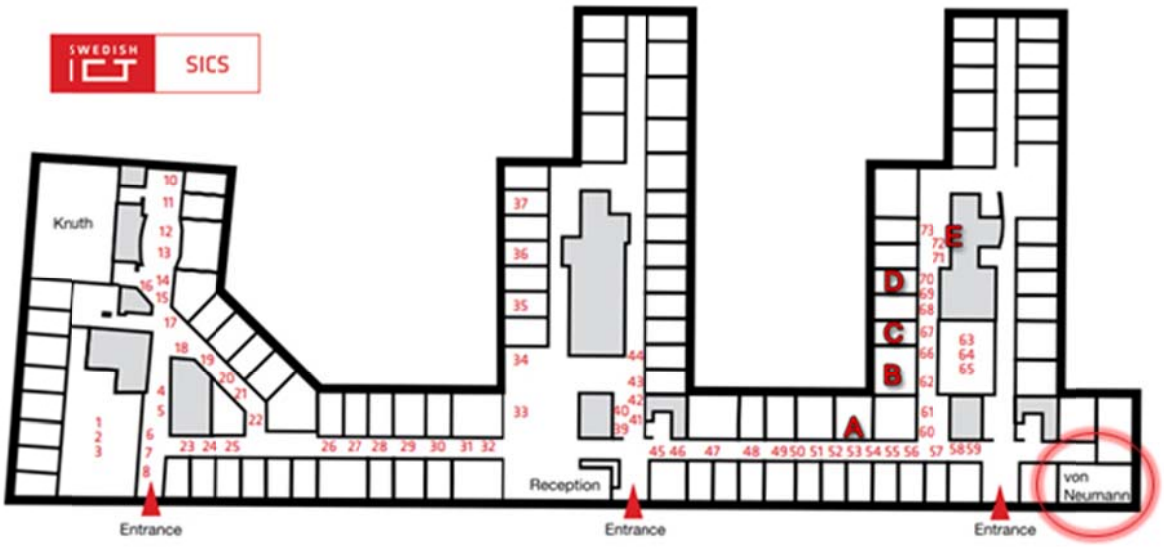


Figure 5-2: SICS Kista Office

One thing that should be noted in Table 5-1 is that unlike other locations, location A's communication path to other devices is around a corner and A's location is on a corridor that is a main walkway for SICS personnel. These attributes sometimes lead to lower transmission quality compared to other locations. This was intentionally done in order that the edge devices (A and E) would have a harder time reaching each other.

Table 5-1: Summary of devices locations

Location	Position	~Height above the floor (cm)	Real-life example
A	Outside the office, on the entrance-window's counter.	73	A person standing there.
B	Inside the office, on the entrance-window's counter, with the door open.	135	Someone having his/her phone on a drawer near the window with the door open.
C	Inside the office, on the entrance-window's counter, with the door closed.	73	A user's mobile device being placed in the office by the window with the door closed.
D	Inside the office, on a drawer, middle right corner of the room (from the entrance point of view), with the door closed.	60	Someone leaving his/her phone on a table in middle of the room with the door closed.
E	Outside the office, on a wooden open cabinet.	77	A person standing or sitting in that position.

This experiment intends to measure various metrics and observe point-to-point communication over a Bluetooth CL with an ICN-based protocol, specifically NetInf. Before starting with more complex scenarios where several devices are involved, we began with two devices and then incrementally added more devices in the subsequent controlled experiments.

Device-1 has 15 unique NDOs in its local cache. These NDOs will be fetched by Device-2 during its synchronization attempts. In order to reset Device-2 after each test, the user(s) manually reset their phone to its initial-state by choosing the "Recover State" item from the menu bar in the application. This function removes any previously fetched NDOs.

The application displays a message with the total number of NDOs in the local cache after the synchronization process is complete. Once Device-2 has synchronized, it was manually reset to its initial-state and the test re-initiated.

5.1.3 Synchronization Process

As explained earlier, in this controlled experiment one device (Device-2), seeks to synchronize NDOs with another device (Device-1), using the NetInf protocol over the Bluetooth CL. How does this synchronization process work? In this scenario, Device-1 is stationed statically at location "A" (shown in Figure 5-2) throughout the experiment. In contrast, Device-2 is placed in four different positions. At each location, Device-2 initiates the Sync process. This scenario is intended to observe the impact of distance and possible obstructions along the transmission path, therefore one device changes its location relative to the other device that remains in a fixed location.

Figure 5-3 illustrates the Sync process carried out during this experiment. Device-2 always initiates the Sync by sending a SEARCH request to Device-1. If Device-1 is within range, it will reply with a SEARCH-RESPONSE containing the NDOs (not Fullput NDOs) in its cache. Device-2 calculates which NDOs do not exist in its local cache and then perform a GET for each missing NDO. Device-1 replies with a GET-RESPONSE to each GET request containing the full NDO (i.e., a Fullput NDO). After Device-2 receives each GET-RESPONSE, it will PUBLISH the corresponding NDO locally, thus indicating that this NDO is now available from Device-2.

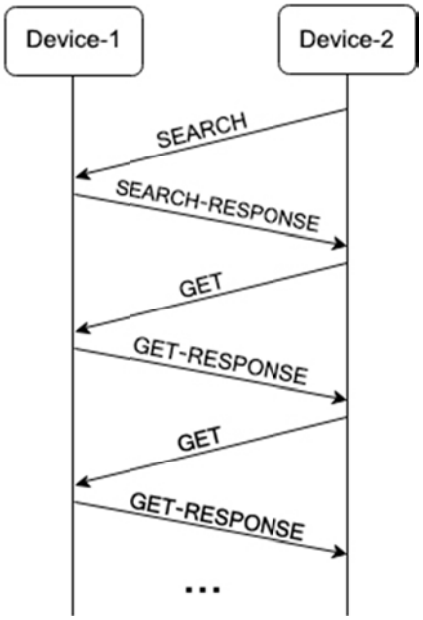


Figure 5-3: Sync Function

In this experiment static routing is used as provided by the NetInf library. This means, Device-1 and Device-2 will only consider each other when creating a Bluetooth socket. This contributes to faster convergence (the network has converged when Device-2 has received all Device-1's NDOs). Device-2 and Device-1 communicate only with each other and ignore any other device(s) in the vicinity.

5.1.4 Metrics

From the MOSES and ICN points of view, what is the most relevant metric to measure in this scenario? Given the description of the synchronization process, it may be obvious that GETs have a big role in a Sync attempt. In this scenario the delay caused by the distance between the devices is the focus, therefore we measure the time it takes for each GET's roundtrip to the remote device.

The process of exchanging NDOs is a set of processes. The first step is to establish a Bluetooth socket to the remote-device, then send a SEARCH request, based upon the response calculate NDOs that do not exist in the device's local cache, and finally performing a GET for each NDO. Therefore, the

most obvious metric is the time required to complete the Sync process in the point-to-point scenario used in this experiment.

If a device attempts to completely synchronize with the other devices in the network, then for every failed GET, the device must send a new GET or postpone this GET to the next Sync attempt. Thus failing to get a NDO would either affect the Sync duration (due to the repeated GET attempt) or cause an extra Sync iteration – in either case increasing the convergence time. Therefore, another metric for this experiment is the success-rate of GETs.

In summary, the following metrics will be measured and analyzed in this experiment:

GET-RESPONSE Delay — The time between Device-2 sending GET until the corresponding GET-RESPONSE is received by Device-2 from Device-1. This delay includes the transmission time over Bluetooth CL plus the processing at the two end-devices.

GET Success-rate — Fraction of successful GETs at the initiating device. A GET is considered successful if a Fullput GET-RESPONSE is completely received.

Sync Duration — The duration for one Sync iteration initiated from Device-2 to Device-1. Each Sync iteration consists of a complete synchronization, i.e., a successful SEARCH-RESPONSE is received and all corresponding GETs are sent independent of whether each GET succeeded or not.

5.1.5 Results

This section presents the results of this experiment. Each of the three metrics is analyzed and described in the following sections.

5.1.5.1 GET-RESPONSE delay per location (First Metric)

Here the GET-RESPONSE delay for Device-2 will be analysed for each location. In general, obstacles in its path and increased distance will reduce the bit rate of a wireless channel. Here we compare the GET-RESPONSE delays for alternatives B→A, C→A, D→A, and E→A. This comparison will be useful later when we consider 5 users in an *ad hoc* network all trying to exchange NDOs. The 5 different positions enable us to evaluate the situation for the edge devices (those devices with only one device near them) and how these edge devices may participate in *ad hoc* networks.

Figure 5-4 presents the GET-RESPONSE delay for Device-2 in location B when fetching NDOs from Device-1 in location A. In each test-iteration Device-2 attempts to fetch Device-1's 15 NDOs. Figure 5-4 shows the duration for each individual GET→GET-RESPONSE, the average GET-RESPONSE delay for each test-iteration, and the GET-RESPONSE delay averaged over all 10 iterations.

As is apparent in Figure 5-4 during a test-iteration, the majority of the GET→GET-RESPONSE durations exhibit values close to each other with only a few outliers. These outliers (e.g. experiment-iteration 1, 8th bar in Figure 5-4) are not necessarily part of a pattern of delay/failure since they present single occurrences that occur from time to time. In contrast, the test-iteration in Figure 5-4 exhibits a higher delay than the other nine test-iterations. The overall median of the GET-RESPONSE delays for this experiment is 232 milliseconds, while the average GET-RESPONSE delay for test-iteration 8 is 481 milliseconds. These kinds of outliers are expected when measuring this metric in this experiment, since different factors may probabilistically and momentarily affect the data transmission. Factors such as interference/path loss in the wireless channel and momentary disturbances in the transmitting/receiving device can occur during one test-iteration which spans ~30 seconds.

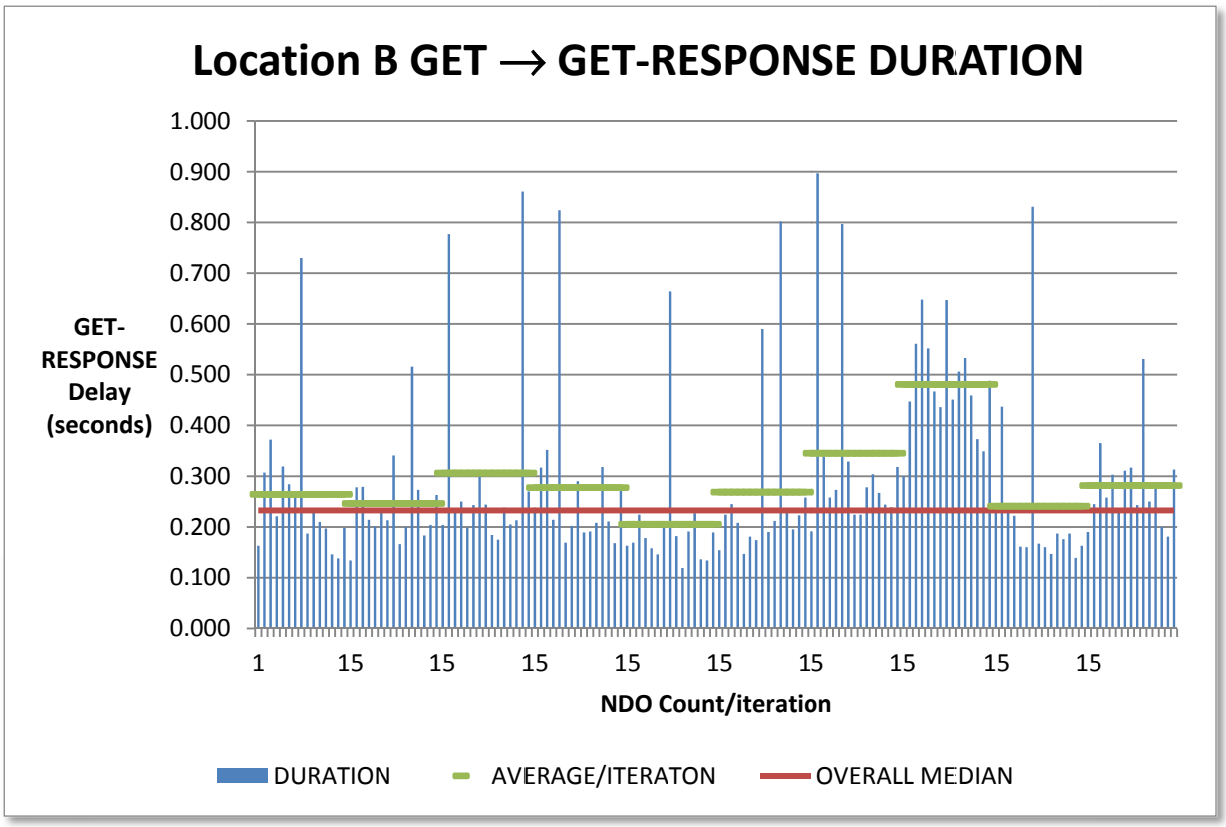


Figure 5-4: **Location B GET-RESPONSE delay**

Figure 5-5 present the results for the same scenario, but this time Device-2 is at location C when attempting to synchronize with Device-1 at location A. The overall average GET-RESPONSE delay for location C was 290 milliseconds, which is a decrease in the overall average GET-RESPONSE delay compared to location B which had an overall average of 292 milliseconds. This might imply that the roughly 5-7 meters increase in distance introduced by location C did **not** affect the overall bit rate during these two sets of tests.

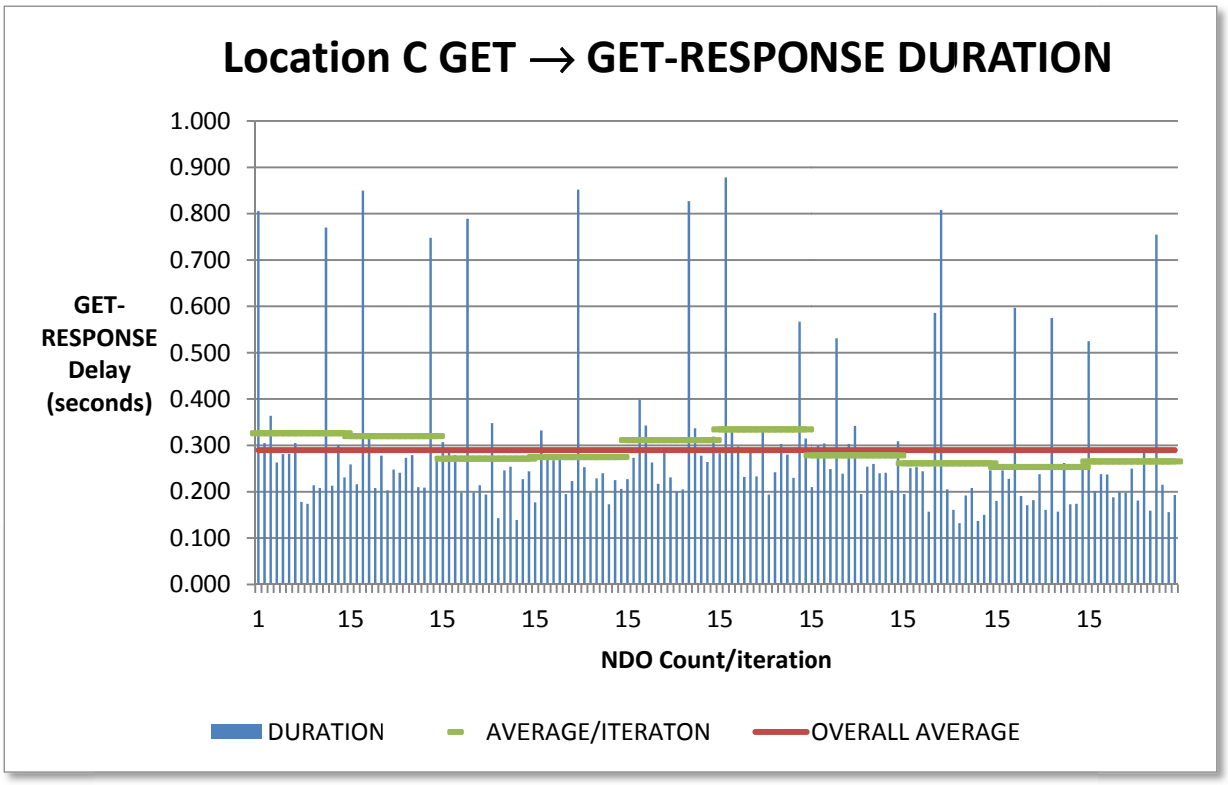


Figure 5-5: Location C GET-RESPONSE delay

The greater the distance (to locations C and D) and the more obstructions (location D is in the middle of the office and distant from the entrance-window) introduced between the two devices, the greater the number of unsuccessful Bluetooth socket establishments between the two devices. This is illustrated in Figure 5-6, where in 8 out of 10 test-iterations, Device-2 could not sense Device-1, thus Device-2 could **not** synchronize with Device-1. In the case of D→A, 80% of the Sync attempts failed. However, when Device-2 *could* sense Device-1 and successfully *established* a Bluetooth socket, then the transfers completed without any interruptions. In this case, Device-2 could fetch all of Device-1's 15 NDOs *without* any GET failures. This seems to indicate that once Device-2 has successfully established a Bluetooth socket to Device-1, the GET success-rate was 100%. Location D showed a ~227 milliseconds increase in the overall average GET-RESPONSE delay compared to locations B and C. This increased delay may be due to the increase in distance or other factors affecting the data transmission.

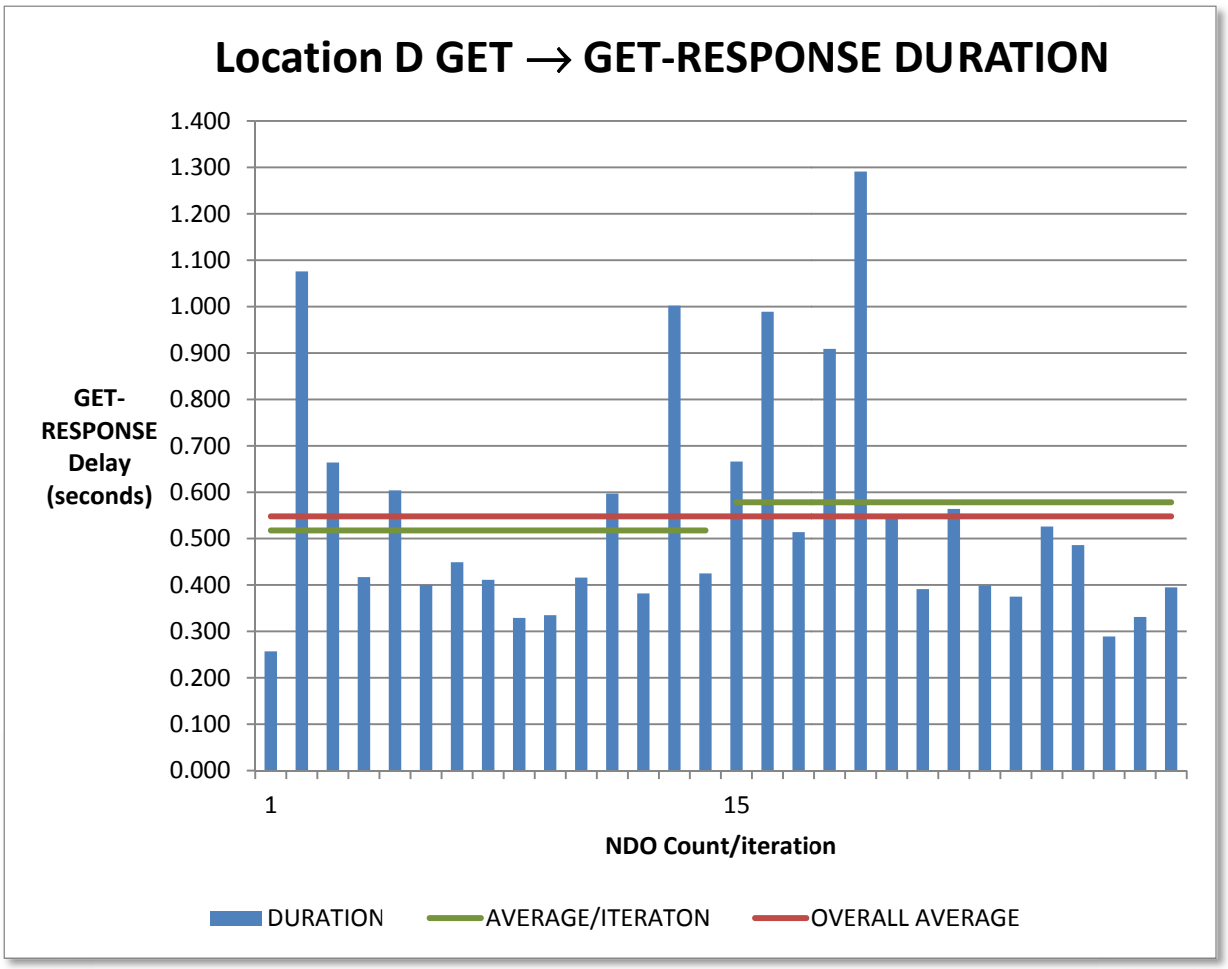


Figure 5-6: Location D GET-RESPONSE delay

After the tests at location D, Device-2 was moved to location E for the last tests in this first experiment. Location E is an open wooden cabinet near the corridor (this corridor is shared by locations B to E). This location was the most distant relative to Device-1's location. In this location all 10 iterations failed as the Bluetooth transceiver failed to establish a link connection, thus Device-2 could not sense Device-1. This may have been expected after the results at location D. In general the further the two devices are from each other the worse the radio signal strength they are expected to have (up to the point at which they are out of range and cannot establish any link connectivity). This decreased radio signal strength should negatively affect their convergence time and introduce increased delay in the synchronization process.

5.1.5.2 GET success-rate and Sync Duration (Second and Third Metrics)

After looking at the individual GET-RESPONSE delays for each location, we look at the overall Sync duration in a simple point-to-point scenario. Table 5-2 shows different factors that might be of interest when analyzing the Sync duration for the case of a single device. As expected, the average Sync duration increases with distance. The average GET-RESPONSE delay as shown in Table 5-2 has a big role in the Sync duration, as the GET function is the main operation within a Sync process. However, as mentioned earlier, in the point-to-point Bluetooth connection used with this application, once a device establishes a Bluetooth socket with a remote device, it is highly probable that it will have a GET success-rate of 100%, meaning all the 15 NDOs are fetched without any GET failures. This behaviour was observed during the 22 successful iterations of this experiment.

Table 5-2: Experiment 1 results

Position	Average Sync duration (seconds)	Sync Standard deviation (seconds)	Average GET-RESPONSE delay (ms)	Average SEARCH-RESPONSE delay (ms)	Number of failed Sync iterations	NDO Success-rate/iteration
B	8.257	1.154	292	422	0	100%
C	8.393	0.987	290	319	0	100%
D	15.447	0.815	548	331	8	100%
E	∞	∞	∞	∞	10	0%

The SEARCH-RESPONSE delay may not have a great impact on the overall Sync duration since it is a onetime operation for each Sync instance. In contrast, since the (up to) fifteen GETs are sequential operations; the amount of time that each one takes has a greater impact on the Sync duration.

Figure 5-7 summarizes the Sync duration results. At locations B and C in Figure 5-7 there are 10 overlapping Sync durations. This figure shows how the Sync durations are distributed for the three locations where synchronization was possible.

