Master Thesis

Using Multiple Transport Networks in NetInf Enabled Android Devices

TRITA-ICT-EX-2012-261

by

Hugo Negrette and Miguel Sosa

Supervisors

Ericsson
Karl Åke Persson
Börje Ohlman
Anders Eriksson

KTH
Flutra Osmani
Björn Knutsson

KTH Information and Communication Technology
Forum 105 SE-164 40 Kista
SWEDEN
Abstract

In recent years, there has been a significant increase of mobile devices usage (e.g. phones, tablets), consequently the demand of network capacity is higher than ever before. In large crowd events (such as football matches, concerts, accidents, etc), current content distribution is done in a host-centric manner where users have to retrieve content from external servers using the 3G-Uplink. This result in problems like Up-link congestion when there is an unexpected boost in data traffic.

The prototype developed within this master thesis project, implements a new control layer in mobile devices, which can be used in events with large crowds as an alternative solution for optimizing content distribution. The usage of this solution decreases the amount of data sent over the 3G-uplink to retrieve similar contents, since they can be obtained from nearby peers using short range transport technologies.

The alternative solution is based on information-centric networking paradigm. Using this approach in Large Crowd events, popular content could be cached in users’ devices creating new sources of content within the crowd. Afterwards, these copies could be shared between users using P2P-like communication over short range transport technologies available in nowadays’ mobile devices. In this new content distribution scheme, the more popular the content gets the more chances it could be retrieved from a cached copy within the crowd and, this could lead to a less amount of data transferred over the 3G uplink.
Acknowledgements

We would like to start thanking our Ericsson supervisor Börje Ohlman, Karl Åke Persson and Anders Eriksson for giving us the opportunity to perform our Master Thesis project within the Ericsson packet research technology group in Kista (Sweden). Thanks for all the support, suggestions and guiding; without those inputs we would not have gotten the great results obtained throughout the project. We appreciate the trust and freedom that we had from the beginning to propose our own ideas and to participate in the main decisions taken along the project. Thanks also for providing us with the necessary resources needed to develop the project (mobile devices, servers, computers, etc.).

Special thanks to Börje and Karl for recommending us in our current employment positions within Ericsson. Thanks to all the other colleagues in the department, it was a pleasure to work with you during these 6 months. Thank for the good Fika times.

We would like to show our gratitude to our KTH supervisor and examiner, Flutra Osmani and Björn Knutsson, for supporting us from the beginning of the project. Thanks for guiding us throughout all the KTH mandatory steps and providing us with all the necessary information and requirements to have a successfully Master Thesis project. We would like to stress that both of them really put special effort to carefully read the entire thesis report, highlighting the strengths and weaknesses.

Thanks also to our families in Venezuela, who despite of the distance, always showed unconditional support in the different parts of the projects. We would also like to thanks our friends in Stockholm for their support during this period. Special thanks to Cristina, Tony and Belgis for showing us their friendship throughout the master program (great Nordic experience!!).
# Table of Contents

List of Figures ............................................................................................................................................. 7  
List of Tables ................................................................................................................................................ 9  

1. Introduction .............................................................................................................................................. 10  
   1.1 Acronyms ............................................................................................................................................... 10  
   1.2 Problem ............................................................................................................................................... 10  
   1.3 Background ......................................................................................................................................... 10  
      1.3.1 Large Crowd Events ......................................................................................................................... 11  
      1.3.2 Peer-to-Peer .................................................................................................................................... 12  
      1.3.3 Host-centric network paradigm ...................................................................................................... 12  
      1.3.4 Information Centric Networks ....................................................................................................... 13  
   1.4 Motivation ............................................................................................................................................. 14  
   1.5 Hypothesis ........................................................................................................................................... 15  
   1.6 Goal ...................................................................................................................................................... 15  
   1.7 Objectives ............................................................................................................................................ 15  
   1.8 Task ..................................................................................................................................................... 15  
   1.9 Approach ............................................................................................................................................ 15  
   1.10 Opportunities .................................................................................................................................... 16  
   1.11 Target Audience ................................................................................................................................. 16  