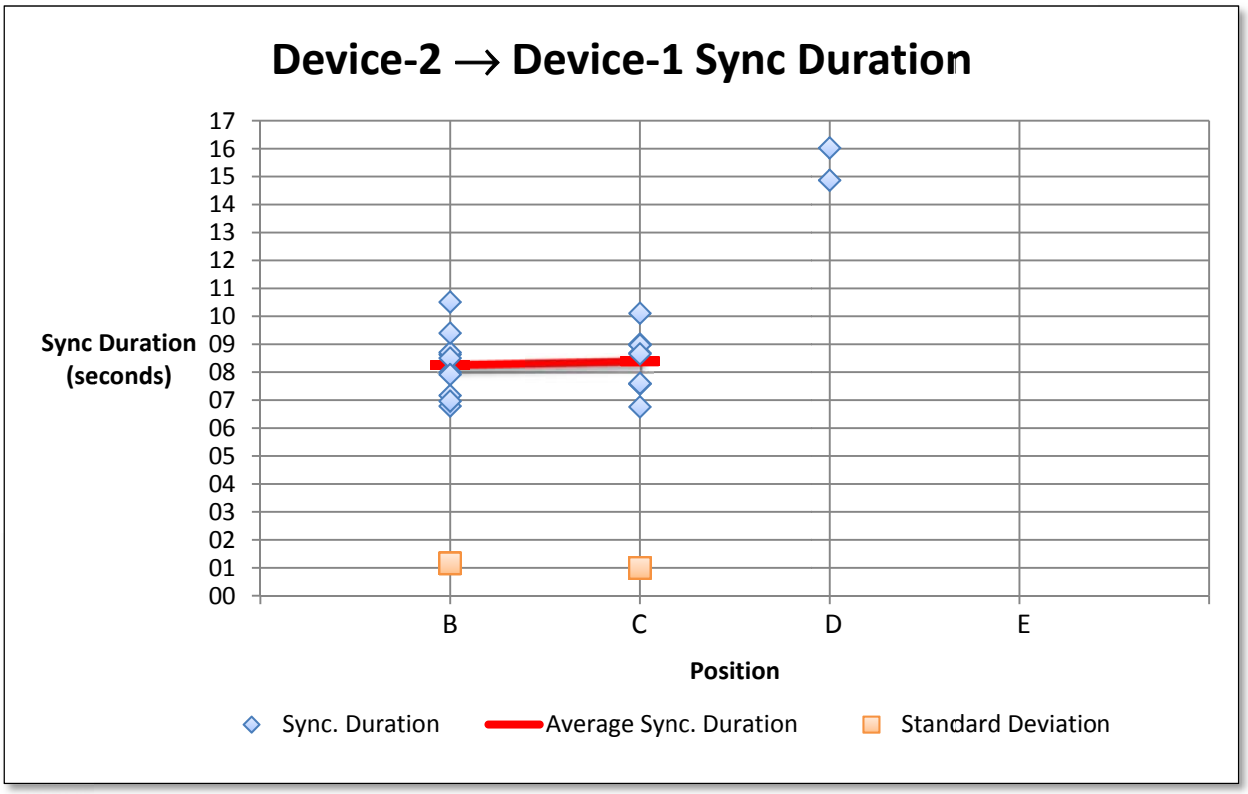


Figure 5-7: Experiment 1 Sync Durations

5.1.6 Discussion

What was learned from this experiment? This experiment revealed the performance of the fundamental functions required for MOSES/ICN opportunistic sharing applications. The key point of this analysis was the factors that contribute to delay in the convergence time of an ICN application in the simple case of a point-to-point link. In addition to the processing delays introduced by the two end-devices, when designing such an application it may be important to take the metrics examined in this experiment into consideration.

5.2 Controlled Experiment 2: Multi-Hop

After experimenting with the behavior of an ICN-based network in a simple point-to-point network, the second scenario adds a bit of additional complexity by adding 3 more devices to the network. In order to keep the scenario simple, the *ad hoc* network will behave in a simple multi-hop manner, thus each mobile device will only directly communicate with its (one or two) neighboring devices. Compared with the scenario of experiment 1, this scenario allows observation of 2-3 devices exchanging NDOs in an ICN-based network. In addition to the increased number of devices in this scenario, the multi-hop behavior gives an opportunity to observe the efficiency of content dissemination in such an ICN-based *ad hoc* network. This should reveal if opportunistic content sharing occurs in an efficient and scalable manner. Establishing whether these two properties are achieved is one of the general goals of MOSES and ICN.

5.2.1 Experimental conditions

The conditions for this experiment are the same as in experiment 1, with only a few changes. There were 5 devices. As before each test was repeated 10 times, with each iteration taking around 15 minutes, thus the total experiment was expected to last $10 \cdot 15$ *minutes* $\cong$ 2 hours and 30 minutes. The devices were configured to periodically synchronize at most every 40 seconds.

The Bluetooth devices were manually paired in a multi-hop order. This was accomplished by putting 4 bonded devices in each device's Bluetooth bonded list: 1-2 real devices and 2-3 dummy paired devices. The dummy paired devices emulate a "real" multi-hop scenario. When the user's device is paired with 4 devices (but only 2 of them are within range). This configuration allows this experiment to measure the delay caused by a non-responsive (e.g. not in range) device.

5.2.2 Scenario

Five mobile phones named Device-1, Device-2, Device-3, Device-4, and Device-5 are at locations A to E (see Figure 5-8) with each device having 15 unique NDOs in its local cache. As in the previous experiment, the devices are manually reset to their initial-state after each test and once the network has converged the devices will be manually reset to their initial-state and then the test will be repeated for a total of 10 iterations.

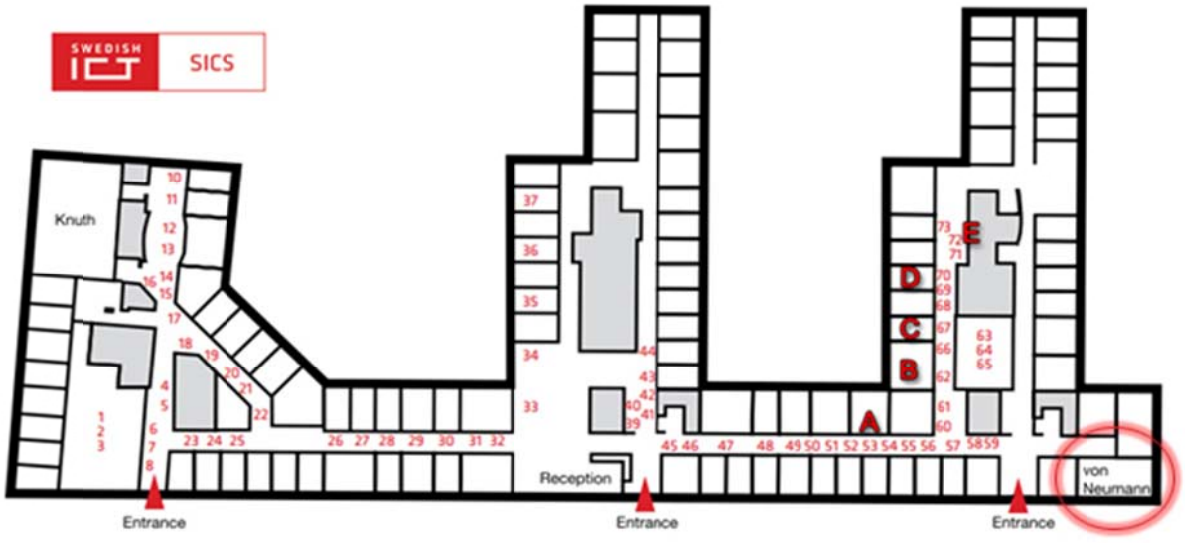


Figure 5-8: SICS Kista Office (Same as Figure 5-2)

As described above, all five devices were manually paired by Bluetooth (according to Table 5-3) as shown in Figure 5-9 to create a linear multi-hop network. Additionally, there were extra dummy devices paired with each device in order to have four bonded devices for each device. These dummy bonded devices allow this experiment to emulate a real life multi-hop scenario when similar delays are among other factors that are expected to be caused both by each hop (dissemination) and

non-responsive (e.g. out of range) devices. This experiment attempts to investigate the behaviour of ICN-based opportunistic networking in a multi-hop manner, e.g. in order for Device-1 and Device-5 to converge, all devices in between (i.e. Device-2, Device-3, and Device-4) must converge first.

Table 5-3: Devices pairing

	Device-1	Device-2	Device-3	Device-4	Device-5
Paired with	Device-2	Device-1	Device-2	Device-3	Device-4
	Dummy Device	Device-3	Device-4	Device-5	Dummy Device
	Dummy Device	Dummy Device	Dummy Device	Dummy Device	Dummy Device
	Dummy Device	Dummy Device	Dummy Device	Dummy Device	Dummy Device

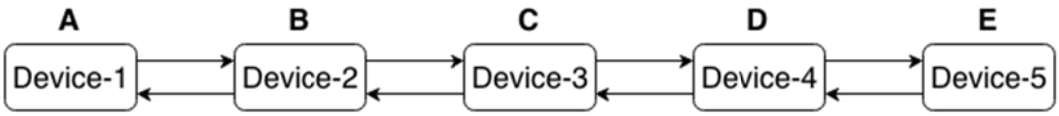


Figure 5-9: Mobile devices organized in a linear topology

5.2.3 Synchronization Process

The synchronization process is similar to that in experiment 1. However, instead of two devices, five devices will synchronize with each other in a multi-hop manner. Unlike experiment 1, where the Sync process was executed manually, in this experiment each Sync process is executed automatically by a periodic timer, as described in Section 5.2.6.

The synchronization process performed in this experiment follows the flow chart shown in Figure 5-10. This flow chart gives an overview of the process and illustrates the main components of the synchronization process performed by the application.

In addition to the SEARCH, GET, and the periodic timer (each of which will be discussed in the upcoming sections), an important state in this flow chart is closing the connection to Bluetooth server at the start of the Sync process. In order to keep this experiment as simple as possible and measure only the metrics of interest, we have turned off all other services that may affect the results of this and the third experiments. As the focus of this experiment is only the Sync function, all of the other services (such as the indoor localization service and the Bluetooth beacon discovery service) of this application are disabled before the start of the experiment.

During the implementation of this application, it was observed that the Android Bluetooth service has a limitation of one client per Universally Unique Identifier (UUID). Further details of this are discussed in section 5.2.6. This issue was observed to cause high processing delays and timeouts in a bi-directional Sync attempt, therefore, a decision was made to have each device perform one activity at a time. By shutting down the Bluetooth-Server at the beginning of a Sync no remote-devices may connect to the device, but the device itself may connect to other devices if they are not performing their own Sync (since if they were performing a Sync they would also have shut off their Bluetooth-Server).

To ensure that each device only handles one activity at a time, if a device receives a connection from a client, it brings down its Bluetooth-Server (as recommended by the Android documentation[20]) and only serves that client. This client should release its connection to the serving device after it has been served, thus each client closes its connection *after* it has completed its Sync to allow the remote-device to initiate its own Sync. As a result, a serving device will *not* initiate its own Sync *until* the client has completed its Sync and released its connection.

The Android Bluetooth API (or the mobile device itself) was observed to misbehave during implementation tests by not releasing a remote connection after the completion of a Sync process. Therefore, the application implements a 30 seconds timeout after which the serving device shutdowns a client's connection if the client has not already done so.

The SEARCH and GET timeouts are shown in Figure 5-10. These timeouts are discussed in Sections 5.2.4 and 5.2.5 where the roles of SEARCH and GET in the Sync process are discussed.

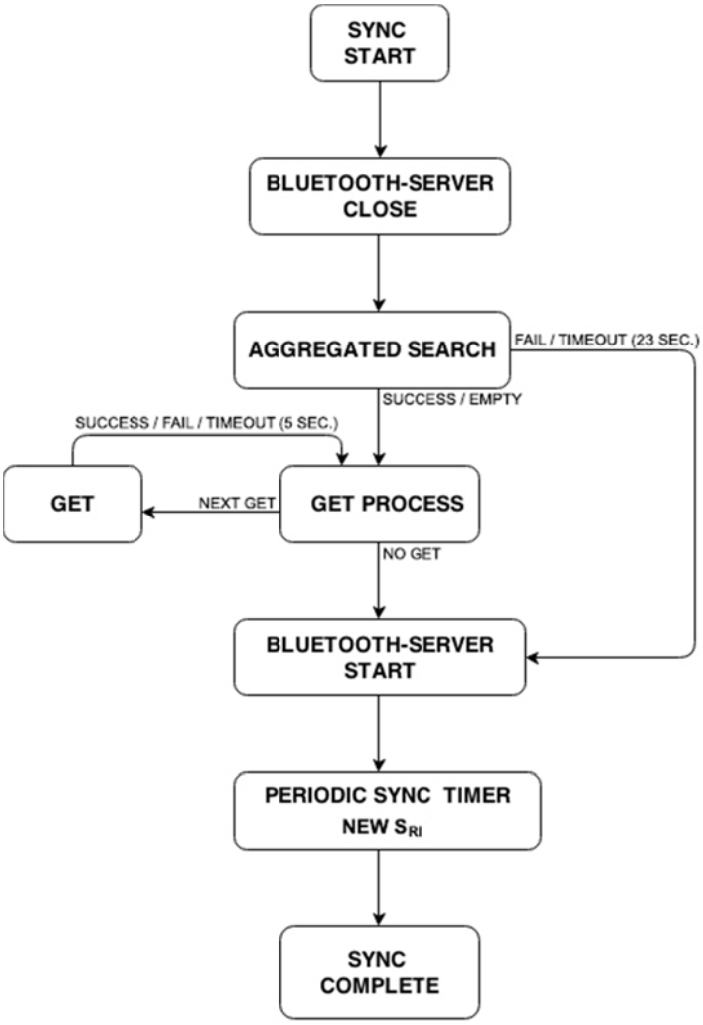


Figure 5-10: Sync. Process overview flow chart (Only overview, not the complete algorithm)

5.2.4 NetInf library aggregated SEARCH mechanism

The NetInf library uses an aggregated SEARCH mechanism. Figure 5-11 shows Device-3 performing an aggregated SEARCH. This mechanism sends a SEARCH to all paired devices that are available. The order of sending SEARCH requests is random so a SEARCH may be sent to more distant devices before nearby devices. In this scenario since each device senses only its neighbouring devices SEARCH requests are sent in a multi-hop manner. Duplicate NDOs in the SEARCH-RESPONSE from different devices are removed before returning the aggregated SEARCH-RESPONSE to the next step in the Sync process.

The SEARCH function of the Android NetInf library is used in the Sync process as was shown in the flow chart in Figure 5-10. There is a SEARCH timeout in the application, to allow the Sync process to proceed even if no SEARCH-RESPONSE is received from any remote device. During tests of the application, it was observed in the logs that it takes up to 5 seconds for each SEARCH-RESPONSE. SEARCH-RESPONSE delays have been observed to be much less than 5 seconds, but considering the highest transmission delay case 5 seconds is considered a reasonable timeout.

In this and the next experiment each device may have up to 4 neighbouring devices, therefore considering 4 devices each with a five seconds timeout, implies that a 20 second timeout would be required for an aggregated SEARCH-RESPONSE. The SEARCH function is an essential component of a Sync process. As shown in Figure 5-10, if an individual GET fails, the GET process proceeds to the

next GET; but if an aggregated SEARCH fails then the Sync process will terminate. A safety margin of 3 seconds was added, thus the SEARCH timeout implemented in application is 23 seconds.

Figure 5-11 illustrates a case where all devices may sense each other. However, this was **not** the case in this experiment.

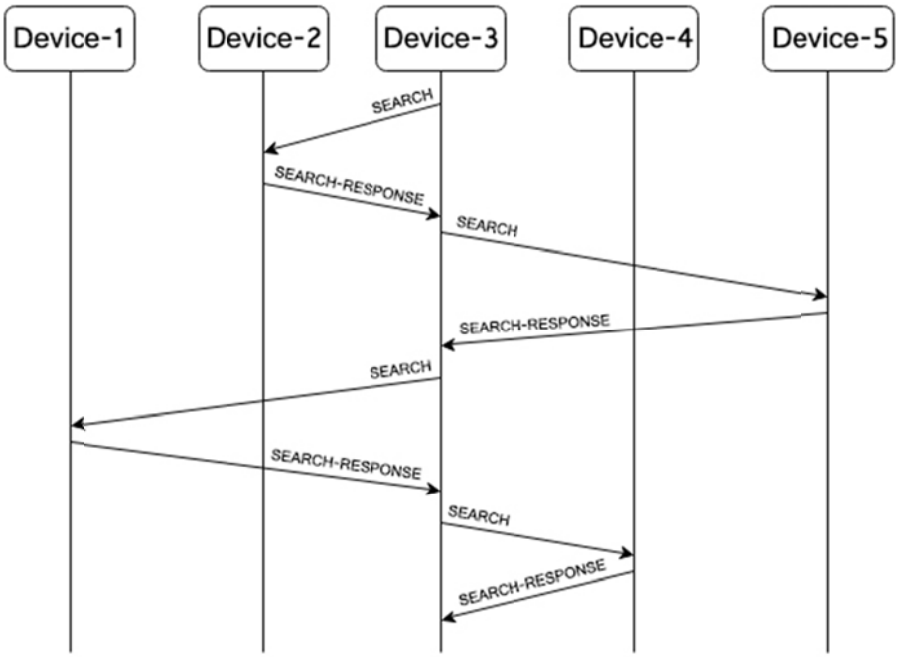


Figure 5-11: NetInf library aggregated SEARCH mechanism (this is *not* the case in this experiment)

5.2.5 NetInf library distributed GET mechanism

After a successful aggregated SEARCH, the Sync process will compute from the aggregated SEARCH-RESPONSE a set of NDOs that do not exist in the user's local cache. The Sync process will perform a GET for each NDO in that set. For each GET, the NetInf library will behave as shown in Figure 5-12 (which shows Device-3 performing one GET). With respect to the ICN/NetInf concept, every GET is randomly and sequentially sent to any available device until a GET-RESPONSE with the Fullput NDO is received. In Figure 5-12 Device-3 sends a GET request randomly to Device-1 and Device-1 responds with a non-Fullput GET-RESPONSE meaning Device-1 did not have the respective NDO in its local cache, therefore Device-3 continues to send requests to randomly selected devices until Device-4 responds with a Fullput GET-RESPONSE. Now Device-3 stops sending GETs (e.g. in this case Device-3 sent a GET request to all its neighbouring devices except for Device-2) and instead does a local PUBLISH of the NDO that it received from Device-4. However as stated earlier, in this scenario both SEARCHs and GETs will follow a multi-hop order since the phones can only communicate each other in this manner.

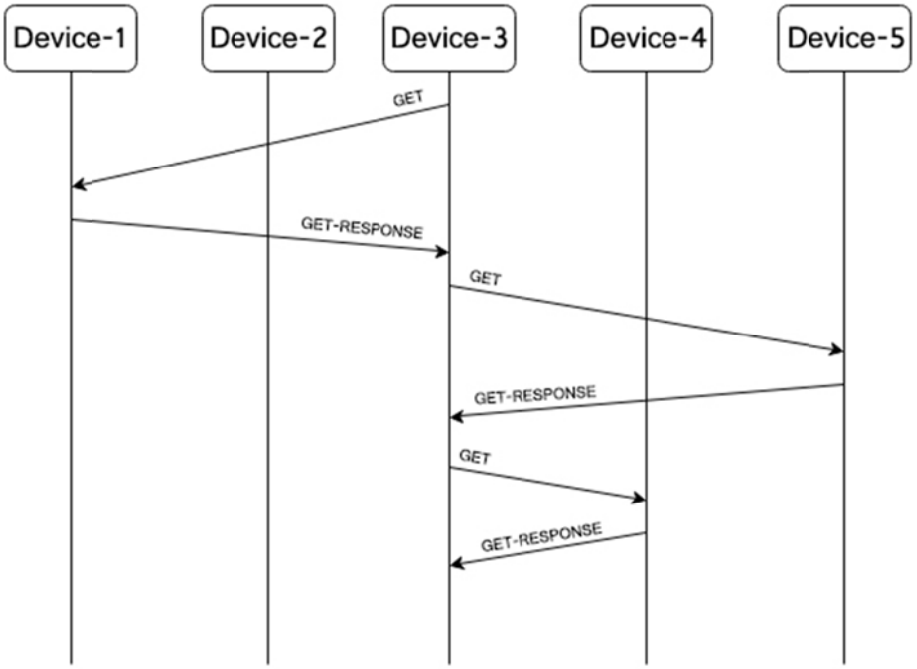


Figure 5-12: NetInf library distributed GET mechanism (this is *not* the case in this experiment)

As was shown in Figure 5-10, the timeout of each individual GET is set to 5 seconds. This value was selected because when performing 60 consecutive GETs during a Sync process, we want the GETs to be processed as quickly as possible. During the test phase of the implementation of this application, it was observed in the logs that depending on the received signal strength of a transmission channel a GET-RESPONSE delay may be up to 5 seconds. For example, a poor received signal strength occurs in the worst case (location D) of experiment 1 (see Section 5.1.5.1). In this case, the GET-RESPONSE delay was 1.291 seconds. Setting too high a GET-RESPONSE timeout may delay the completion of the Sync and setting too low a GET-RESPONSE timeout may affect the success-rate of the GETs, which may cause extra Sync iterations – in both cases negatively affecting the convergence time. Therefore, 5 seconds seems to be a reasonable GET-RESPONSE timeout as we will see when we consider the results of this experiment in Section 5.2.10.2.

5.2.6 Bluetooth CL

As described earlier the NetInf library is being used with the Bluetooth CL. The following subsection describe the special actions that are needed to manage the network sockets with Bluetooth and some limitations of the Android Bluetooth API that necessitated introducing a particular master and slave configuration for use with the Android application.

5.2.6.1 Socket management

A tricky part of this experiment occurs at the Bluetooth transmission level. The NetInf library was modified for this experimental scenario to open a test-socket to all the devices in the user's Bluetooth bonded-devices list and to verify the availability of the paired devices. According to the logs, each test-socket takes between ~1-10 seconds (~12 seconds according to the Android documentation[20]) to timeout if the device does not respond (e.g. it is out of range or unavailable for some other reason).

The NetInf library has a Socket-Manager class that keeps an established socket open until some event tears it down; therefore, the test-socket process is skipped if an open-socket already exists. During a Sync process, a Bluetooth socket may unpredictably be torn down for one of several reasons. One reason may be due to problems at the physical layer, as transmissions on the 2.4GHz wireless channel are subject to interference, path loss, etc. Another reason for Bluetooth socket failure is the Bluetooth master/slave configuration discussed in the next section.

In order to complete the Sync process, the device has to re-establish lost connections to those devices that responded to the initial SEARCH request. To achieve this, the Sync process saves a set of all the Bluetooth devices with whom a socket was successful established after the first successful SEARCH-RESPONSE. Before performing each GET request, the Sync process requests the Socket-Manager to re-establish a socket (if one is not already established) to each of these saved devices. This assumes that all the devices that were available during the initial SEARCH are still within range and therefore the device tries to re-establish a connection with the same set of devices. A complete Sync iteration (explained later in Section 5.2.10.5) is usually very quick and the Bluetooth wireless range (considering line of sight and the fact that the handsets are equipped with Bluetooth class 2 interface) is usually up to ~10 meters (~60 meters outdoors with line of sight, according to the field experiments with the Moto G phones that were used in all the experiments reported in this thesis). It should be very unlikely that a remote-device after responding to a SEARCH-RESPONSE, would disappear completely. However, in the worst case, this does happen, and then all the GETs to that device will fail/timeout. Each of these failures will contribute between ~1-12 seconds (timeout for opening one Bluetooth socket) delay to the Sync iteration. Fortunately, this is an unlikely event when the device(s) is (are) not moving rapidly (e.g., in a vehicle).

5.2.6.2 Master/Slave configuration

The reason why we close the Bluetooth-Server while serving a client or being served by a server is due to various limitations introduced by our choice of the Android Bluetooth API as the CL for this application. In addition to the compulsory pairing in this version of Bluetooth (as we do not use BLE), a master/slave configuration is introduced. The information presented in this section is based on observations during our implementation and testing. Unfortunately, the master/slave configuration of Bluetooth for the Android platform is not described in detail in the Android documentation[20].

Bluetooth[21][22] (i.e. Bluetooth core system) is limited to a maximum of 7 simultaneous active clients per server using different UUIDs. Thus a given Bluetooth-Server (master) may have up to 7 concurrent clients (slaves) forming a small *ad hoc* network, called a Bluetooth "piconet".

The Android NetInf library provided by MOSES uses a single UUID for serving its clients. Generally, a Bluetooth slave (client) uses a UUID to refer to the Bluetooth service on the serving device (master). In terms of Bluetooth connectivity, this means that there is a need for a UUID **per** client→server connection. How would this work in an *ad hoc* network with no protocol for informing each device what UUID each server is listening to? A simple solution is to assign seven different UUIDs to the NetInf protocol and query each device to discover its neighboring devices. However, these additional test queries will add additional delay to the neighbor discovery process. If each attempt at Bluetooth neighbor discovery with a test connection to a neighboring device times out after ~1-12 seconds for each UUID, then if the neighboring device is listening to the 7th UUID in the list an additional ~6-72 seconds delay will be added to the Sync process.

Another reason for closing the Bluetooth-Server is the change in behavior of the Sync process. If a device performing consecutive GETs with large payloads (i.e., a large NDO in terms of its size in bytes) attempts to serve another client with the same NDOs, would the Sync process appear to be smooth to the end user? During the test phase of the implementation, a similar approach was implemented and tested, but an increased number of timeouts and GET failures were observed. Therefore, we decided to limit the devices to doing only one thing at a time.

5.2.7 Periodic Synchronization Timer

In this experiment and the third experiment, the synchronization process is periodically executed by a separate thread based on the following formula $S_{RI} = S_I \cdot R \cdot K$ where S_{RI} is the random interval for synchronizations, which is a product of S_I a Sync interval defined by the user, R a random value with the constraint $0.45 \leq R \leq 1.0$, and K is a value used to control the randomness.

Since this application is to demonstrate opportunistic behavior, the application does not implement a synchronization protocol and nearby devices are unaware of each other's synchronization status. The Sync interval rather than being a static value defined by the user, has a controlled

randomness (based upon the value R). This randomness may prevent neighboring devices from simultaneously initiating the Sync process. If all devices that are within range of each other initiate their Sync process at the same time, then no device gets the chance to become a client of another device since the first step in the Sync process is to shut down the Bluetooth-Server (as explained in Section 5.2.6.2). If the Sync interval were a static value then all of the devices could continuously attempt to synchronize, fail, and then start over until some event breaks the loop. The controlled randomness of the Sync interval S_{RI} should give one of the devices a chance to initiate its Sync process before any of the other devices have started their Sync process.

R has a minimum value of 0.45 as the minimum Sync interval S_I in this experiment is 40 seconds with a K value (which should not be less than 1 in a normal case) of 1 would give S_{RI} of 18 seconds which is a convenient amount of time before a device initiates a Sync under normal condition. In this experiment each device is paired by Bluetooth with a maximum of 4 devices. As explained in Section 5.2.6, a Sync process starts by sensing its neighbors by opening a test-socket to each of the devices in the user's bonded-device list. A Bluetooth connection attempt to an unavailable Bluetooth device will timeout after ~1-10 seconds (and does so according our logs). With a ~5 seconds timeout (a value between 1-10 seconds) for every individual device this leads to a timeout after a total of 15 seconds if the first 3 paired devices in the bonded-devices list are not in range and the fourth device is in range (this is considered the worst-case in this experimental scenario). Allowing an additional 3 seconds for socket-establishment to the fourth device gives a total of 18 seconds that a device would need in the worst case to establish a successful connection to another paired device. Even if we ignore the worst case in the test-socket process, under normal conditions (K=1), having a Sync interval of less than 18 seconds is not recommended. During the test phase of the implementation, it was observed that very short Sync intervals lead to simultaneous Sync attempts between devices.

The K value controls the randomness of S_{RI}. Thus in the case of consecutive GET failures, the Sync process will back-off for a limited amount of time before retrying (back off timers are discussed in the next section). Following such a back-off it may be convenient to retry syncing much earlier, rather than postponing the synchronization to the next S_{RI}. As a result, a device may start its synchronization within 15 seconds, rather than after 48 seconds. However, even with this randomness, since there is no synchronization protocol between the devices, there is still be a risk that devices become stuck in a state where they continuously simultaneously initiating their synchronizations. The back off timers (explained in the next section) and adjustments to the K value are an attempt to break these kinds of loops.

5.2.8 Synchronization back-off timer

As explained in the previous section, there are cases where devices face consecutive SEARCH/GET failures/timeouts. This may be caused due to momentarily poor transmission quality of the wireless channel or a mobile-device's momentarily hardware/software misbehavior. In order to reduce the inefficiency in the Sync procedure, a back-off mechanism was implemented to address these types of events.

The NetInf library implementation[16] will return an empty SEARCH-RESPONSE if the remote-device does not have a copy of the requested NDO(s), thus a SEARCH does *not* fail even if the remote-device does *not* offer any NDOs. If an aggregated SEARCH fails/timeouts, this *may* imply that something went wrong in the transmission/reception of this request to/from the neighboring devices. As a result, immediately retrying a new SEARCH may be inefficient, instead we should give the other devices an opportunity to initiate their Sync process so that they are more likely to have something to exchange on the next Sync attempt. The same logic applies to consecutive GET failures. If in the middle of a Sync process an individual GET request fails or times out, this may indicate an intermittent problem, therefore it may be wise to back-off for a small amount of time before retrying.

There are various ways to implement back-off timers, but in order to keep everything simple, this implementation uses the same S_{RI} formula used for the Sync periodic timer. This where the K variable is used.

As shown in Table 5-4, by adjusting the K value S_{RI} will vary between the minimum and maximum S_{RI} values depending on the random number R generated by the application. The minimum and maximum S_{RI} values presented in Table 5-4 are calculated by inputting the minimum and maximum limits of the R variable together with an S_I of 40 seconds. For example, if $K = 0.5$ and $0.45 \leq R \leq 1.0$ then the minimum S_{RI} would be $40000 \cdot 0.45 \cdot 0.5 = 9000$ ms and the maximum S_{RI} would be $40000 \cdot 1 \cdot 0.5 = 20000$ milliseconds. By using the variable K, the same formula used for the periodic Sync interval can be used to calculate a back-off time. The minimum and maximum S_{RI} values in Table 5-4 are with $S_I = 40000$, thus 40 seconds is the minimum S_I value for this and next experiment. An S_I value of less than 40 seconds is not recommended since with the chosen K values the Sync attempts and reattempts would be too rapid which would increase the probability of simultaneous synchronization between devices.

Table 5-4 shows the K values used in the implementation of this application depend on the device's current state. For synchronization during normal circumstances, i.e. no occurrence of timeouts or failures, it would be convenient for the user to initiate the synchronization close to the S_I interval defined by the user while including some randomness to prevent devices from simultaneous synchronization. If a device has already synchronized with its neighboring devices, then it would be desirable for the network to back-off longer, as a device that is already synchronized with its neighboring devices should give other devices the opportunity to converge as well.

Table 5-4: K values used in the implementation

K Value	Min S_{RI} (ms)	Max S_{RI} (ms)	Condition
0.5	9000	20000	SEARCH timeout, high GET error-rate, Sync failure due to unknown reason, No device in range, SEARCH-Failure
0.7	12600	28000	Active-Client in progress, No device in range, SEARCH-Failure
1.0	18000	40000	(Normal condition boundary), No device in range, SEARCH-Failure
1.2	21600	48000	No device in range, SEARCH-Failure
1.5	27000	60000	Normal Synchronization
2.0	36000	80000	Already Synchronized

If a device attempts to initiate a Sync but there is an "Active-Client in progress", meaning that the device itself is currently serving a client, then it would be inefficient to postpone the Sync to the next usual S_{RI} (by using $K = 1.5$). If a client frequently does not require a long amount of time to receive only one device's NDOs, then a K value of 0.7 would be suitable. The K value of 0.5 is a lower value that is suitable for timeouts, while allowing faster reattempts.

In Table 5-4 there are several conditions that share the same K values. In the case of the error-events "No device in range" and "SEARCH-Failure" the application will back-off using one of the following K values: 0.5, 0.7, 1.0, or 1.2. As discussed in the previous section, if the reattempt occurs too quickly there may be a risk that the devices continue to simultaneously retry synchronizing with each other. Forcing the devices to choose randomly between these four different K values may reduce the risk of simultaneous reattempts.

It may be *unnecessary* to continue retrying after a failed Sync. For example, when the remote device is no longer in range or is unavailable. Therefore, after an unsuccessful Sync the application continues retrying with small K values a maximum of three times and then assigns a K value of 1.5. This K value will delay the Sync to the next periodic interval. This method also assists in breaking loops when several devices consecutively reattempt to synchronize with short back-off durations.

5.2.9 Metrics

From the MOSES perspective, one of the core efficiency factors for content distributing/sharing is the total amount of time it takes for all the devices to synchronize, i.e., the network convergence time. The less time it takes the devices to converge, the better the overall user experience. Therefore, it seems natural in this experiment to measure the time it takes for all five devices to synchronize with each other.

The network convergence time depends on how fast the individual NDOs are retrieved. The current implementation retrieves each NDO via an individual GET request and GET-RESPONSE message, thus the longer it takes to receive the GET-RESPONSE messages, the longer the synchronization time and the longer the overall network convergence time. It is therefore desirable to measure the median time it takes for each device to send a GET request and receive the corresponding GET-RESPONSE.

As mentioned earlier, the prototype application utilizes Bluetooth CL for transport of NDOs and one of the constraints of using Bluetooth is the discovery of neighboring devices and opening sockets to them. The process of opening test-sockets with bonded devices to check their availability takes a considerable amount of time, especially in the case of a multi-hop scenario where each device has at least two dummy bonded devices (as we must wait for each of these dummy devices to timeout). This test-socket process affects the overall network convergence time and it is therefore important to measure and analyze the time required for this process.

An important metric that affects the network convergence time is the number of successful synchronization attempts each device needs to make before they have all 75 NDOs. Each synchronization attempt takes considerable time and makes the attempting device unavailable for the duration of its synchronization process. The more synchronization attempts it takes a device to converge, the longer the network convergence time. Therefore, it is especially interesting to measure the required number of synchronization attempts required to converge in a multi-hop scenario as well as the duration of each synchronization attempt.

In summary, this experiment uses the following metrics:

Network Convergence Time	The time required for all devices to fully synchronize with each other (in this case to each have 75 NDOs in their cache).
GET-RESPONSE Delay	The time from a device sending a GET until the corresponding GET-RESPONSE is received by the same device. This time includes the transmission time over the Bluetooth CL plus the processing delay at the two end-devices.
Bluetooth test-socket delay	The time required to establish a Bluetooth connection to a remote device, whether it succeeds or fails/timeouts.
Sync Success-Rate	The fraction of successful Sync iterations for a device. A successful Sync iteration is an instance of a synchronization process where a successful SEARCH-RESPONSE is received and all of corresponding GETs are sent - whether the GETs succeed or not.
Sync Duration	The duration for one successful Sync iteration for one device.

5.2.10 Results

This section presents and analyzes the results of experiment 2 with the metrics defined in Section 5.2.9. In the presentation of the results for this experiment and experiment 3, each device is represented by a color according to the color scheme used in Figure 5-13. This is to clarify and separate each device's individual results from the others.

Figure 5-13: Devices color code

5.2.10.1 Network Convergence Time (First Metric)

Network convergence time is one of the most significant metrics in the context of opportunistic communications and content sharing in an *ad hoc* environment. The faster the network converges, the better the experience for the end users. Long convergence times are undesirable and make the application inefficient for opportunistic content sharing. As stated previously, for this experiment the network is said to have converged when all of the devices have 75 NDOs in their local caches. The network convergence time is computed by subtracting the time of the first synchronization attempt by any device from the time that the last device to converge retrieves the last (75th) NDO.

Since experiment 2 emulates a multi-hop network, the expected network convergence time is higher than for a mesh network where each device hears every other device and has more opportunities to synchronize. Figure 5-14 presents the network convergence time for all 10 iterations in this experiment. It is worth noting that the edge devices (Device-1 and Device-5) have a longer average convergence time than the middle devices. This is due to the fact that in a multi-hop environment the edge devices, although they have no one else to compete with, will have a lower chance to converge as they have only one device to synchronize with.

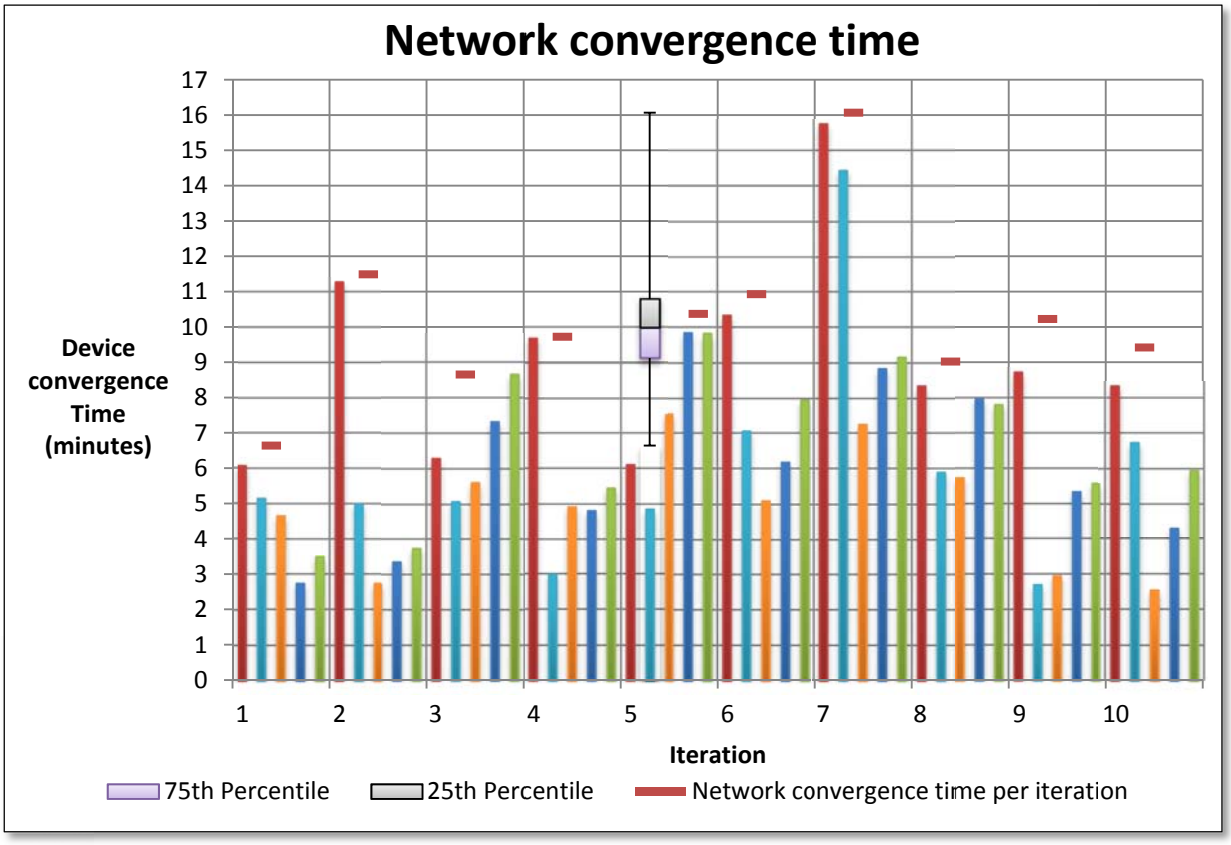


Figure 5-14: Network Convergence Time

The network convergence time for this experiment ranges from a minimum of 6 minutes and 38 seconds (iteration 1) to a maximum of 16 minutes and 4 seconds (iteration 7) with an overall average of 10 minutes and 15 seconds. Given that each device has to fetch 60 NDOs and each NDO has an average size of 29 KiB the total amount of data is 29 KiB · 60 = ~1.74 MiB. Although the amount of data is not

large, the fact that each NDO is retrieved through individual GET requests and GET-RESPONSE messages increases the convergence time. Moreover, since the application is based on the Bluetooth CL (which is prone to high transmission loss and error rate as observed in the experiment logs), the synchronization process is probabilistic, and the fact that each device can serve only one client at a time contributes to longer convergence times.

Digging further deep into the logs it was observed that in most cases the convergence time was affected mainly by the edge devices taking longer to converge and specifically by Device-1. The reason for this is due to the fact that being an edge device, Device-1 had only one device available to converge with and secondly, it had a higher transmission delay due to its location (as will be shown in Figure 5-16) compared to other devices. These factors are explored more in subsequent sections.

As observed in Figure 5-14, iteration 7 took the longest time to converge. Taking a closer look at iteration 7, it was observed that Device-1 was stuck at 74 NDOs after 6 minutes had passed and it tried 7 additional times to synchronize with Device-2 (indicated by the orange dots) to get the last missing NDO for the next 9 minutes without any success as depicted in Figure 5-15. These synchronization attempts failed mainly due to Device-2 being busy serving Device-3, trying to synchronize itself, and because of transmission errors on the Bluetooth channel. Had there not been a problem during Device-1's Sync process after it received the 74th NDO, the whole network would have converged in less than 9 minutes. More details of this behavior are present in Appendix A.

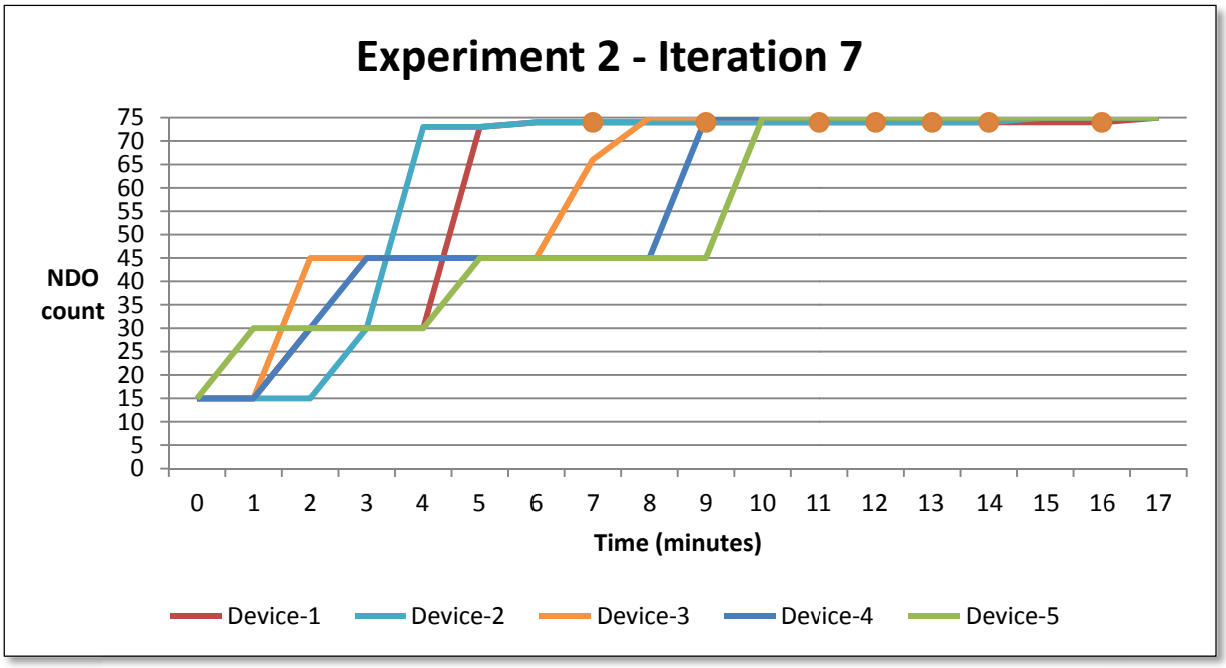


Figure 5-15: Device convergence pattern during experiment 2 iteration 7

5.2.10.2 GET-RESPONSE delay (Second Metric)

One important factor that affects the overall network convergence time is the time it takes for a device to receive individual NDOs. Since the synchronization process is based on GET requests sent by each device to its neighboring bonded devices, it is important to look into each individual GET request and the corresponding GET-RESPONSE duration and the resulting effect on the overall network convergence time. This metric has been measured by subtracting the time when a GET request was sent from the time when the corresponding successful GET-RESPONSE was received. Figure 5-16 shows the median GET-RESPONSE duration in seconds for all the 10 iterations for each device. As expected Device-1 has a higher GET response duration due to its location (as described in Section 5.2.2) compared to other devices. This transmission delay results in a slower convergence time for the device as well as causes timeouts and NDO losses that in turn results in an overall longer network convergence time.

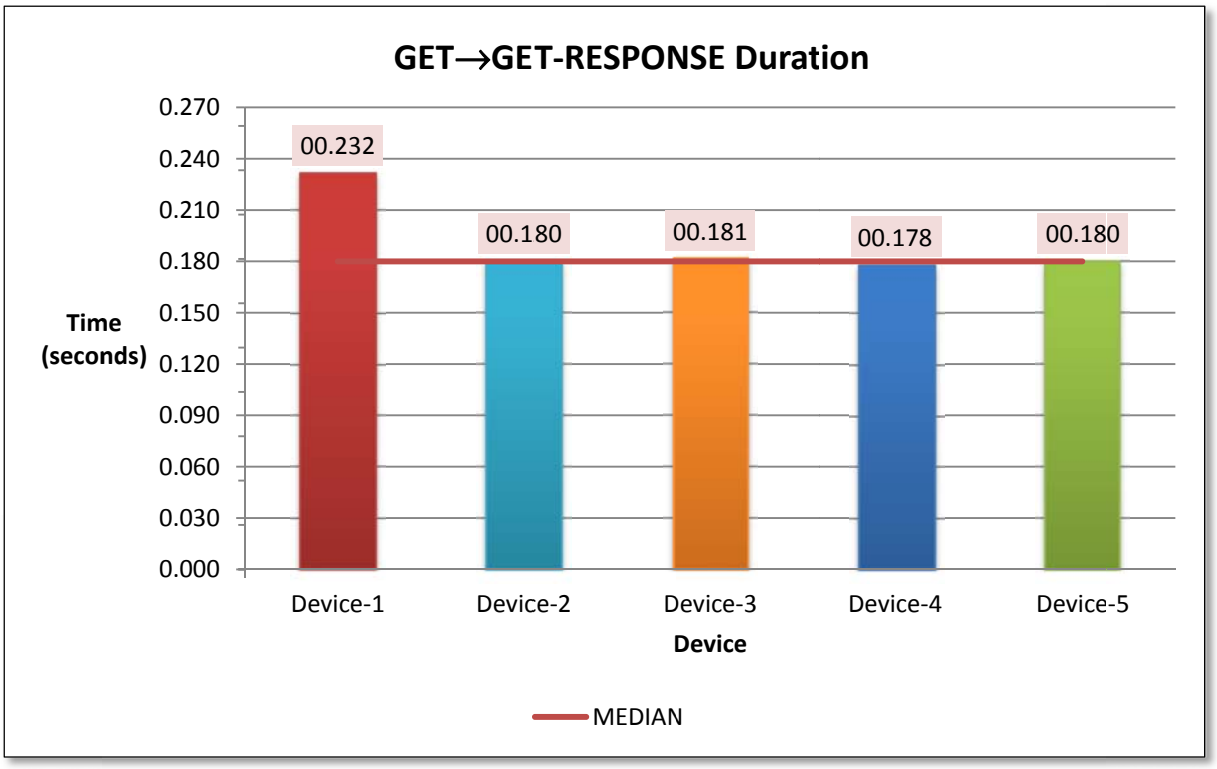


Figure 5-16: GET-RESPONSE Delay

5.2.10.3 Test-socket delay (Third Metric)

As stated in Section 5.2.6.1, each device opens a test-socket to each of the devices in its list of bonded devices in order to verify their availability at the start of each synchronization process. If a device is available, then it will accept the connection request and the socket is established to that device. This establishment step takes between 1-2 seconds. However, an issue arises when a device is unavailable, as we have to wait for the test socket to timeout. Since in this experiment each device is paired with at least two dummy devices, opening tests-sockets to these inevitably results in timeouts. The issue is particularly bad for edge devices (Device-1 and Device-5) since 3 out of their 4 bonded devices are unavailable, thus they will have on average longer synchronization times than the middle devices.

As stated previously, it was observed from the experiment logs that the timeout varies between 1-10 seconds and takes on average 5 seconds. Given that each device performs this test at the beginning of every synchronization process, this consumes a considerable amount of time and has a major effect on the overall network convergence time. In fact, on average 59% of the total synchronization duration is due to the time consumed by the test socket process. This negatively affects the overall convergence time and is due to the usage of the Bluetooth CL. This delay could be avoided by using an alternative convergence layer, such as Wi-Fi Peer-to-Peer[23] as will be discussed in Section 7.2.

Figure 5-17 shows the average amount of time taken by the test-socket process for each device. It is evident that the edge devices test-socket process takes a relatively longer time than the middle devices as the end device have one additional dummy device, thus they spend more time waiting for timeouts for unavailable devices.

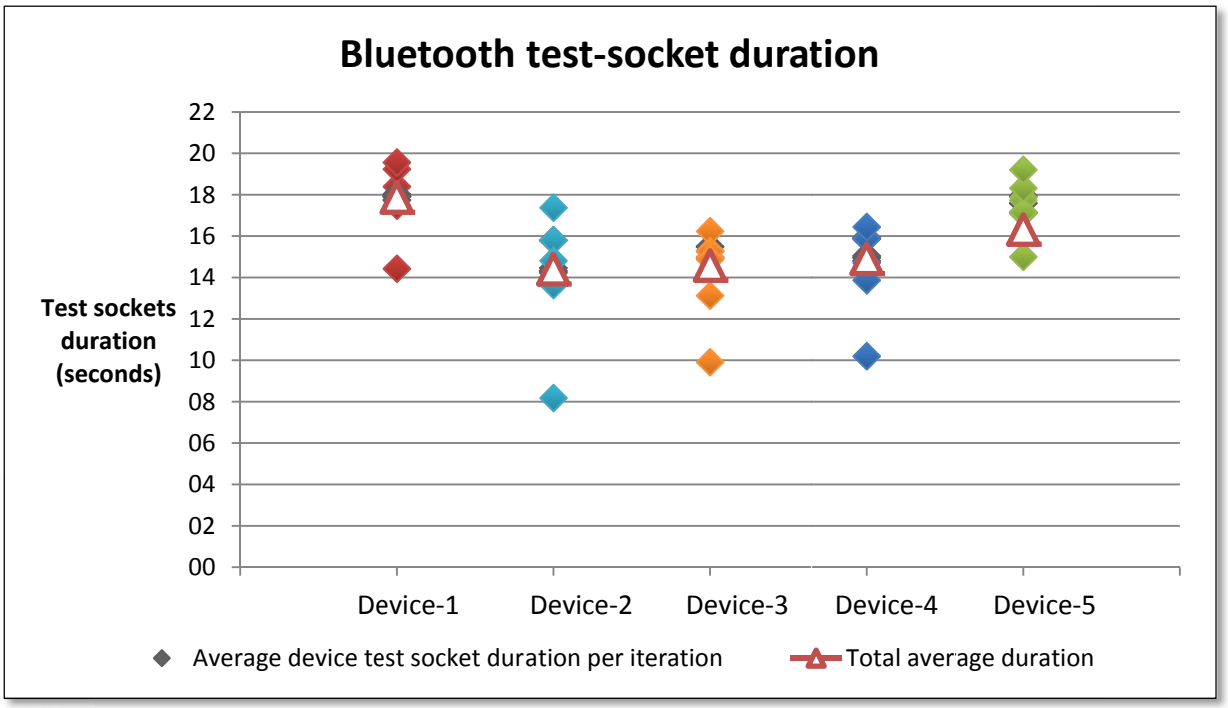


Figure 5-17: Bluetooth test socket duration

Digging deeper into the logs, it was observed that on average ~16 seconds (which corresponds with the minimum Sync interval of 18 seconds as explained in Section 5.2.7) is spent on test sockets each time a Sync Process is initiated. Given that the average number of successful synchronization attempts required to converge is 6.4, then on average $\sim 6.4 \cdot 16$ seconds $= 102.4$ seconds are spent on opening test sockets during each iteration. This means that 17% of the average network convergence time is spent on the test socket process checking the availability of neighboring devices. The issue is worse for devices that have a large number of Bluetooth bonded devices that are unavailable during a synchronization attempt. This not only delays the convergence time for the synchronizing device itself, but also reduces the chance for other devices to converge since the converging device will not be serving other devices when it is attempting a synchronization process itself. This is an area that can be improved by altering the synchronization mechanism through the usage of another CL.