2. Background and Related Work .................................................................................................................. 17  
   2.1 Previous ICN Approaches ..................................................................................................................... 17  
      2.1.1 Data Oriented Network Architecture (DONA) .............................................................................. 17  
      2.1.2 Content Centric Networking (CCN) ............................................................................................... 17  
      2.1.3 Publish/Subscribe Internet Routing Paradigm (PSIRP) ............................................................... 18  
   2.2 Networking of Information (NetInf) ..................................................................................................... 19  
   2.3 Comparison between the different ICN approaches ............................................................................ 21  
   2.4 Technologies and Programming Tools ............................................................................................... 23  
      2.4.1 Android OS ..................................................................................................................................... 23  
      2.4.2 Bluetooth ...................................................................................................................................... 23  
      2.4.3 Wi-Fi Direct ................................................................................................................................... 25  
      2.4.4 Java ............................................................................................................................................... 26  
      2.4.5 Eclipse ........................................................................................................................................ 26
3. OpenNetInf Platform ................................................................................................. 29
   3.1 OpenNetInf Components ...................................................................................... 29
      3.1.1 Node ........................................................................................................... 29
      3.1.2 Event Service .............................................................................................. 29
      3.1.3 Application and tools .................................................................................. 30
   3.2 Node Structure ...................................................................................................... 30
   3.3 Data Model ........................................................................................................... 31
   3.4 Node Interface ...................................................................................................... 32
   3.5 Application Interface ........................................................................................... 33
      3.5.1 Restful HTTP .............................................................................................. 33
   3.6 Resolution ............................................................................................................ 35
      3.6.1 BO Handling ............................................................................................... 36
   3.7 Porting OpenNetInf to Mobile Devices (Android OS) ............................................ 37
4. Implementation ............................................................................................................ 43
   4.1 Architecture Overview ......................................................................................... 43
      4.1.1 Application Layer ....................................................................................... 44
      4.1.2 NetInf Control Layer .................................................................................. 48
      4.1.3 Internal Cache ............................................................................................. 57
      4.1.4 Name Resolution Server ............................................................................ 59
      4.1.5 Ni Content Server ...................................................................................... 62
      4.1.6 QR code Server ......................................................................................... 64
      4.1.7 Web Server (IIS7) ...................................................................................... 68
   4.2 Code Overview ...................................................................................................... 69
5. Prototype Evaluation .................................................................................................. 70
   5.1 Connection Time .................................................................................................... 70
      5.1.1 Connection time test case ............................................................................ 70
   5.2 File Transfer Time ............................................................................................... 74
      5.2.1 File transfer time test case ......................................................................... 74
   5.3 Prototype interaction with external components .................................................. 77
5.3.1 Interaction with the NRS and the NCS test case ................................................................. 77
6. Results ........................................................................................................................................ 84
   6.1 OpenNetinf in Mobile Devices ......................................................................................... 84
   6.2 Architecture overview ....................................................................................................... 84
       6.2.1 Internal components .................................................................................................. 84
       6.2.2 External components ............................................................................................... 85
   6.3 Short range transport technologies .................................................................................. 86
   6.4 Prototype Evaluation ......................................................................................................... 86
7. Conclusion .................................................................................................................................. 87
8. Future Work ............................................................................................................................... 88
   8.1 OpenNetInf Platform .......................................................................................................... 88
   8.2 Wi-Fi Direct stability .......................................................................................................... 88
   8.3 Media files supported ......................................................................................................... 88
   8.4 QR code server improvement ............................................................................................. 88
   8.5 Control layer implementation .............................................................................................. 88
   8.6 Further areas of study ......................................................................................................... 88
9. Bibliography ............................................................................................................................... 89
10. Appendix .................................................................................................................................... 93
    10.1 List of Acronyms .............................................................................................................. 93
    10.2 Demo ................................................................................................................................ 94
    10.3 Code Package Overview .................................................................................................. 98
    10.4 UML Diagrams ............................................................................................................... 102
        10.4.1 Bluetooth Discovery Task ........................................................................................ 102
        10.4.2 WIFI Discovery and connection task ........................................................................ 103
        10.4.3 GetIO ....................................................................................................................... 104
        10.4.4 PutIO ....................................................................................................................... 108
List of Figures