5.2.10.4 Sync Success-Rate (Fourth Metric)

One of the important metrics to look at is the number of successful synchronization attempts required to converge. Successful synchronization attempts are those during which a device has detected an active neighbor, sent a SEARCH request, and received the corresponding SEARCH-RESPONSE even if no NDOs are retrieved during that attempt because the neighboring device did not have any new NDOs available at that time. The fewer the synchronization attempts required to converge, the lower the overall network convergence time. Since this experiment has a multi-hop topology and each device can serve only one client at a time, the average number of synchronization attempts are expected to be higher than would occur in a mesh topology (where each device will have more synchronization opportunities).

Figure 5-18 shows the average number of successful Sync attempts per device and the average successful Sync duration per device taken to converge. As expected the edge devices (Device-1 and Device-5) have the most number of attempts taken to converge since they had only one active neighbor from whom they could retrieve data. It should also be noted that the edge devices have the longest synchronization duration as well. This is caused mainly by transmission errors on the Bluetooth channel which causes GET timeouts as well as socket re-establishments.

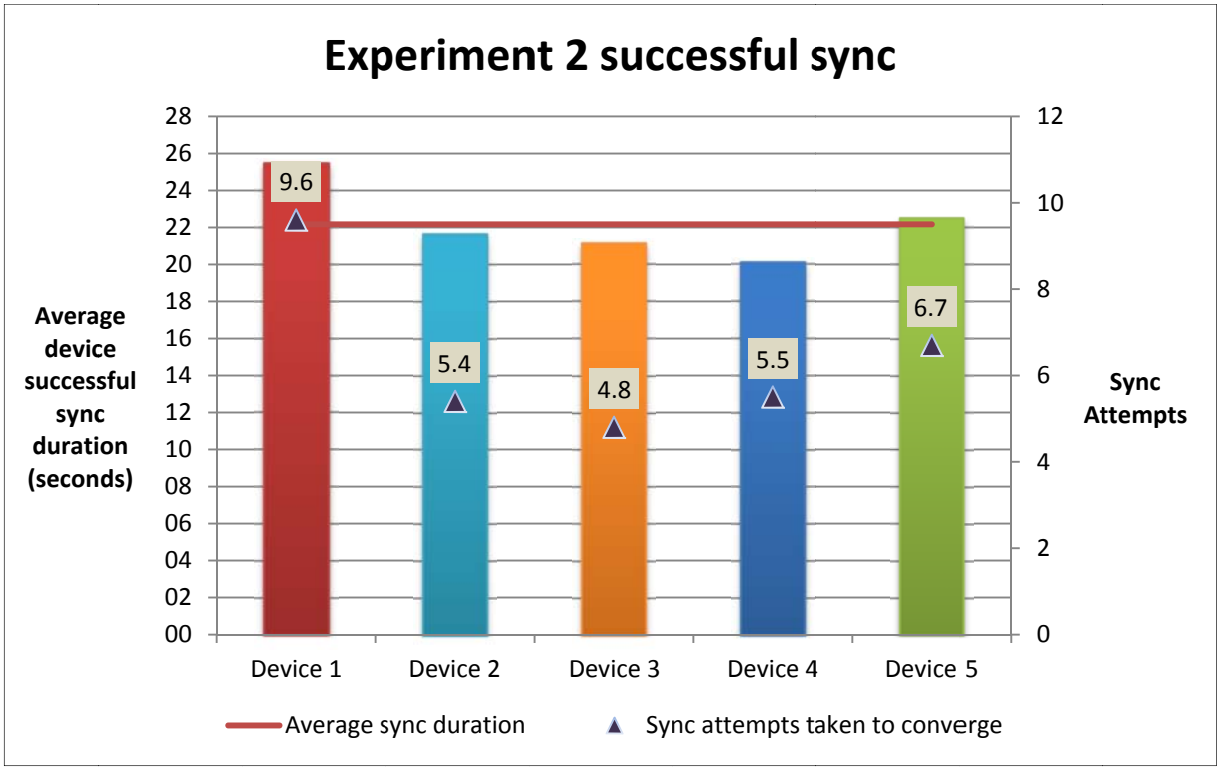


Figure 5-18: Per device average number of successful Sync attempts and sync duration

Figure 5-19 shows per device failed synchronization attempts and the type of error that caused the synchronization attempt to fail. As expected, the edge devices have the most failed attempts and 89% of the failures are due to "No device in range" that is caused when a device attempts to synchronize but cannot establish a socket to any of its listed bonded devices and has to abort the synchronization attempt. The high number of errors on Device-1 corresponds to the maximum number of Sync attempts by Device-1 as most of its Sync attempts failed due to "No device in range" error and it consequently had to make additional attempts in order to converge. This type of error is prevalent in this scenario for the edge devices due to the absence of additional available bonded devices. In a mesh topology, each device has multiple neighboring nodes available with which the device could attempt to synchronize.

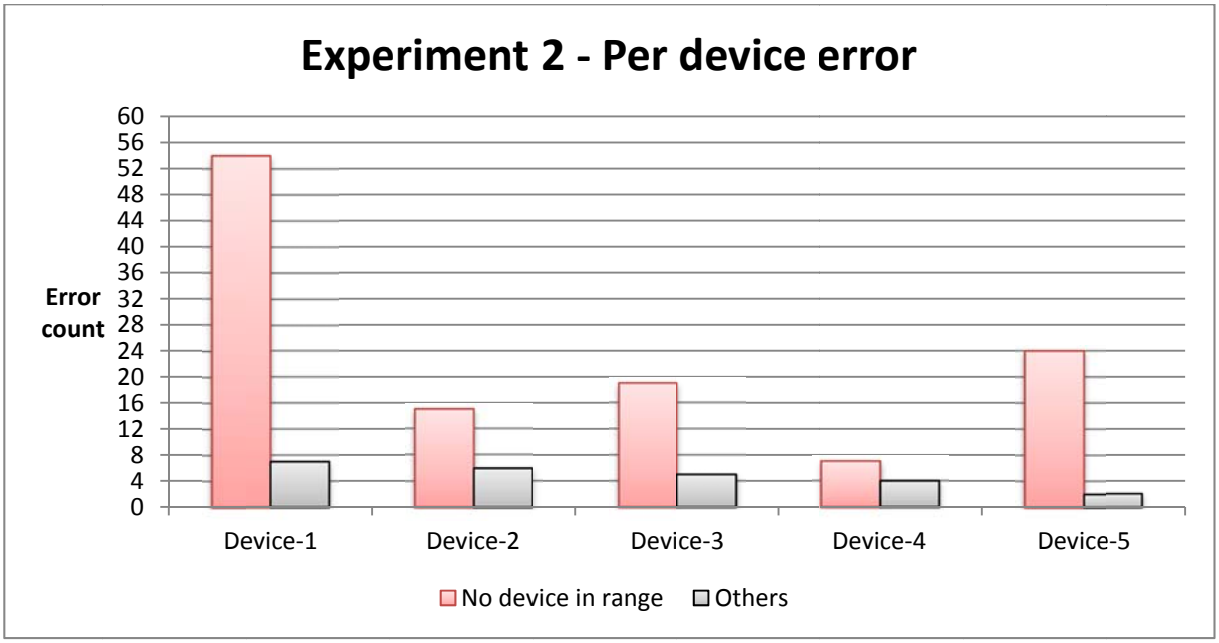


Figure 5-19: Per device error-count

5.2.10.5 Sync duration (Fifth Metric)

Another interesting metric to analyze would be the duration of successful synchronizations per device. During each iteration a device makes several Sync attempts in order for the network to converge. If a device takes a long time for each synchronization attempt, it will be unavailable for longer time and this reduces the opportunity for its neighboring devices to synchronize. This is especially important in a multi-hop scenario where each device has only a small number of available neighbors, thus being unavailable for longer periods will drastically prolong the convergence time.

Figure 5-20 shows the per device successful synchronization duration for all the 10 iterations. As we can observe from this graph, most of the synchronization attempts are less than 20 seconds, with a couple of spikes above 1 minute. The average is roughly 22 seconds and 168 milliseconds. The spikes are due to transmission errors on the Bluetooth channel. Examining the three first long spikes for Device-2 that are above 50 seconds it was observed that the synchronization process was delayed due to socket re-establishment and GET timeouts. As noted in Section 5.2.5 the timeout for individual GET requests are 5 seconds and GET requests are sent sequentially. This application waits a maximum of 5 seconds for a GET-RESPONSE message after sending a GET request. If a response is not received within 5 seconds, then the application proceeds with the next GET request. As stated in section 5.2.6.1, a device keeps a list of its available bonded devices that responded to the initial SEARCH request and when a socket is torn down during the synchronization process then the device tries to re-establish a socket back to that device. This socket re-establishment process takes between 1-10 seconds.

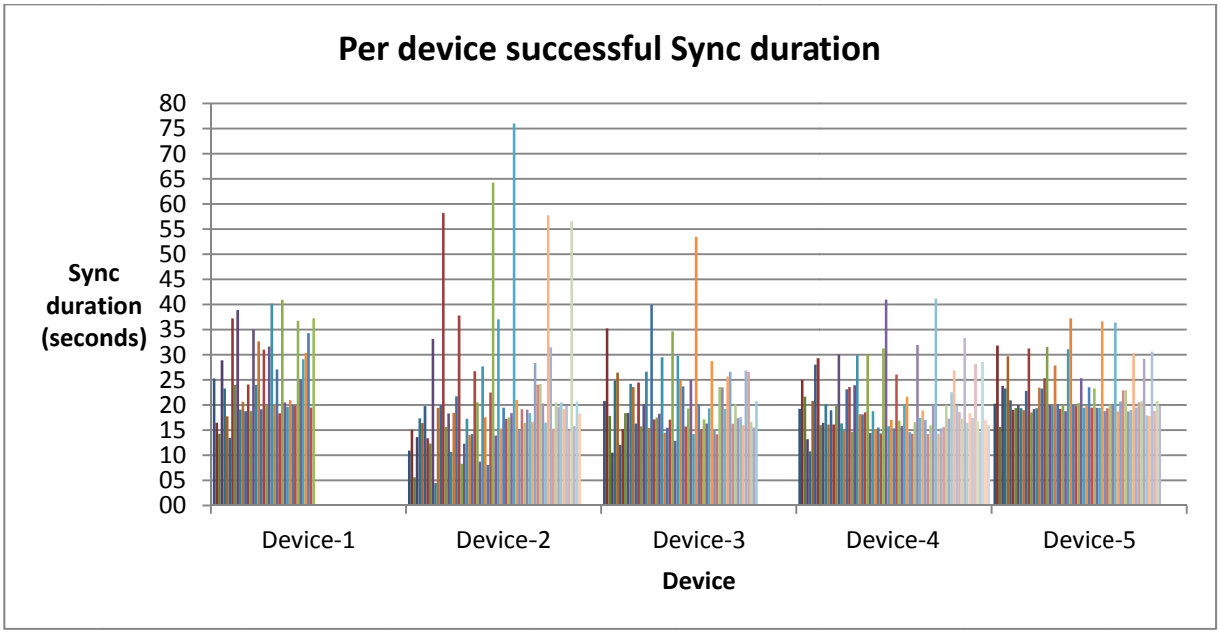


Figure 5-20: Per device successful Sync duration

During the first spike there were 3 GET timeouts and 2 socket re-establishments to Device-1 that took 15 ($3 \cdot 5 = 15$) seconds, thus in total the synchronization duration was prolonged 30 seconds due to bad transmissions via the Bluetooth channel. Similarly, during the second spike there were 5 socket re-establishments and one GET timeout that in total contributed to 26 seconds of delay. Moreover, the individual GET→GET-RESPONSEs were also slower than usual. During the third spike (which is the longest synchronization duration in the whole experiment), there was very poor connection to Device-1 with 7 socket re-establishments and two GET timeouts contributing to a delay of 35 seconds. Additionally, long transmission delays were observed for the individual GET-RESPONSEs prolonging the whole synchronization process by 1 minute and 16 seconds.

Given that the average successful synchronization duration is ~22 seconds and that ~16 seconds are spent on the test sockets during each synchronization process, this means that the average synchronization time is 6 seconds. That is to say, once a device has found its active neighbors then the SEARCH, GET and GET-RESPONSEs on average take 6 seconds to complete given that there is good quality Bluetooth channel.

5.3 Controlled Experiment 3: Semi-Mesh

The conditions and the scenario for this third experiment are exactly the same as in experiment 2 with only one difference: all devices may communicate each other *except f*or Device-1 and Device-5. Device-1 and Device-5 are edge devices that may hardly or not sense each other. The organization of the devices are the same as in experiment 2 with the locations shown in Figure 5-21.

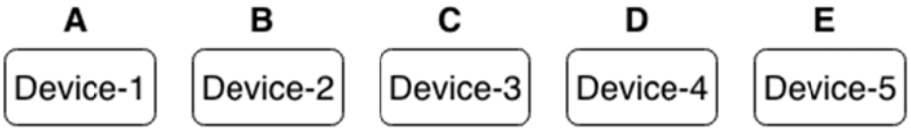


Figure 5-21: Location of each of the devices

In this experiment since almost all devices may sense each other, the behavior and results of this experiment are expected to be quite different from the previous experiment since each device has the opportunity to become a client of four other devices. The devices are paired by Bluetooth with each other, but unlike experiment 2 there are no dummy bonded devices in this experiment. Each device has a total of 4 real bonded devices as shown in Figure 5-22. The reason for a semi-mesh topology

rather than a full mesh topology is due to the locations of the edge devices (Device-1 and Device-5) they could generally not sense each other during this experiment. However, if this experiment was reproduced, there is a possibility that these two devices might sense each other. The goal of this experiment is to emulate a real life scenario where users meet opportunistically rather than intentionally, thus two users depending on time, distance, and many other factors may or may not be within Bluetooth range (i.e. in the vicinity) of each other.

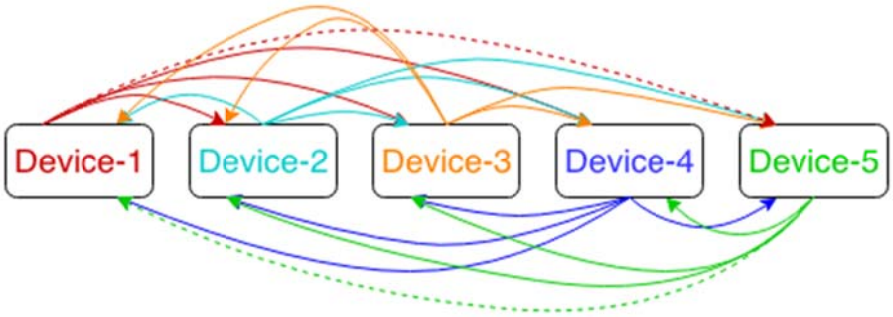


Figure 5-22: Mobile devices connection state

5.3.1 Metrics

Since most of the parameters of this experiment are the same as in previous experiment, similar metrics have been chosen to highlight the difference in results of a semi-full mesh topology compared to a multi-hop linear topology.

The following metrics have been chosen:

Network Convergence Time	The time for all devices to have fully synchronized with each other with 75 NDOs in their cache.
GET-RESPONSE Delay	The duration between a sent GET by a device until the corresponding GET-RESPONSE is received by the same device. This includes the transmission time over Bluetooth CL plus the processing delay caused at the two end-devices.
GET Success-Rate	Fraction of successful GETs at the initiating device. A GET is considered successful if a Fullput GET-RESPONSE is received.
Sync Success-Rate	Fraction of successful Sync iterations for one device. A successful Sync iteration is one synchronization process, where a successful SEARCH-RESPONSE is received and all corresponding GETs are sent, whether the sent GETs succeed or not.
Sync Convergence Time	The duration for a successful Sync iteration for one device.

5.3.2 Results

Now that all five devices may sense each other (with the limitation described above) and may exchange NDOs, the five devices compete with each other to synchronize with another available device. Additionally, this scenario allows the devices to have a higher chance to find an available device once other devices have converged. In this section, the results gathered from the logs of all 10 iterations are analyzed. The metrics that have been chosen for the analysis of this experiment are individually explored in the upcoming sections.

5.3.2.1 Network Convergence Time (First Metric)

If we assume that this in an exhibition scenario, then when these five users opportunistically come near each other, it would be of interest to know how long it takes for all five devices to completely synchronize with each other. In the context of MOSES and information-centric networking, the focus is on how efficiently content is disseminated and shared between users. This experiment attempts to estimate the network convergence time for such an *ad hoc* network. The complete network

convergence time depends on many factors, such as GET-RESPONSE delay, probability of performing a successful Sync, success-rate of individual GETs, and the quality of the Bluetooth CL transmission channel.

Figure 5-23 presents the network convergence time for all 10 iterations of this experiment. Since in this experiment almost all of the five devices may communicate with each other, they exhibit probabilistic behavior as any device at a given point in time can become client to any other device and vice versa.

Although the behavior of all of these devices are very probabilistic Device-1 (the first bar in each iteration in Figure 5-23) still exhibits a higher convergence time in the majority of the iterations compared to the other devices. This may be due to its location where its communication may be subject to a higher transmission delay (GET → GET-RESPONSE duration) compared to the other devices. This will be discussed further in the next section.

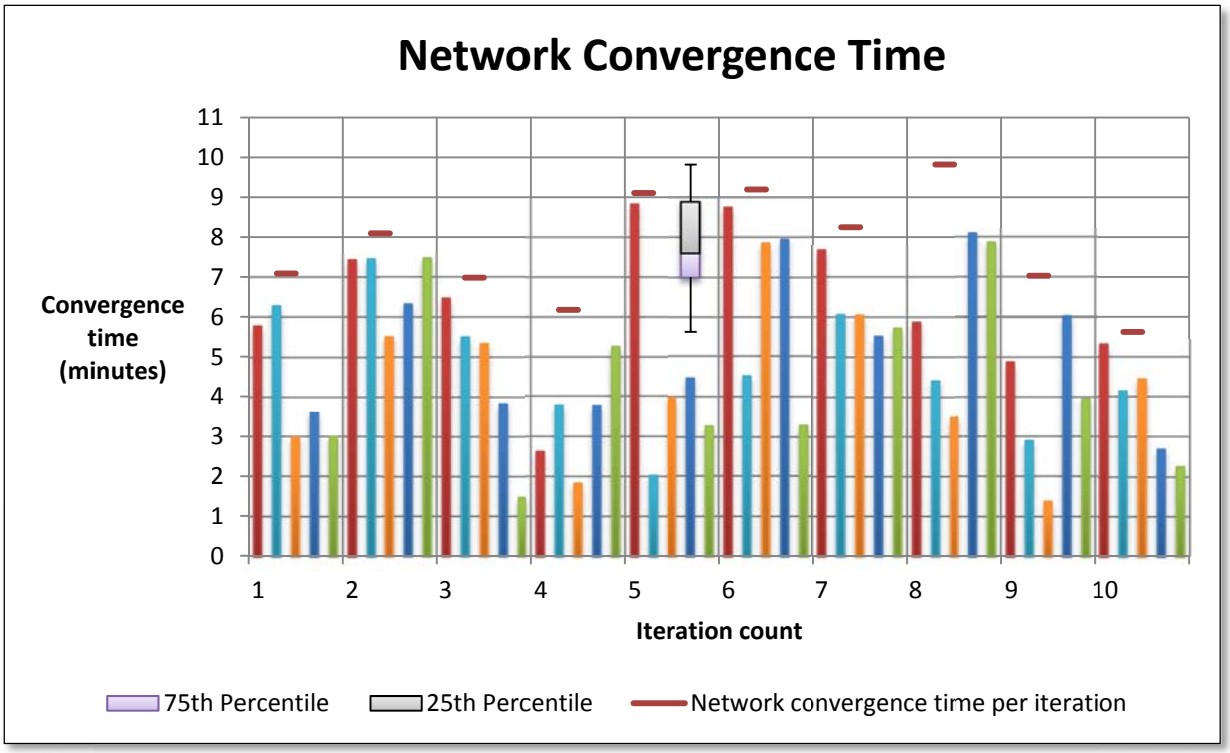


Figure 5-23: Network Convergence Time (in minutes)

In order to understand how the network convergence behaves in this topology (i.e., where almost all devices are in range of each other), the following sections will examine each of the metrics which lead to the results shown in Figure 5-23.

5.3.2.2 GET-RESPONSE Delay (Second Metric)

All five devices are within range of each other, from a device's point of view, during a Sync process up to $4 \cdot 15\ NDOs = 60$ GETs will be sent from one device to all other devices. This metric is computed from the logs of each device. The duration between a GET sent by a device until the reception of the corresponding GET-RESPONSE by the same device is computed.

Figure 5-24 presents the GET-RESPONSE average duration for each device for all 10 iterations of this experiment. The average values for each device are fairly close to each other, except for Device-1 and Device-2 that both have slightly higher values. During all three experiments, Device-1 was expected to have a bit higher transmission delay due its location. However, in this scenario Device-2 has the highest average GET-RESPONSE delay compared to all other devices.

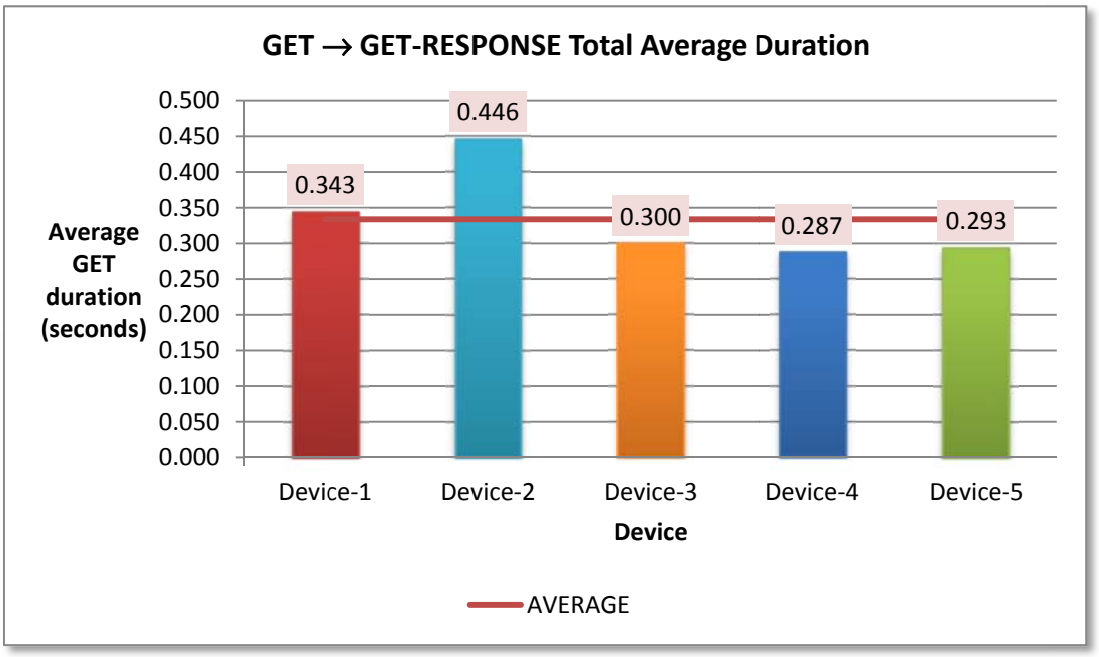


Figure 5-24: GET-RESPONSE Total Average Duration

After analyzing Device-2's logs, as shown in Figure 5-25 during experiment-iterations 1, 3, and 6, Device-2 experienced higher GET-RESPONSE delays. According to the logs, Device-2 had several GET-RESPONSE delays between ~1-2 seconds that increased its average GET-RESPONSE delay. Those individual GET-RESPONSEs that increased the total GET-RESPONSE delay average for Device-2 were mostly received from Device-1 and Device-5. This may be explained by the high transmission delays caused by the two edge device's locations. Device-1's location as explained earlier is subject to a corner and a main walkway and Device-5 has the greatest distance from Device-2 (when compared to the other devices). Since the synchronization process is unidirectional at each Sync iteration, only the initiating device synchronizes with the remote-device(s) and the remote-device(s) waits until the device has completed its synchronization. Therefore, in Figure 5-25 the results of each device in each experiment-iteration are unrelated to another device's results. In another words, just because Device-2 had a poor connection to Device-5 and Device-1, does not mean that Device-5 and Device-1 during their synchronization necessarily had poor connections to Device-2 (e.g. Device-5 during iteration 3 only synchronized with Device-4 and during the same iteration Device-2 synchronized mostly with Device-5). In this network topology, almost anyone may synchronize with anyone.

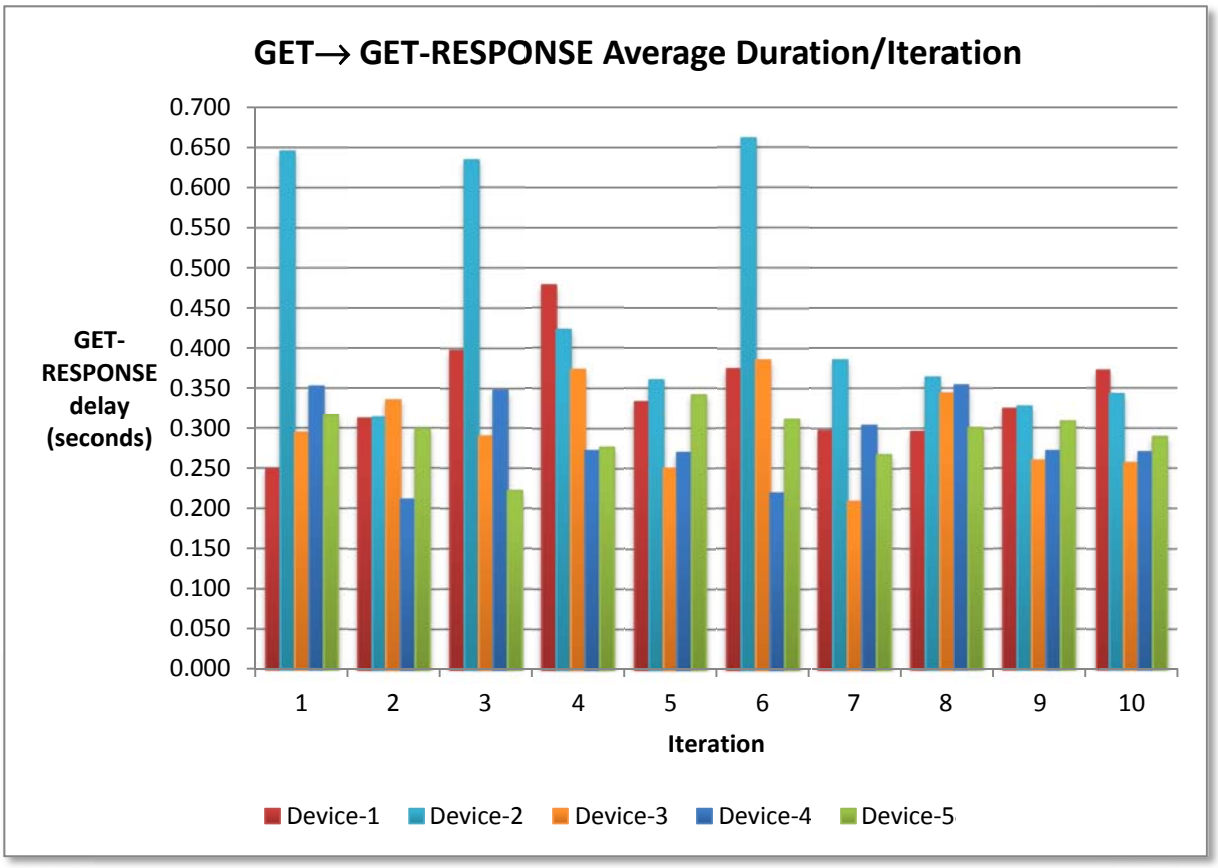


Figure 5-25: GET-RESPONSE Average Duration/Iteration

5.3.2.3 GET Success-Rate (Third Metric)

Both from a protocol/concept and from synchronization efficiency point of view it would be interesting to examine the success-rate of the performed GETs. After all, the ICN concept and the NetInf GETs have a big role in the behavior and the efficiency of this application in the MOSES opportunistic networking concept. Any failure of individual GETs causes extra Sync iteration(s) in order for the device to converge and since successfully becoming a client to other devices is very probabilistic, this may delay the network convergence time.

Both Figure 5-26 and Figure 5-27 show a special behavior of the GETs. This behavior would not occur in a host-centric networking approach (i.e., using TCP). In this experiment, each device should perform 60 individual GETs. In a host-centric network each GET would be transmitted only to the device holding a copy of the requested information. In the case of a transmission-failure or packet-loss the request (or packet) would be re-transmitted. This re-transmission is necessary, therefore it is not a reason for inefficiency. However, in an information-centric networking approach, a request (GET) for information is sent to the network and not to a specific device. Therefore as described earlier in Section 5.2.5, in this application GETs are randomly sent to neighboring devices whether the remote-device holds a copy of the NDO or not. In an ICN-based infrastructure type of network, there may be more control over where an NDO might be available, but in an *ad hoc* network this NDO could be in any of the other nodes. The Sync process uses the aggregated SEARCH mechanism and performs GETs randomly to those devices that the initial SEARCHs were sent to. Therefore a GET is sent to several devices and the first GET-RESPONSE is accepted and the rest are ignored. This behavior is apparent in Figure 5-26. For example during iteration **4**, Device-2 has sent 123 GET requests to its neighboring devices and 60 GETs where accepted. The NDOs marked as "FAIL" in Figure 5-26 are the GETs that were sent without receiving/considering their corresponding GET-RESPONSEs. These GET-RESPONSEs might have failed at the physical layer, timed out, or it may have been ignored since the remote device did not have a copy of the requested NDO or it was a duplicate GET-RESPONSE.

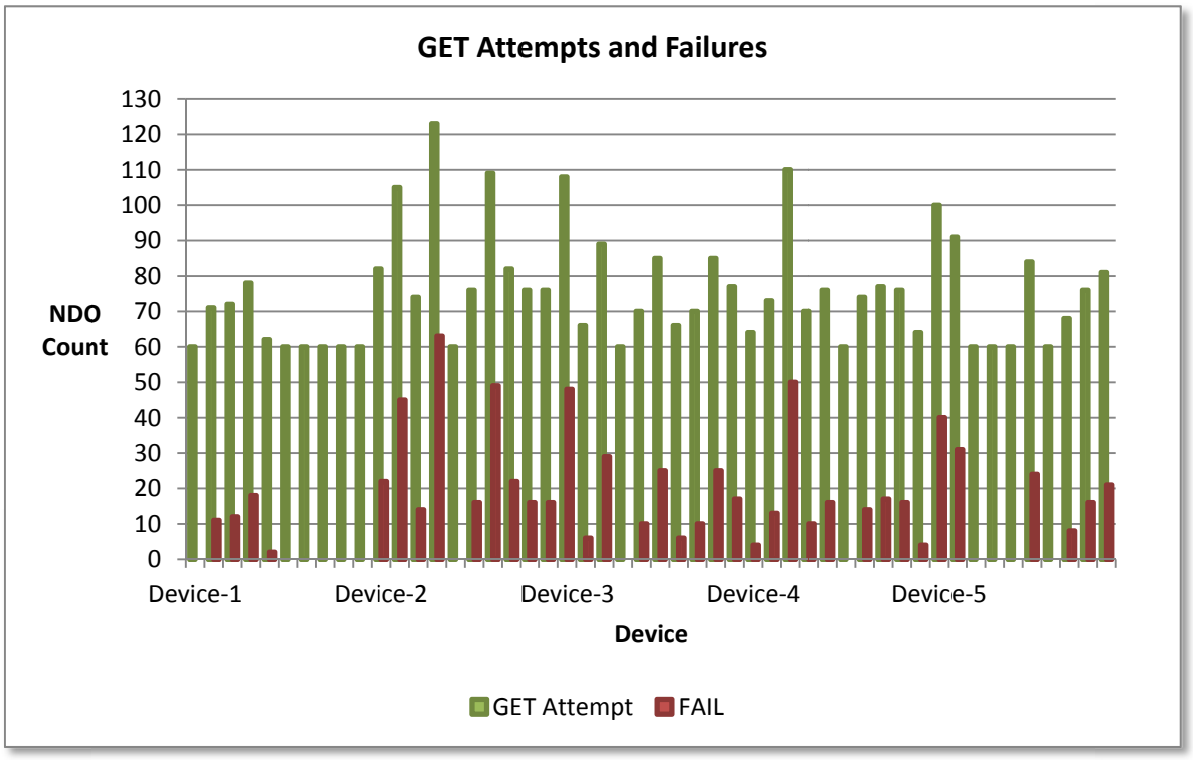


Figure 5-26: GET attempts and failures

This ICN-based behavior of this application of sending GETs randomly to neighboring devices may lead to efficiency or inefficiency. Since in this experiment (nearly) all devices could sense each other and may Sync with each other, a less distant device may fully converge before another device has initialized its own Sync. Therefore, this second device may GET an NDO from a less distant neighboring device than the originating device (the device who owned the NDO first before giving it to others). This originating device may be at a longer distance or even be out of range now. As a result, this behavior may lead to greater efficiency for the network. However, there are cases such as seen in iteration 4 in Figure 5-26 where Device-2 (Device-2's fourth bar in Figure 5-26) has sent 123 GET-requests, 60 of the GET-RESPONSES where accepted, and 18 of the remaining 63 GETs according to Device2's logs where subject to a Bluetooth socket-failure. Which leaves us with 45 remaining GETs which are duplicate GETs that were sent to other devices and their corresponding GET-RESPONSEs were ignored either because they were received late or the GET-RESPONSE did not contain a Fullput NDO. In this case the ICN-based behavior leads to inefficiency in the network, since 45 GET requests were unnecessarily sent. This inefficiency may contribute to congestion of the transmission channel, as well as occupying the two end-device's hardware/software with unnecessary processing.

According to the logs 85.90% of all the "failures" marked in red in Figure 5-26 where due to duplicate GETs. An additional 13.45% of all the "failures" were due to Bluetooth socket-failure and the remaining 0.65% of "failures" were due to timeouts. A Bluetooth socket-failure means that the device was unable to establish a connection with the remote-device. This may be due to failure at the physical layer or the remote-device was unavailable for the moment (e.g. performing its own Sync) or the remote device was out of Bluetooth range.

Figure 5-27 presents the efficiency/success-rate of each device in each of the 10 iterations. A device's result is considered 100% successful/efficient if in each iteration only 60 GETs and 60 corresponding GET-RESPONSEs were exchanged between a device and its neighboring device(s).

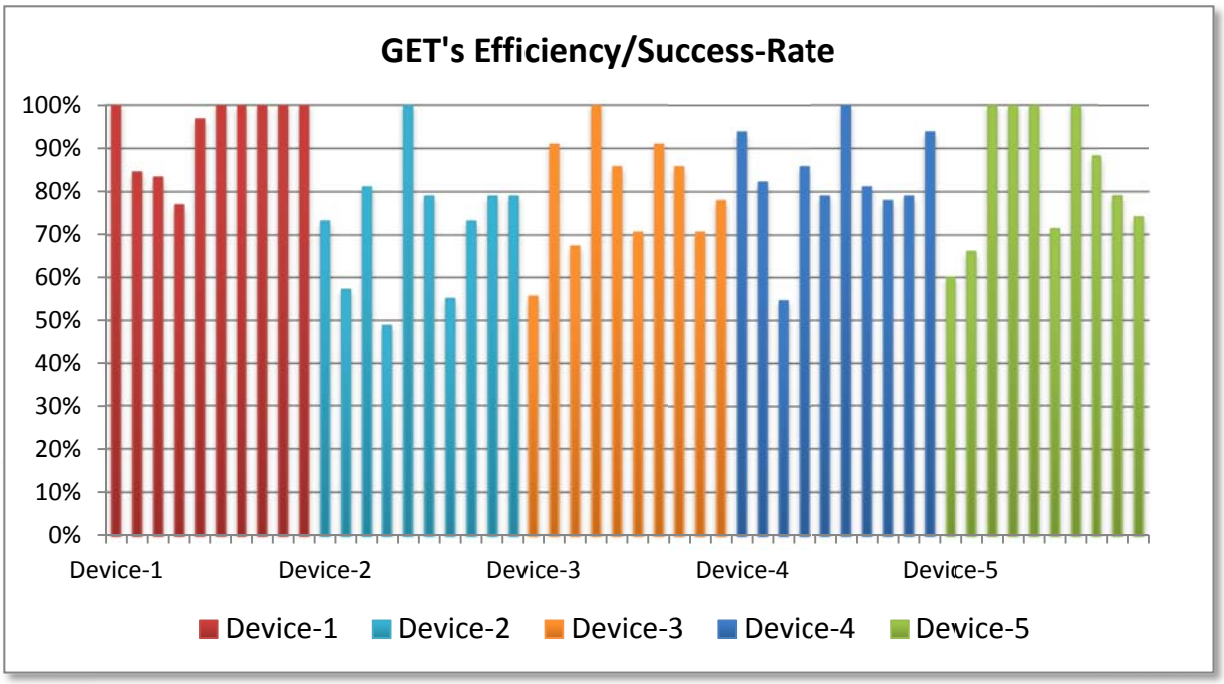


Figure 5-27: GET's efficiency/success rate

One may prefer to analyze the efficiency and success-rate of GETs separately and explore them in more detail, but the main idea of this analysis is to give an overview ICN's GET behavior in a MOSES application and to identify possible factors for inefficiency in the protocol/concept itself.

5.3.2.4 Sync Success-Rate (Fourth Metric)

After clarifying the GETs behavior and influence on the network convergence time in a semi-mesh topology in this experiment, we analyze which factors may affect/interrupt a synchronization process. We analyze each Sync iteration success/failure in the logs. For every Sync iteration that fails/postpones, an extra Sync iteration is required in order for the device to converge, which adds extra delay to the network's convergence time.

Figure 5-28 presents the results of all the Sync attempts and failures during the whole experiment. There were a total of 451 Sync process attempts carried out by all the 5 devices during all the 10 iterations. Of these 451 attempts, 174 Sync iterations failed. Examining the logs and analyzing the cause of each failure/interruption, it turned out that 143 of the 174 failures were due to "No device in range" error. This error occurs every time the Sync process is performed and the Bluetooth CL sensed no remote-devices. This does *not* actually mean that there were no Bluetooth devices within range, as the neighboring devices might have shutdown their Bluetooth-Servers due to either serving other clients or performing their own synchronization.

The remaining 31 failures are categorized as "others" (as apparent in Figure 5-28). These failures were due to momentary software/hardware misbehavior, which might be fixed by improvements to the implementation. The errors in this category are not primarily related to the concept/protocol itself, therefore they are not discussed further in this thesis.

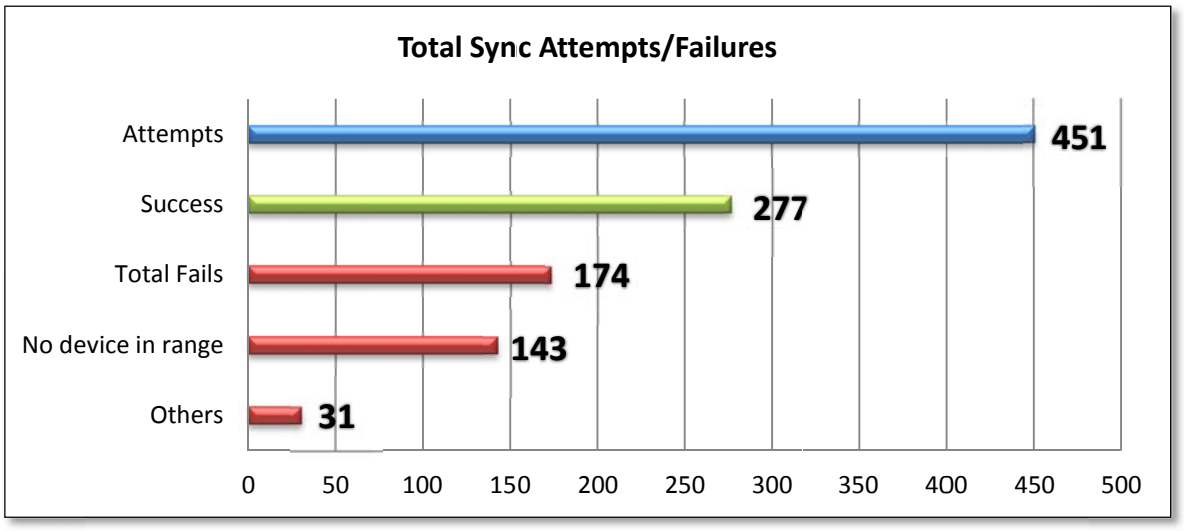


Figure 5-28: Total Sync. Attempts/Failures

As presented in Figure 5-28, 82.18% of all the Sync process interruptions during the whole experiment were caused by the "No device in range" error. The primary cause for this error is due to the design of this application and the limitations of the Bluetooth CL. In opportunistic networking where users have limited opportunities to exchange information (i.e. NDOs) with each other, it is unfortunate to lose synchronization opportunities due to serving one client at a time.

5.3.2.5 Sync Convergence time (Fifth Metric)

Similar to the previous experiment, one of the metrics in this experiment is the Sync duration for each device. In opportunistic networking, a user comes within range of another user and the two users have a limited amount of time to exchange NDOs before they move out of range of each other. Therefore, when a device gets an opportunity to exchange NDOs with another device, it should try to exchange as many NDOs as possible.

Table 5-5 presents the average Sync duration as well as extra data which may affect the duration of each Sync iteration. As stated earlier, Device-1 is positioned at location "A" which is subject to a corner and the corridor were Device-1 is positioned is a main walkway for personnel. These factors may have negatively affected the wireless transmissions of Device-1 and Device-2. As stated earlier and as presented in Table 5-5, Device-1 and Device-2 have the highest average GET-RESPONSE time compared to the other devices, which may explain their high average Sync duration.

Table 5-5: Sync duration results

Device	Average Sync duration (seconds)	Total number of failed Sync iterations	Sync Total Success-rate	Average GET-RESPONSE Duration (ms)
Device-1	23.951	42	53.3%	343
Device-2	21.663	22	72.5%	446
Device-3	14.615	31	66.3%	300
Device-4	19.102	40	55.6%	287
Device-5	17.678	39	60.6%	293

A device is considered converged when it has received all of the other device's 15 NDOs, leading to a total cache of 75 NDOs (i.e., after including the device's own 15 NDOs). This individual convergence time was calculated by considering each device's cache size after each Sync process using the log files. More information about the behavior of each device throughout the 10 iterations of this experiment is present in Appendix B. This appendix presents in which order in time each device updated its cache until it finally reached a total cache of 75 NDOs and converged. The analysis of each device's convergence time and behavior in this *semi-mesh* topology may be of interest since it may illustrates the efficiency and anomalies that may arise due to the ICN concept or opportunistic networking in

ad hoc mode. One interesting anomaly is present in Appendix B Experiment 3 - Iteration 5, which illustrates how a device due to a GET failure may be delayed in converging. Each device probabilistically becomes a client to another device and synchronizes. Since becoming a client to another device is very opportunistic and probabilistic, it might take one to more Sync iterations in order to find an available neighboring device to synchronize with. In Appendix B Experiment 3 - Iteration 5, Device-1 during its second to last successful Sync the device failed to receive one GET, therefore it continued to have 74 NDOs for 3 minutes until it finally found an available device to retrieve this last NDO and to converge. In this experiment since all devices may sense each other, missing an NDO may only delay a device's convergence time by a small amount of time because a device may retrieve the final missing NDO from any of the other four neighboring devices that have already converged. However, in a multi-hop scenario as in experiment 2, the chances of finding an available neighboring device is lower and this may delay a device's convergence time much more.

Figure 5-29 shows the synchronization convergence time and number of sync attempts from an overview perspective. We analyzed whether Sync duration and the number of Sync iterations share any common factors. A successful Sync duration as discussed earlier, is very dependent upon the number of GETs performed and their durations. However, a Sync attempt might be interrupted/postponed for various reasons (as discussed previously). Does a longer Sync duration mean more Sync attempts are necessary to converge? One commonalty between the two may be that a longer Sync duration is usually caused by poor link quality. This may affect/delay the initial socket establishment, which means that a device that is far away has a lower probability to successfully compete a simultaneous connection establishment to a common neighboring device. In another words if two devices perform a simultaneous Sync to the same remote-device, then the device that is closer may be more likely to become a client of the remote device.

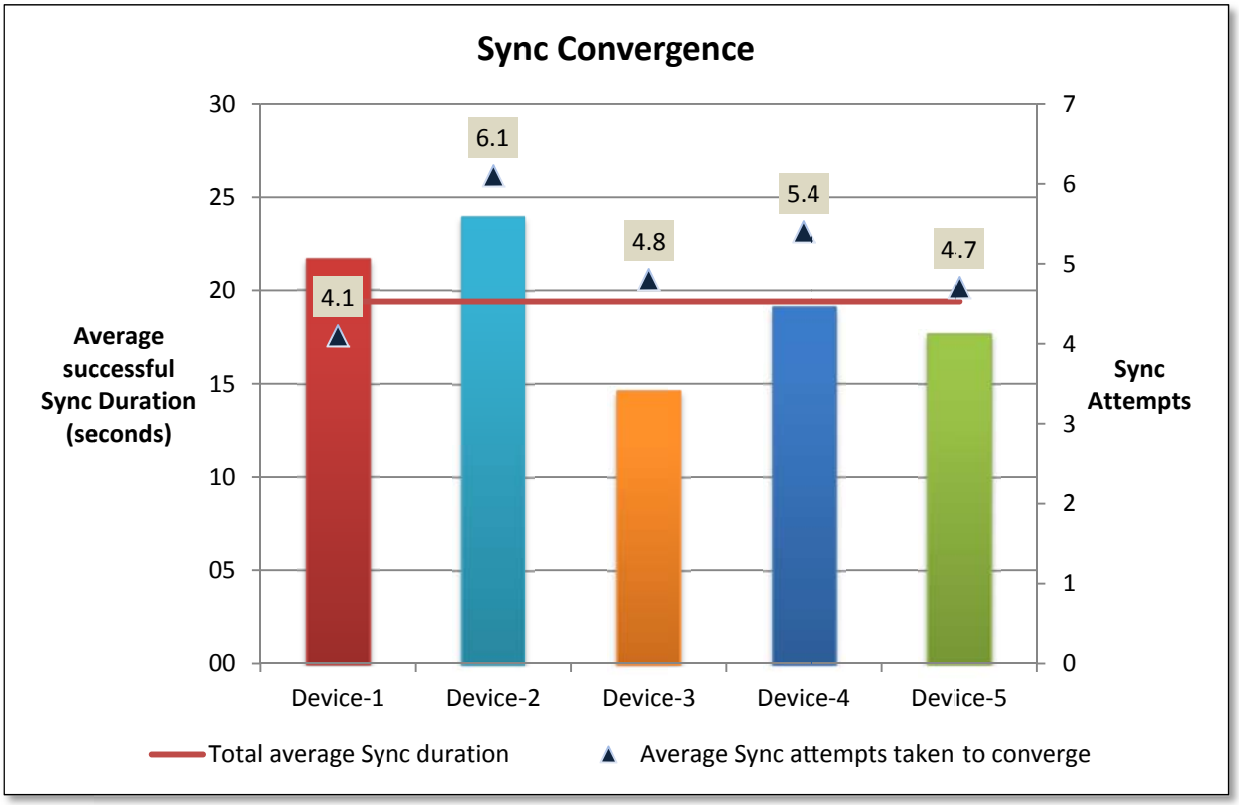


Figure 5-29: Sync. Convergence

5.3.2.6 Device Sync patterns

In this experiment since each device may have the opportunity to Sync with four other devices, it may be of interest to view the pattern of synchronization in this semi-mesh topology. The term "pattern" in this section refers to the pattern of connectivity and synchronization of NDOs between the five devices, i.e., from whom did a device get its NDOs. Table 5-6 presents from which other device each device got what portion of its NDOs during all 10 iterations of this experiment.

Table 5-6: Sync pattern (N/A indicates not applicable)

Synced with	Device-1	Device-2	Device-3	Device-4	Device-5
Device-1	N/A	24.67%	41.17%	0%	0%
Device-2	31.5%	N/A	19.17%	36%	41%
Device-3	67.83%	24.5%	N/A	26.67%	7.83%
Device-4	0.67%	20.33%	39.67%	N/A	51.17%
Device-5	0%	30.5%	0%	37.33%	N/A

As described in Section 5.1.5.1 regarding the results for location D in a point-to-point synchronization no matter what the distance is if a device successfully establishes an initial Bluetooth connection to another device, it may have a GET success-rate up to 100%. As presented in Section 5.1.5.1, the distance may cause delay in the transmission but it does not necessarily affect the success-rate - as observed in experiment 1. This behavior can be seen in the results presented in Table 5-6 as well. For instance, Device-1 is subject to a corner and a main walkway for personnel, but it still managed to receive the majority of its NDOs (i.e. 67.83% of 600 NDOs) from Device-3 which is not its closest neighboring device. Looking into Device-1's results, as present in Table 5-6, shows also that Device-1 received only 0.67% of its NDOs from Device-4, which shows the similarity to the A→D case in experiment 1 as well as the A→E case since in this experiment Device-1 received no NDOs from Device-5. This similarity enables us to conclude that once a device has successfully established a Bluetooth socket to another device, no matter what the distance, obstructions, interference, or path loss, the device can perform the majority of its Gets and synchronize with the remote device. Therefore initial Bluetooth socket establishment is one of the essential factors in an opportunistic networking, as if a device manages to establish a Bluetooth socket to a relatively (relative to the range of Bluetooth) near or faraway device, it could successfully Sync with up to 100% success-rate and may converge.

It should be kept in mind for all the conducted controlled experiments in this thesis, that there was no specific Sync protocol, therefore the synchronization pattern is very opportunistic and thus probabilistic. There are three probabilistic behaviors involved in the Sync process. The first probability is the value of *R* chosen by the devices in a network. For instance, assume that two devices with identical application settings (i.e. same CL, routing, Sync timers, etc.) initialize their Sync process by chance (depending on the *R* value). However, they may or may not start Syncing at the same time, because if the *R* values picked by the two devices are the same or close to each other (without the 18 seconds window) then the two devices may not become a client to one another and will postpone their Sync. The second probability involved is the availability of the device itself, which is the probability of not serving a client when initiating a Sync to it. The third probability is the availability of the neighboring devices, depending on the number of the devices that are within the range of a device, what is the probability that the neighboring devices are not serving a client or performing their own Sync at the point in time when the device initializes its Sync process.

6 Contributions to MOSES

Since this thesis project is part of a bigger research project, some tools were developed in the course of this project – have not yet been discussed in this thesis. The following sections describe two implementations that were developed and might be of use in future development of applications of MOSES.

6.1 Extension of the Android NetInf Library

The Android NetInf library[16] provided by MOSES, among other functionalities supported transmission over Bluetooth, HTTP, and UDP CLs. The HTTP CL of the library did not support SEARCH and PUBLISH functions. Thus two mobile devices could only perform a GET on each other over the HTTP CL. This library was extended with two Java classes: PublishHandler.java and SearchHandler.java. These classes were implemented to use the same architecture as the existing classes of the library and reused code from the library. Using these two new classes, devices can perform SEARCH and PUBLISH over the HTTP CL both to other devices and a NetInf server.

6.2 Python Synchronization Script

Section 4.4 mentioned the idea of synchronization checkpoints (i.e. NetInf servers) where users may use the infrastructure to disseminate their NDOs. This idea was deployed during the uncontrolled experiment at the SICS Open House event, but it was more an idea rather than a product of this thesis, therefore it was not discussed earlier in this thesis. A Python script was written in order to keep the two synchronization checkpoints synchronized with each other. As a result the NDOs published by users to each synchronization checkpoint would be available in the two "zones".

This script completes the synchronization process in two steps. In the first step a NetInf SEARCH message is sent from server “A” to server “B” with the “notavailable” flag along with a list of all the cached NDO names. Upon receiving this search message server “B” replies with a SEARCH-RESPONSE message with all the NDOs that it does *not* have in its local cache. Server “A” then publishes all of the NDOs on server “B” that server “B” does not have, thus server "B" becomes synchronized with server "A". In the second step, server “A” sends a normal SEARCH message with the search token "*" and server “B” responds with list of all NDO names it has in its cache. Server “A” then compares the received list in the SEARCH-RESPONSE message and sends a GET message for each NDO that is not present in its local cache and thus server "A" becomes synchronized with server "B". This script could be run periodically (e.g. every 5 minutes) to keep the two NetInf server's caches synchronized with each other.

7 Conclusions and Future work

This chapter begins by presenting some conclusions. This is followed by some suggestions for future work and reflections on the social, ethical, and other issues relevant to this thesis project.

7.1 Conclusions

A functioning Android prototype application for the MOSES project was created, thus the main goal of this thesis project was achieved. This prototype was built based on the NetInf protocol. This prototype can be used for experience sharing by visitors to an entertainment venue. The application uses Bluetooth to disseminate location relevant content opportunistically between mobile devices in an *ad hoc* environment. A series of controlled experiments were conducted under supervision of MOSES researchers in order to test, verify, and evaluate the functionality of the prototype application as well as to analyze its efficiency for opportunistic content sharing. The results of these controlled experiments provided additional insights for MOSES researchers into using ICN (NetInf) for opportunistic content sharing and allow them to review the concept. Additionally, they can further develop and refine the prototype Android application and the involved algorithms.

The three controlled experiments attempted to evaluate different scenarios and based on these results some key results and concerns regarding MOSES/ICN concept can be summarized as follows:

i. The network converges faster with more devices, as evident from the results of experiment 3 where the average network convergence time was 7 minutes and 44 seconds as opposed to 10 minutes and 15 seconds for experiment 2. Thus, additional devices reduce the convergence time, but since a device cannot have more than 7 Bluetooth active connections at a time any node in the network will be limited to 7 active neighbors.

ii. The Bluetooth limitation of accepting one client per UUID limits devices to serving only one client at a time. Having a Server/Master listening on different UUIDs does *not* improve the efficiency and will only prolong the test-socket process further.

iii. The test-socket timeouts are very high when a device is unavailable and thus considerably increases the synchronization time. Moreover, it has been observed that the Bluetooth transmission channel is prone to transmission errors for distances greater than a few meters and many socket tear downs (i.e. socket-failures) were observed between devices during each synchronization process further prolongs the convergence time.

iv. As stated in Section 5.2.5, once a device receives a SEARCH-RESPONSE message, it calculates which NDOs it is missing in its local cache and starts sending GET requests to randomly selected neighbors. This is to comply with ICN architecture as the device is only interested in receiving the content (i.e. NDO) and it is irrelevant of where the NDO comes from. The node sends requests sequentially to all available devices and waits for the first response. This mechanism of sending multiple GET requests to fetch an NDO causes congestion on the transmission channel and unnecessarily uses the device's processing and other resources. This becomes worse when several convergence layers are used simultaneously as the GET requests will be sent over all convergence layers. For example, if both Bluetooth and HTTP CLs are chosen by user then after receiving a successful SEARCH-RESPONSE, the application will send GET requests to all of the available devices via both convergence layers. This issue can be avoided with a host-centric approach by binding a SEARCH-RESPONSE to a GET request. In this approach after receiving each successful SEARCH-RESPONSE, the device remembers which device sent the response and then during the GET process each individual GET request is sent only to the device that actually holds the NDO instead of sending the request randomly to all available devices.

7.2 Future work

The thesis focused primarily on building a prototype application using the NetInf protocol for opportunistic content sharing and conducting controlled experiments to analyze and evaluate the performance of the prototype application. Several aspects were out of scope of this thesis and could be further developed to extend the functionality of the prototype application. This section summarizes some of those aspects that remain undone, together with some related work that may be of interest for future thesis projects.

i. It is evident from the experiment results that the transmission channel is of great importance and has a major impact on the efficiency of the application. Currently there are two alternatives for mobile *ad hoc* networks namely Bluetooth and Wi-Fi Direct. The application used Bluetooth since it had been previously worked on and developed during a Master's thesis project by Linus Sundé[16] and the Bluetooth CL was already in the Android NetInf library. As discussed in Section 5.2.6 there are several issues which make use of Bluetooth undesirable for opportunistic content sharing. As future work, it is recommended to spend time and effort on a new CL, specifically Wi-Fi Direct[23], as an alternative before proceeding with further development of the prototype application.

ii. Since the main aim of this Master's thesis project was to build a functioning prototype application for opportunistic content sharing, little time was spent on optimization of the application itself. Caching and sharing NDOs consume considerable resources (such as random access memory and battery power) of the mobile device. In future work it would be advisable to further develop and optimize the application to use fewer resources as well as to refine the GUI. Moreover, there have been some refinements done to the Android NetInf library during this thesis project, but more investigations are required to further optimize and develop the library as suggested by Linus Sundé[16].

iii. Although the application can use HTTP CL for both *ad hoc* and infrastructure synchronization this CL was out of the scope of this thesis project, hence no experiments were performed using the HTTP CL. It would be of interest to conduct similar experiments simultaneously using both Bluetooth and HTTP CL.

iv. Rather than implementing a completely opportunistic/probabilistic behavior as is the case with the current implementation, it would be advisable to either extend the current Sync algorithm or devise a completely new synchronization protocol that would be more efficient within the framework of MOSES and the information-centric concept.

v. The controlled experiments were limited to only five devices. As is evident from the results, the convergence time decreases when more devices communicate with each other. It would be of interest to conduct experiments with more devices and observe the resulting behavior. Moreover, in order to analyze the behavior of the application for real life scenarios, a series of uncontrolled experiments should be conducted where the devices are distributed to end users whom would generate and share content opportunistically in an uncontrolled manner.

vi. The results of the controlled experiments presented in Chapter 5 are primarily intended for MOSES researchers to reproduce the experiments with the desired metrics that are of most interest to them. An interested researcher should repeat the experiments presented in Chapter 5 with more than 10 iterations in order to acquire better distributions and observe in more depth the deviations of the results.

7.3 Reflections

From the end user's point of view, applications such as MOSES intend to decentralize sharing and dissemination of digital content, thus offloading content servers. This approach might give end users the opportunity to exchange data on their own and in a more on-demand manner *without* needing a subscription to an Internet service. Opportunistic location relevant content may also be beneficial from an educational perspective, as users who own a Smartphone device may receive informative news/information about the locations they pass through.

Offloading content servers could lead to reduced load on the infrastructure and thereby a decrease in deployment and maintenance costs for commercial information providers. Because if every user's device is a content server, there would be no need to ask for content from content servers on the Internet when that content is available from the user(s) nearby. This could also imply less load or distribute the load on the environment. Because mobile devices are usually power efficient (since they are constrained their battery's power), short ranged wireless technologies could be used by power efficient devices (i.e. Smartphones) to satisfy their users' demands. This is likely to lead to a reduction in the size of content size (via compression or other algorithms), thus decreasing the content's size on the global network.

With all the benefits of MOSES, there are also privacy and ethical concerns introduced. Personal data confidentiality and end user activity discretion are two factors that may violate the end user's privacy via a MOSES application. Opportunistic sharing or sharing in general introduces privacy issues such as revealing a person's identity, activity, location, or any other personal information to unauthorized users. Confidentiality of end user data are discussed in some papers of ICN[18], but there are yet no RFCs that describe how the personal data of an end user should be handled. In addition to confidentiality, MOSES relies on geoTagging user's data – hence coupling location and content. There exists no standard procedure for geoTagging end user content. Unfortunately, geoTagging end user content could violate the user's privacy by revealing the user's location and activity at a specific point in time. Furthermore in opportunistic sharing environments there should exist procedures for managing the exchange of data between the users. These procedures should provide a means for the end user(s) to have full control over what is shared and how much is shared, while at the same time preventing users from violating another user's rights. Additionally, in a streaming scenario if a user chooses to stream a public video from *n* peers then this user should also share their received & cached video with *n* other users in order to maintain a fairness in the sharing environment.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, “A survey of information-centric networking (draft),” Inf.-Centric Netw., no. 10492, 2011.

[2] S. Farrell, E. Davies, and D. Kutscher, “The NetInf Protocol,” 2013.

[3] “MOSES-Background - MOSES-Background.pdf.” [Online]. Available: https://www.eitictlabs.eu/fileadmin/files/docs/Tenders/MOSES-Background.pdf. [Accessed: 04-Apr-2014].

[4] A. Lindgren, “Open position for Master Thesis work: IoT Monitoring and Analysis System,” SICS Swedish ICT, Thesis proposal. 19-Dec-2013. [Online]. Available: https://www.sics.se/media/news/open-position-for-master-thesis-work-iot-monitoring-and-analysis-system

[5] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design Science in Information Systems Research,” MIS Quarterly, 28(1): 75-105, March-2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.1725&rep=rep1&type=pdf. [Accessed: 26-May-2014].

[6] A. Cockburn, “Using Both Incremental and Iterative Development,” STSC CrossTalk (USAF Software Technology Support Center) 21 (5): 27–30. ISSN 2160-1593, May-2008. [Online]. Available: http://www.crosstalkonline.org/storage/issue-archives/2008/200805/200805-Cockburn.pdf. [Accessed: 30-May-2014].

[7] C. Larman and V. R. Basili, “Iterative and Incremental Development: A Brief History,” IEEE Computer Society, 2003. [Online]. Available: http://www.computer.org/csdl/mags/co/2003/06/r6047-abs.html. [Accessed: 26-May-2014].

[8] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, “LIPSIN: line speed publish/subscribe inter-networking,” in ACM SIGCOMM Computer Communication Review, 2009, vol. 39, pp. 195–206.

[9] B. Ahlgren, M. D’Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone, “Design considerations for a network of information,” in Proceedings of the 2008 ACM CoNEXT Conference, 2008, p. 66.

[10] S. Nechifor, “How to Manage and Search/Retrieve Information Objects,” in Architecture and Design for the Future Internet, Springer, 2011, pp. 201–223.

[11] S. Farrell, C. Dannewitz, P. Hallam-Baker, D. Kutscher, and B. Ohlman, “Naming things with hashes,” Internet Request for Comments, RFC 6920 (Proposed Standard), Apr. 2013 [Online]. Available: http://www.rfc-editor.org/rfc/rfc6920.txt

[12] D. Kutscher, S. Farrell, and E. Davies, “draft-kutscher-icnrg-netinf-proto-01 - The NetInf Protocol,” 10-Feb-2013.

[13] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking named content,” in Proceedings of the 5th international conference on Emerging networking experiments and technologies, 2009, pp. 1–12.

[14] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A data-oriented (and beyond) network architecture,” ACM SIGCOMM Comput. Commun. Rev., vol. 37, no. 4, pp. 181–192, 2007.

[15] E. Davies, “Implementing the NetInf Protocol with HTTP and DTN Convergence Layers and Using NetInf over DTN as the Primary Communication Protocol for a Device,” 14-Jun-2013. [Online]. Available: http://developer.android.com/training/connect-devices-wirelessly/wifi-direct.html. [Accessed: 29-May-2014].

[16] L. Sunde, “NetInf Node for Bluetooth Enabled Android Devices,” Uppsala Universitet, Department of Information Technology, IT 13 081, 2013, http://uu.diva-portal.org/smash/get/diva2:668342/FULLTEXT01.pdf.

[17] “Network of Information / Code / [e93120].” [Online]. Available: http://sourceforge.net/p/netinf/code/ci/default/tree/. [Accessed: 31-Mar-2014].

[18] M. Ion, J. Zhang, and E. M. Schooler, “Toward Content-Centric Privacy in ICN: Attribute-based Encryption and Routing,” In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (SIGCOMM '13). ACM, New York, NY, USA, Aug-2013, pp. 513-514. DOI=10.1145/2486001.2491717 Available: http://conferences.sigcomm.org/sigcomm/2013/papers/sigcomm/p513.pdf. [Accessed: 28-May-2014].

[19] “Reality matters — Preorder for Estimote Beacons available, shipping this summer.” [Online]. Available: http://blog.estimote.com/post/57087851702/preorder-for-estimote-beacons-available-shipping-this. [Accessed: 04-Apr-2014].

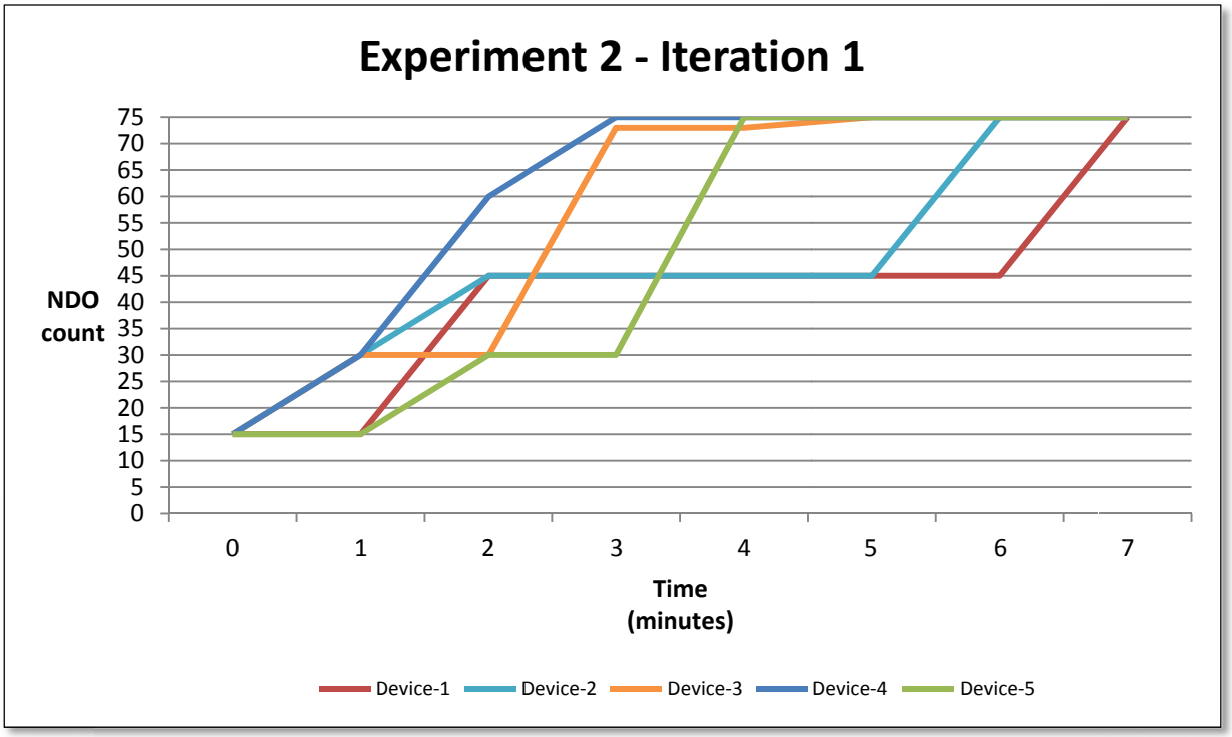
[20] “Bluetooth | Android Developers.” [Online]. Available: http://developer.android.com/guide/topics/connectivity/bluetooth.html. [Accessed: 17-May-2014].

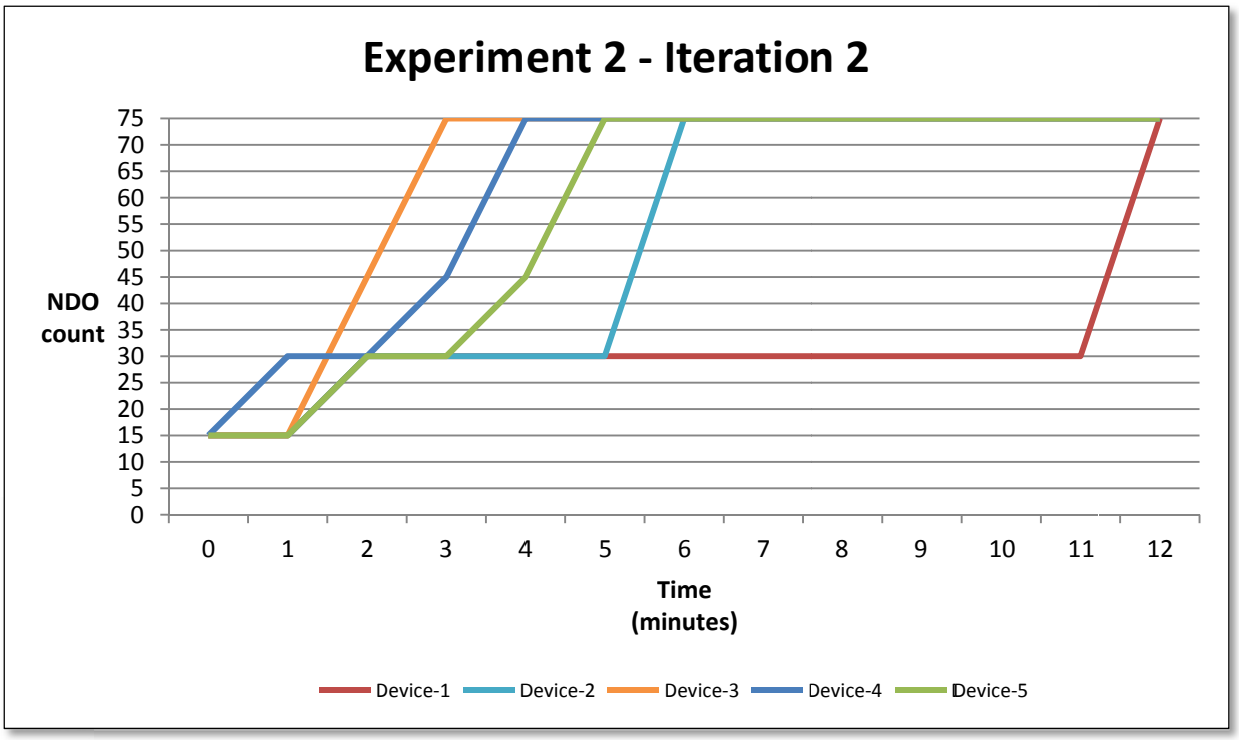
[21] Bluetooth SIG, “Baseband Architecture | Bluetooth Development Portal,” Baseband Architecture | Bluetooth Development Portal, 26-May-2014. [Online]. Available: https://developer.bluetooth.org/TechnologyOverview/Pages/Baseband.aspx. [Accessed: 26-May-2014].

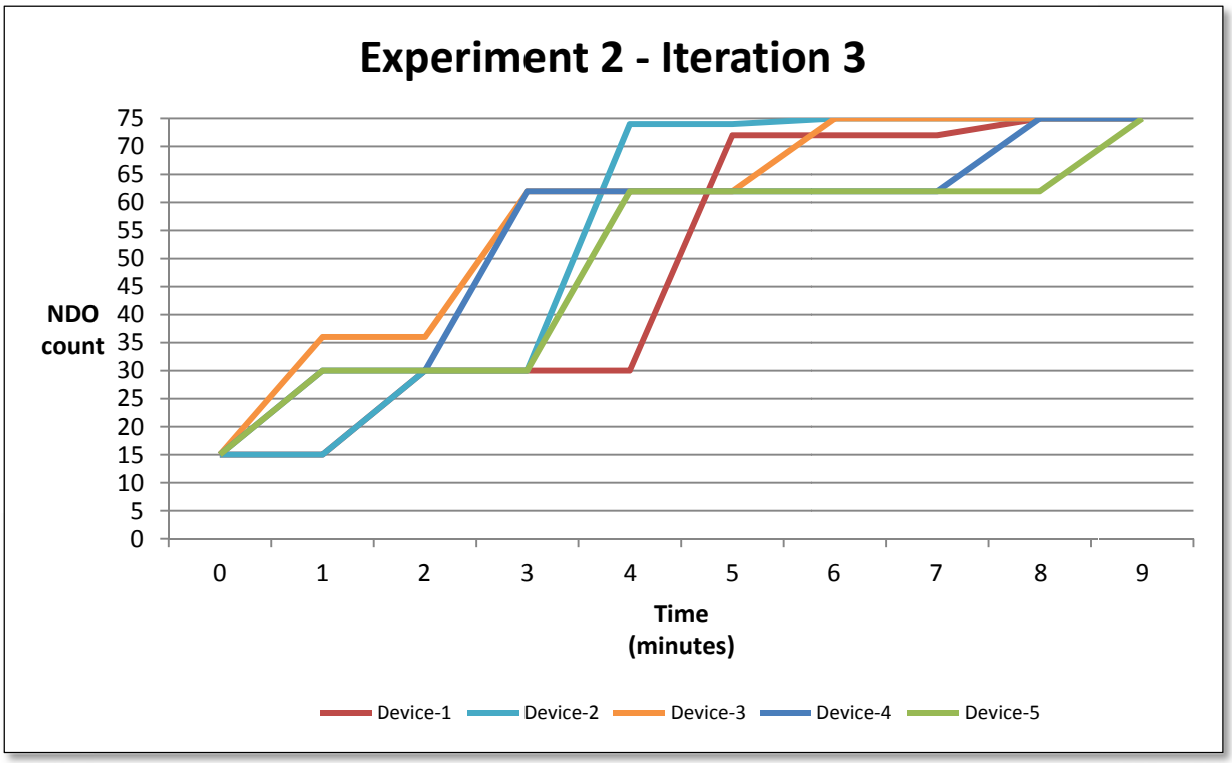
[22] “Specification of the Bluetooth System, Covered Core Package version: 4.1.” http://www.bluetooth.com, 03-Dec-2013.

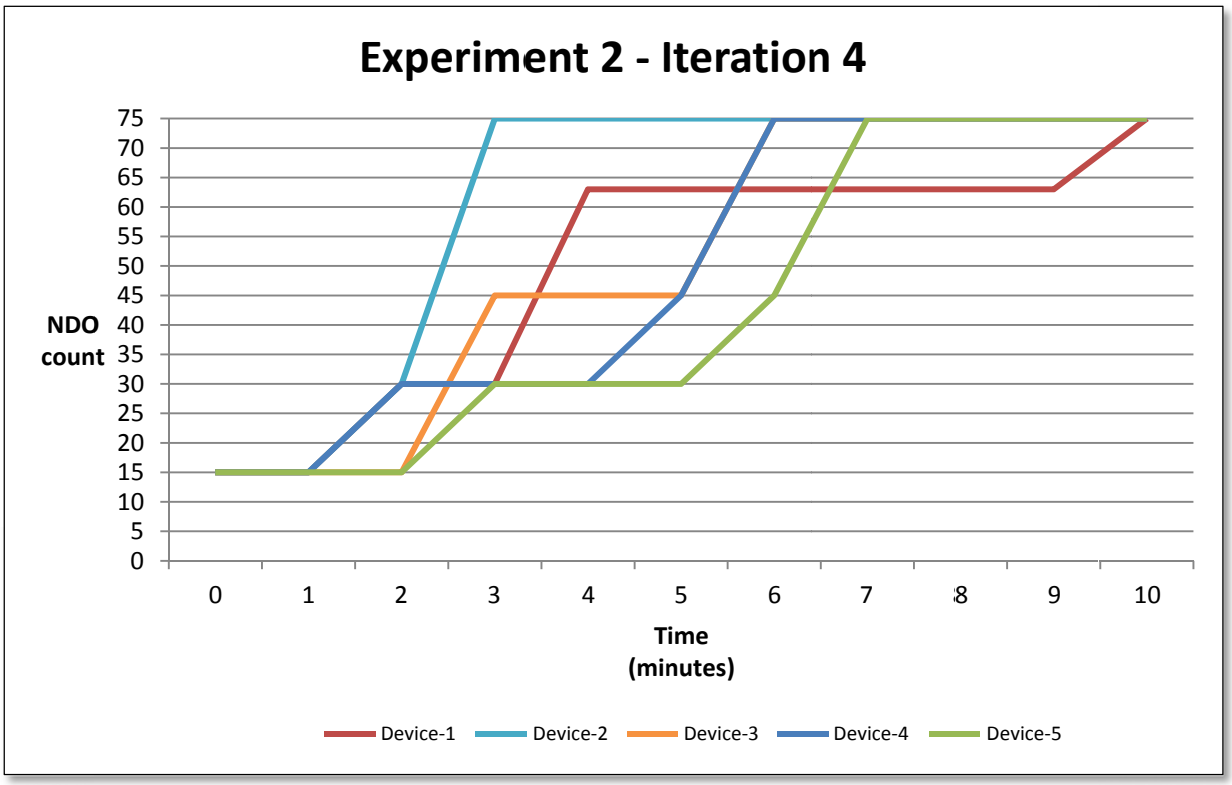
[23] “Wi-Fi Peer-to-Peer | Android Developers.” [Online]. Available: http://developer.android.com/guide/topics/connectivity/wifip2p.html. [Accessed: 29-May-2014].

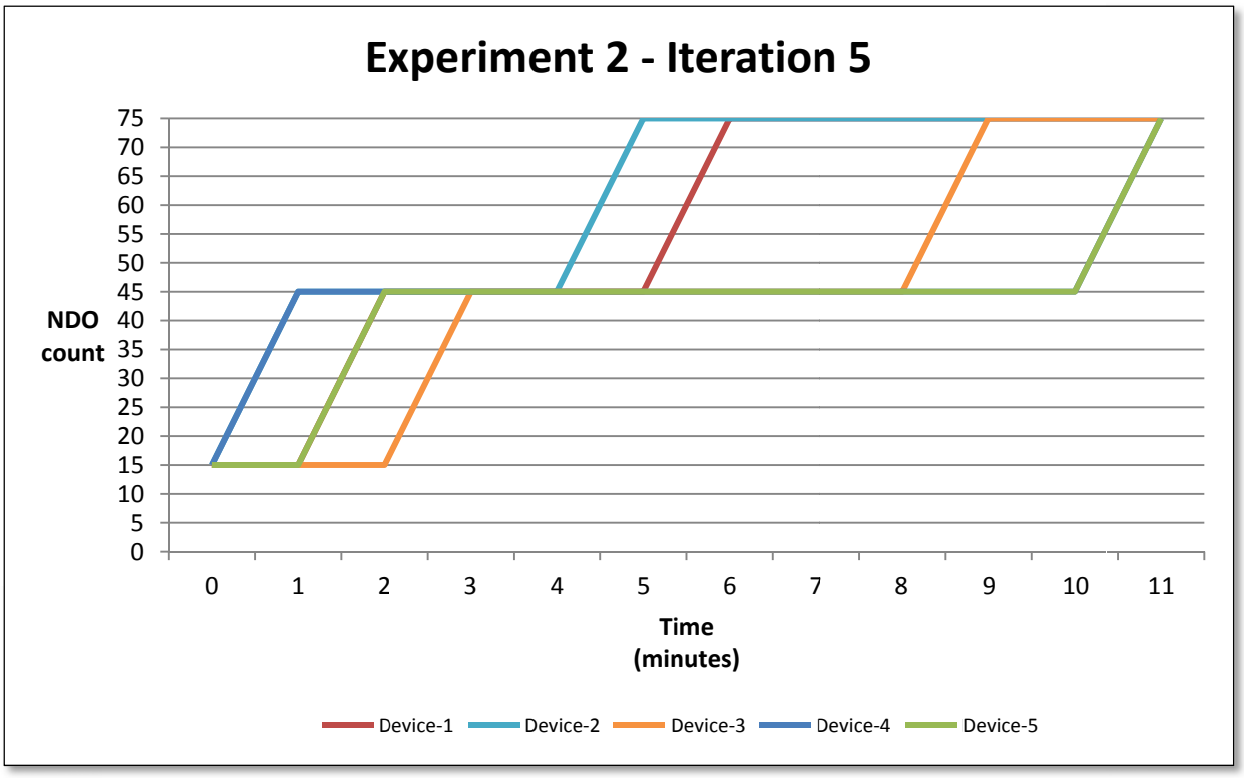
Appendix A. Experiment 2 - Device convergence time

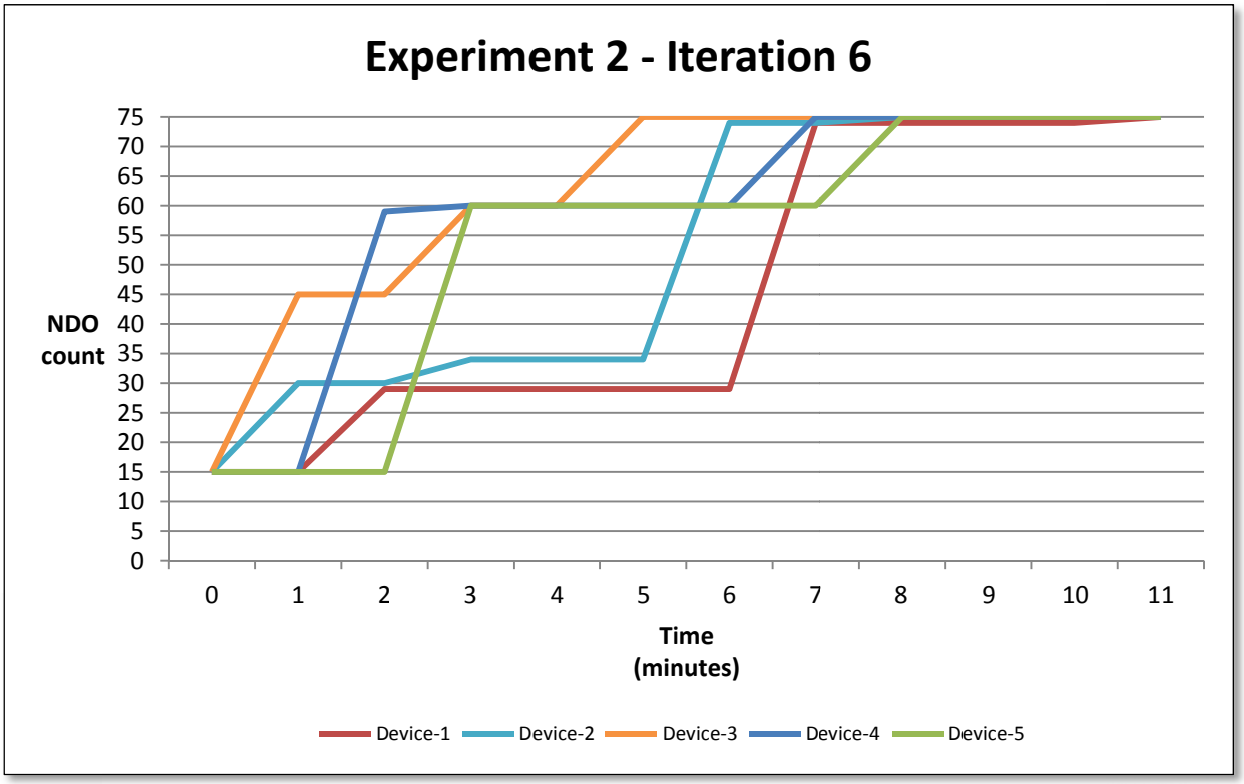


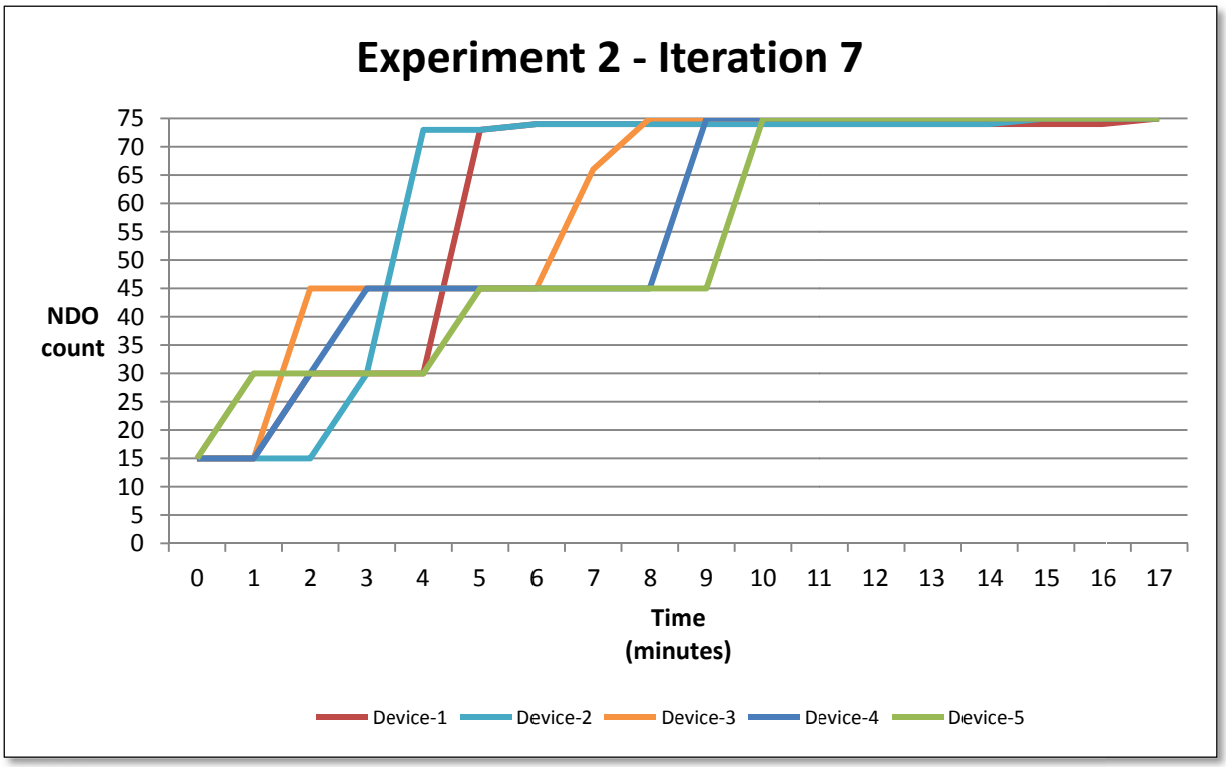


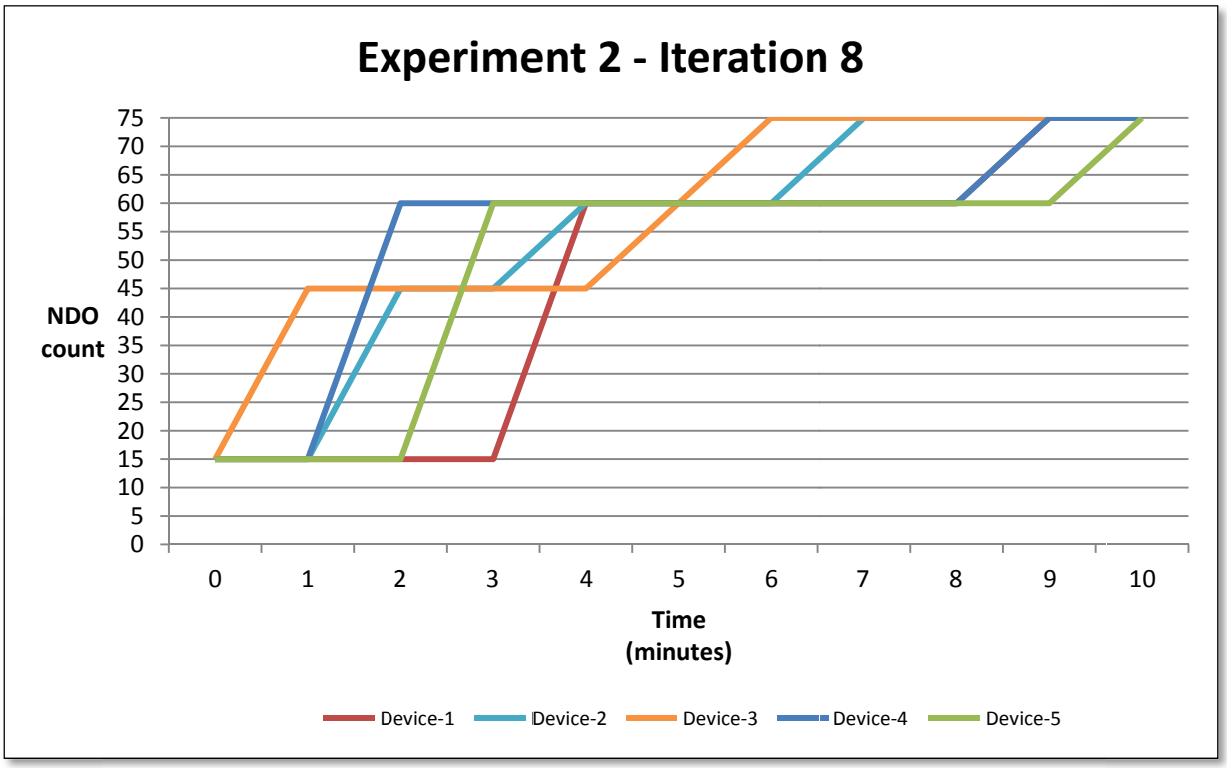

Experiment 2 - Iteration 3
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

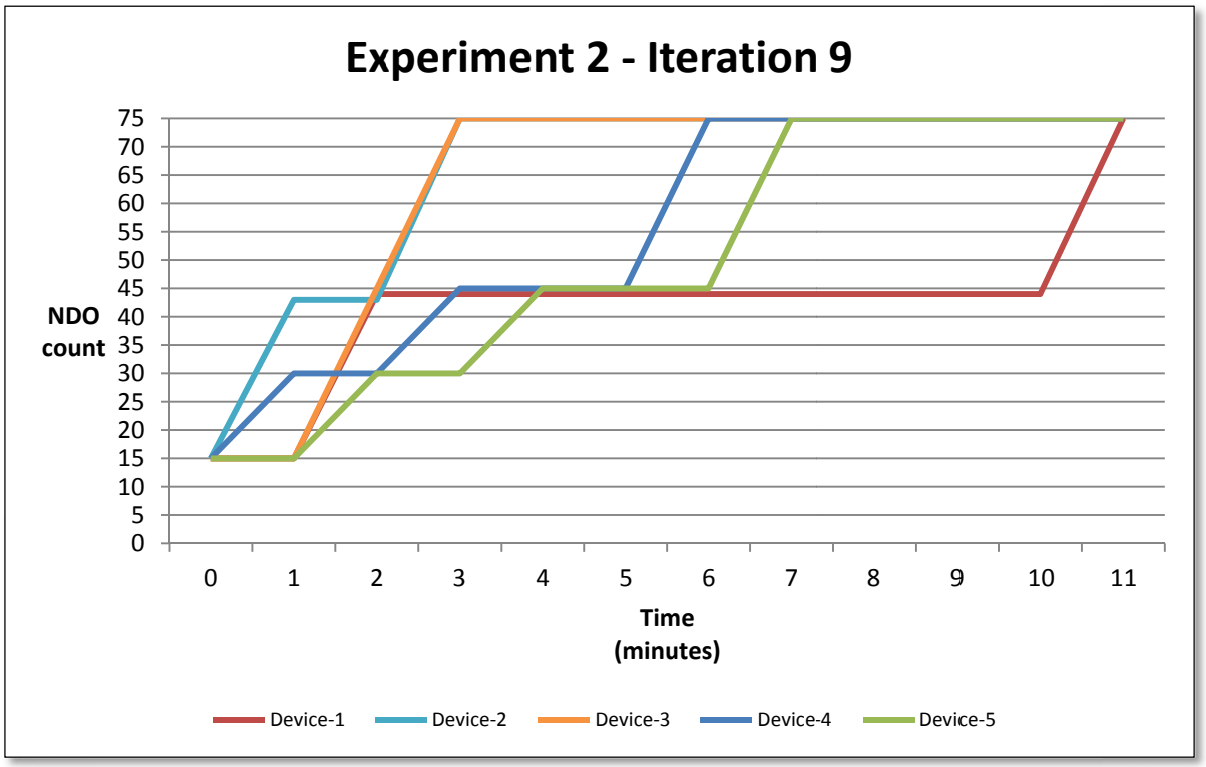

Experiment 2 - Iteration 4
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

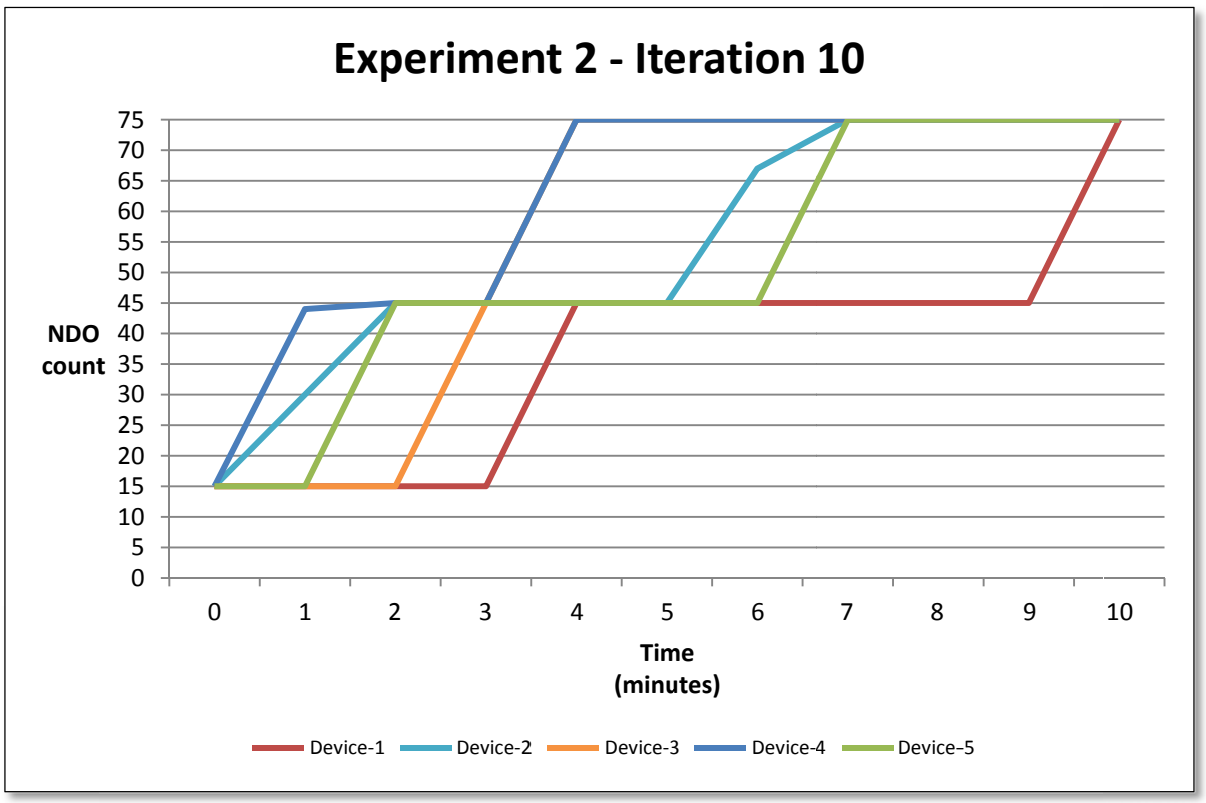

Experiment 2 - Iteration 5
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
11
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5


Experiment 2 - Iteration 6
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
11
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

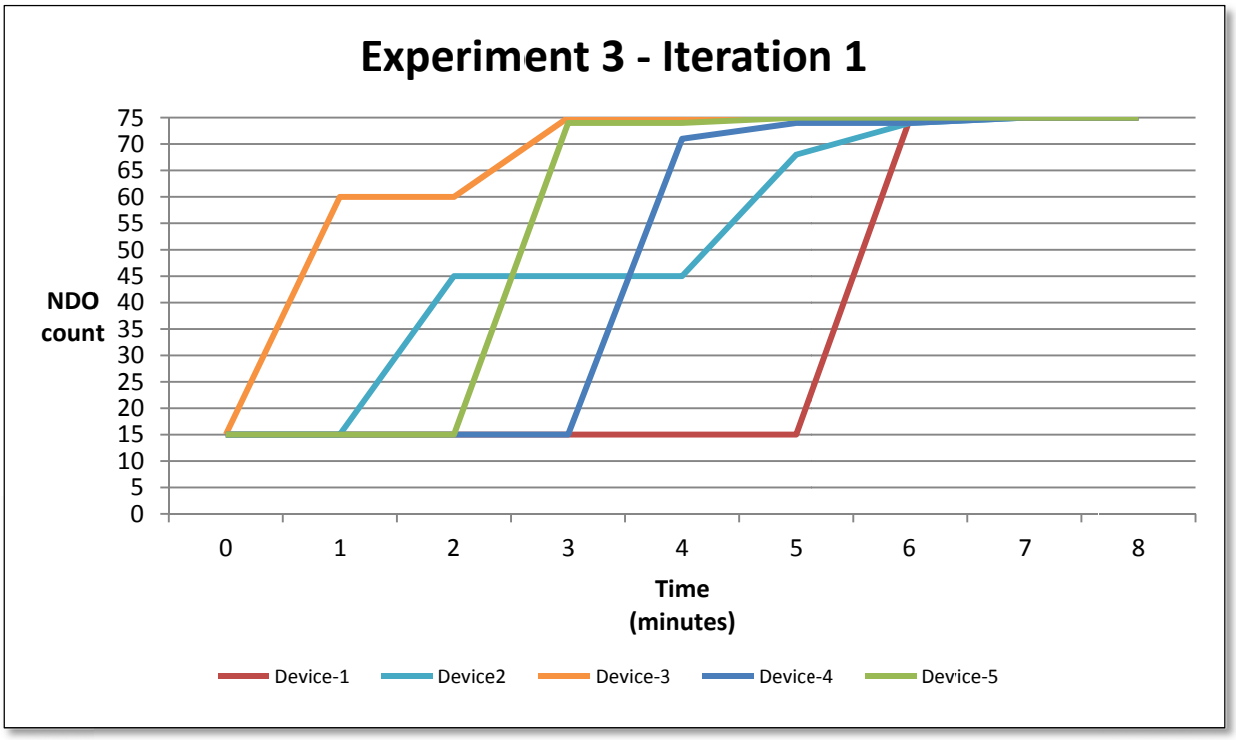

Experiment 2 - Iteration 7
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

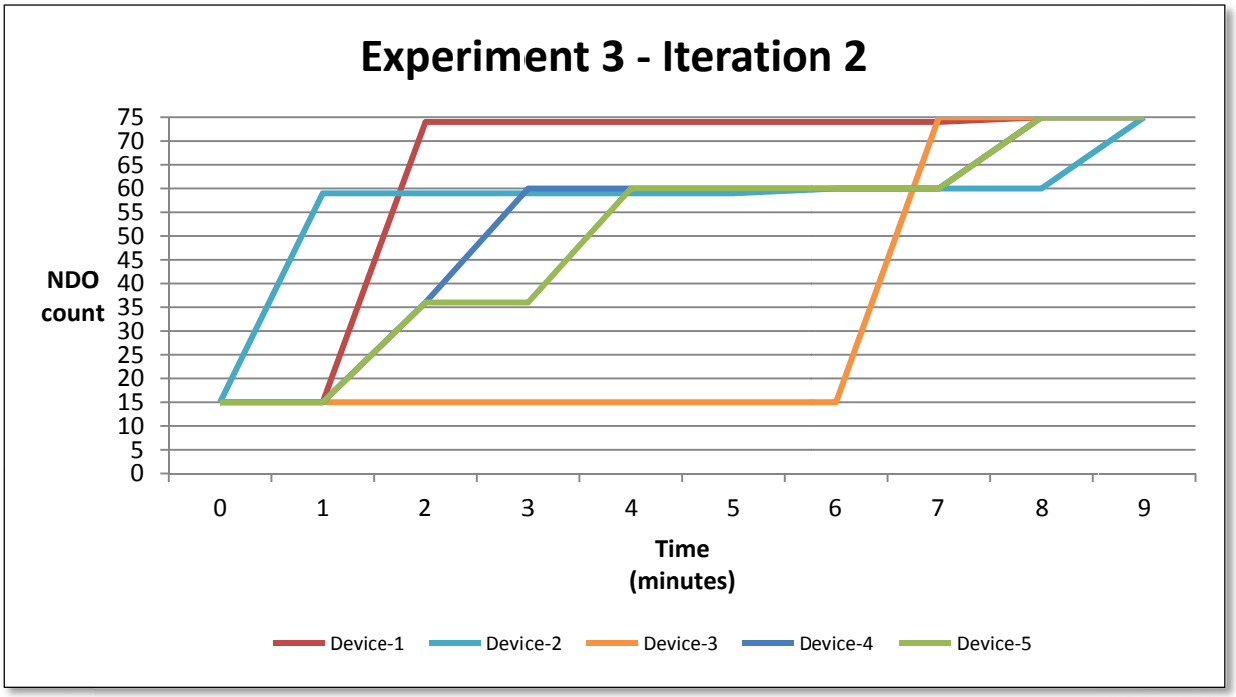

Experiment 2 - Iteration 8
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9 10
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

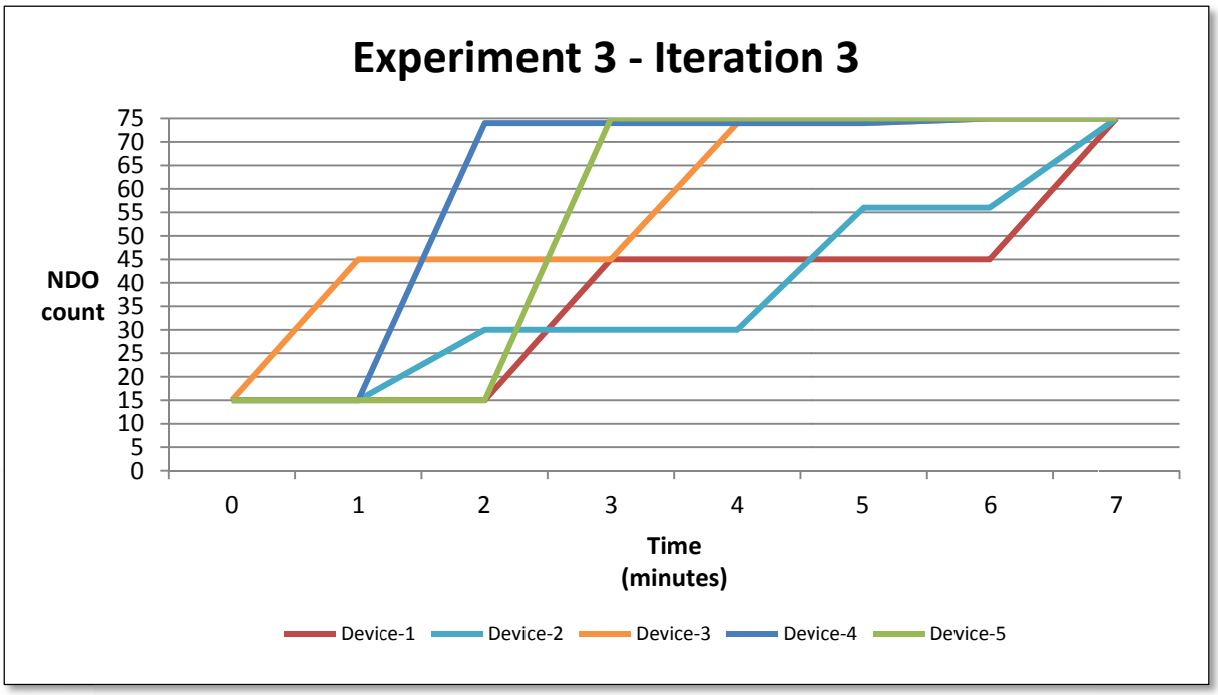

Experiment 2 - Iteration 9
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9 10 11
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

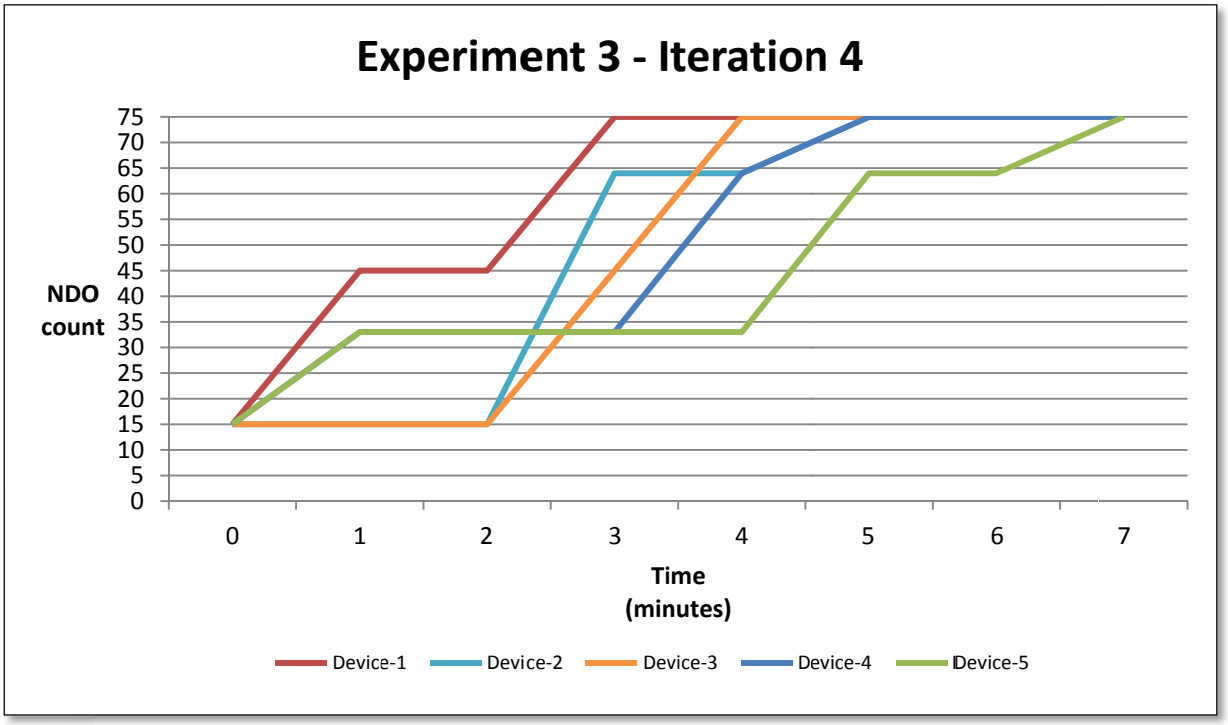

Experiment 2 - Iteration 10
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0 1 2 3 4 5 6 7 8 9 10
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

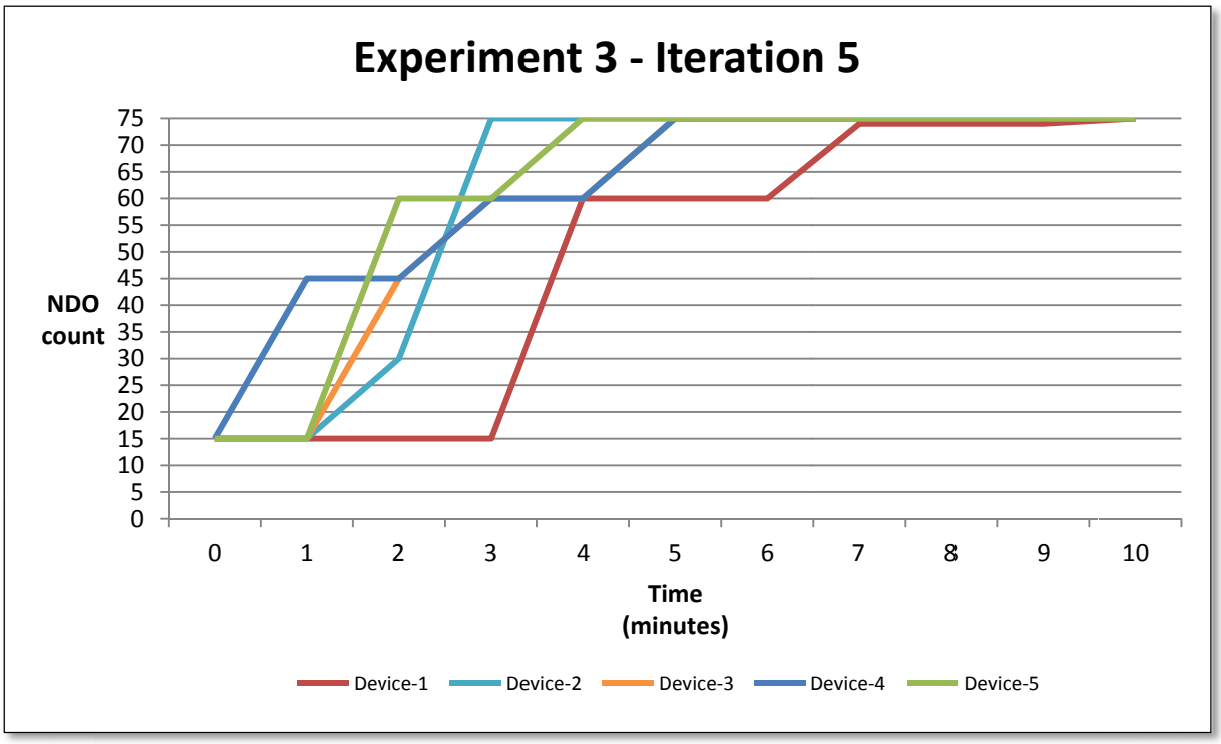
Appendix B. Experiment 3 - Device convergence time

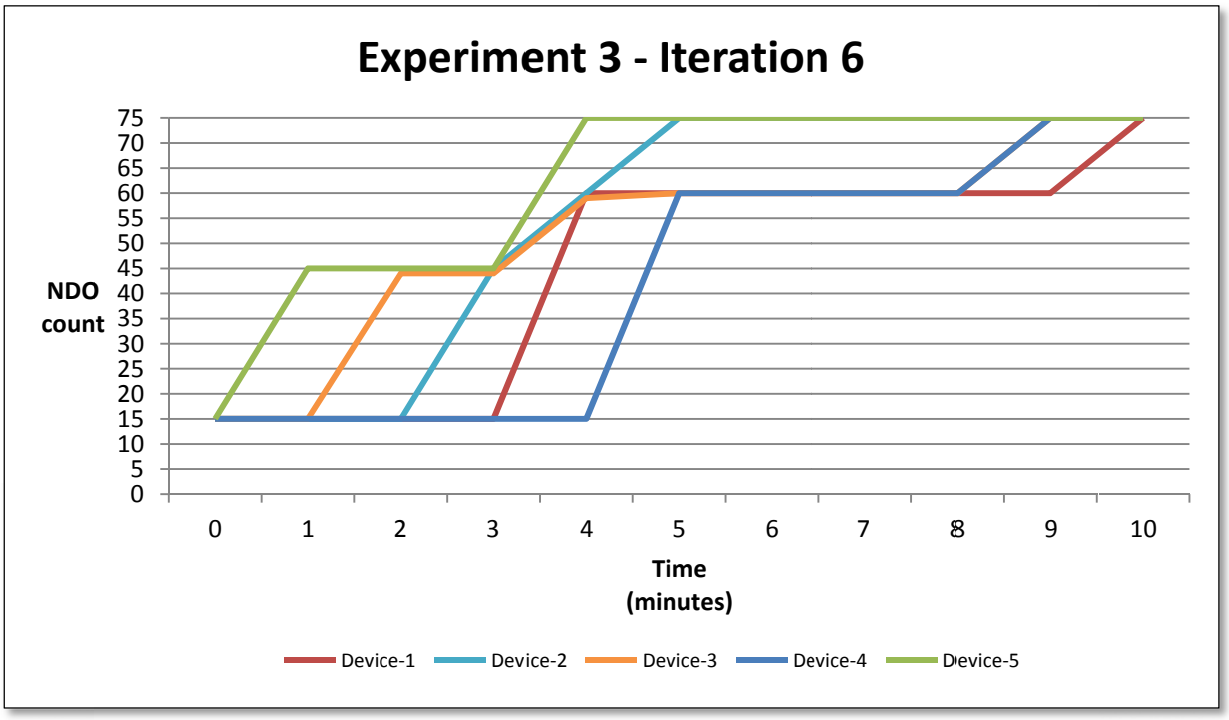


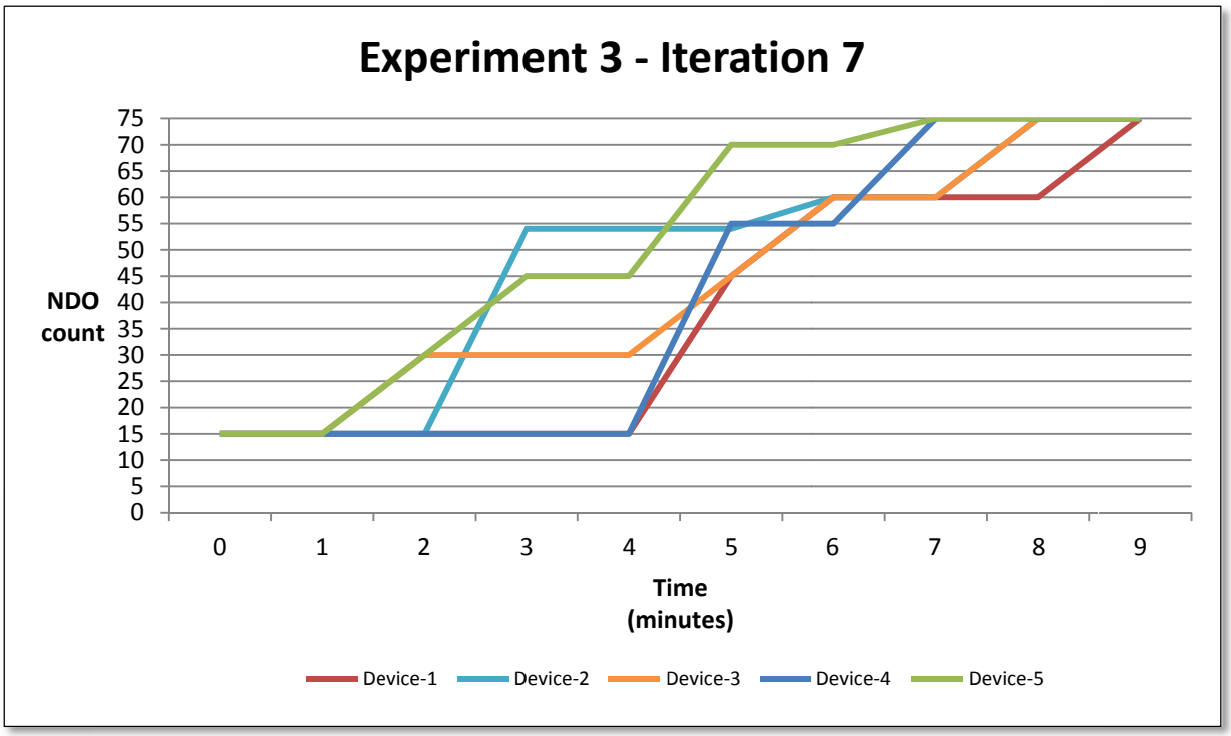


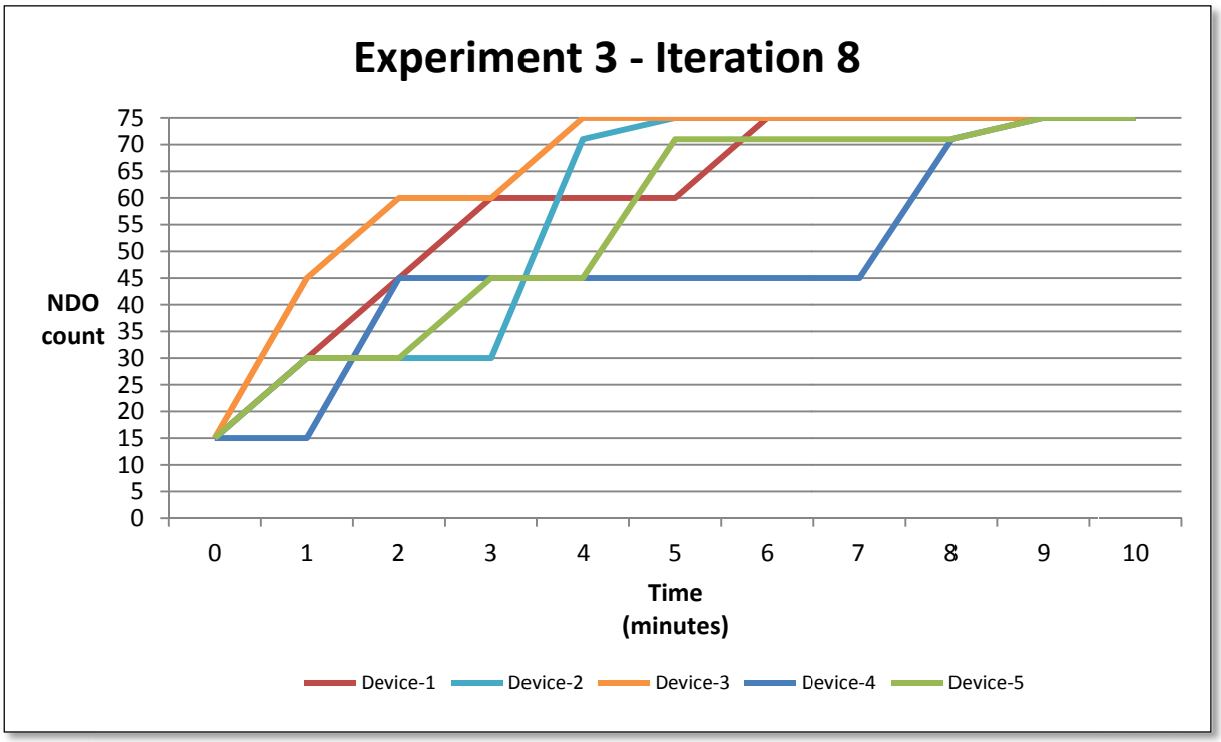

Experiment 3 - Iteration 3
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

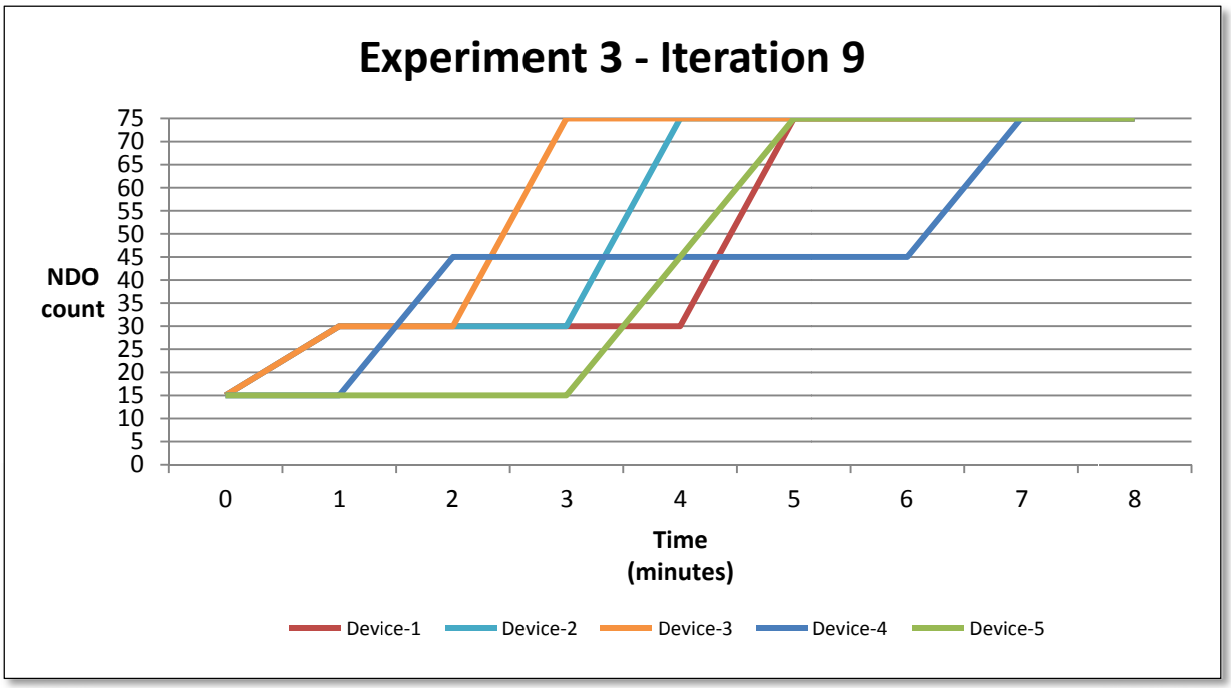

Experiment 3 - Iteration 4
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

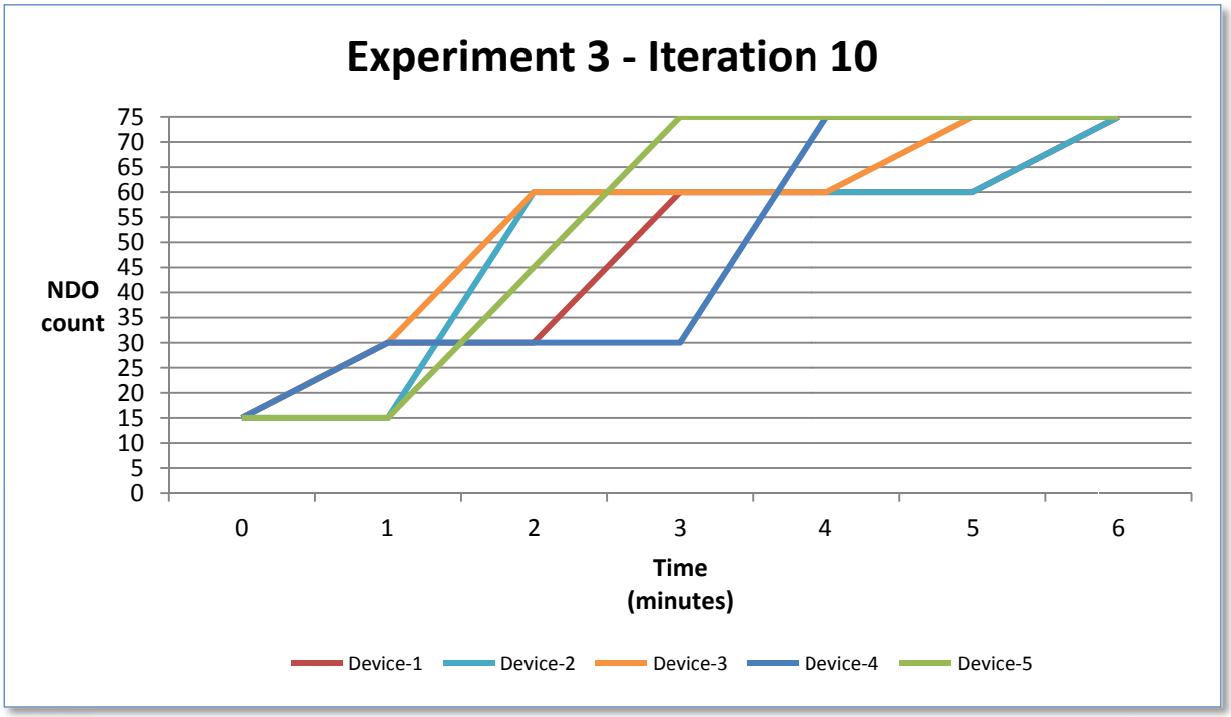

Experiment 3 - Iteration 5
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5


Experiment 3 - Iteration 6
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5


Experiment 3 - Iteration 7
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5


Experiment 3 - Iteration 8
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5


Experiment 3 - Iteration 9
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5


Experiment 3 - Iteration 10
NDO count
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
Time (minutes)
Device-1
Device-2
Device-3
Device-4
Device-5

TRITA-ICT-EX-2014:58