Figure 1. Event with large crowd scenario.................................................................12
Figure 2. Host-Centric Networks ............................................................................13
Figure 3. Information-Centric Networks ................................................................14
Figure 4. Different NetInf domains with different underlying technologies example [16] ....20
Figure 5. Request and Publish functions in a NetInf domain [16] .............................20
Figure 6. SAIL NetInf name example ......................................................................21
Figure 7. Google Guice example .............................................................................27
Figure 8. Components within NetInf Architecture [39]..........................................29
Figure 9. Node Structure [39] ................................................................................30
Figure 10. OpenNetInf Identifier example ................................................................31
Figure 11. Relationship between different IO types ..................................................32
Figure 12. OpenNetInf’s inter-components communication [39] ............................33
Figure 13. IO resource URI Pattern .........................................................................34
Figure 14. IO resource redirect URI pattern ..............................................................34
Figure 15. BO resource URI Pattern .........................................................................34
Figure 16. BO resource redirect URI pattern ..............................................................34
Figure 17. OpenNetInf’s Resolution Process ..............................................................35
Figure 18. BO handling process ..............................................................................36
Figure 19. Architecture proposed ...........................................................................43
Figure 20. Architecture application layer components .............................................45
Figure 21. Discovery process example ....................................................................47
Figure 22. Architecture control layer components ....................................................49
Figure 23. HTTP server processing different types of requests ..................................50
Figure 24. Local resolution service putting and IO into the MDB ...............................51
Figure 25. Remote Name Resolution Service putting and IO in the NRS using HTTP POST ....52
Figure 26. Use of the locator priority by the Locator Selector ..................................54
Figure 27. Use of the locator priority by the Locator Selector ..................................56
Figure 28. Use of the locator priority by the Locator Selector ..................................57
Figure 29. Use of the MDS and MDB .....................................................................58
Figure 30. NRS (Publish requests from two different Android NetInf nodes) .........59
Figure 31. NRS (GET requests and GET/Response request) .....................................60
Figure 32. Ni Content Server (Publish requests, including content) ........................................ 63
Figure 33. Ni Content Server (GET response) ........................................................................ 64
Figure 34. QRCode Server ........................................................................................................ 65
Figure 35. Form of the request redirected to the BOserver resource ..................................... 65
Figure 36. Example of a request redirected to the BO server resource .................................... 66
Figure 37. .txt file created from the entrance showed in Figure 36 ........................................ 66
Figure 38. QRcode generated from the request showed in Figure 36 ...................................... 66
Figure 39. wwwroot folder where all the website content is stored ......................................... 67
Figure 40. Form of the request redirected to the Delete server resource ................................. 67
Figure 41. Example of a request redirected to the Delete server resource ............................... 67
Figure 42. Example of an object with two MAC address associated ....................................... 67
Figure 43. Result after the delete request showed in Figure 41 ................................................. 68
Figure 44. URL Rewrite ............................................................................................................ 69
Figure 45. Delete HTTP request redirected to QR server ........................................................ 69
Figure 46 Bluetooth connection time test case setup ............................................................... 71
Figure 47 Wi-Fi Direct connection time test case setup ........................................................... 71
Figure 48 Connection time comparison between Bluetooth v2.1 +EDR, Bluetooth v3.0 and WIFI Direct ................................................................................................................... 72
Figure 49 File transfer time test case setup ............................................................................... 74
Figure 50 Average file transfer time for the different transport technologies .......................... 75
Figure 51 Connection and transfer times for Bluetooth v3.0 .................................................... 76
Figure 52 Connection and transfer times for Wi-Fi Direct ........................................................ 76
Figure 53 Interaction with external components test case ....................................................... 78
Figure 54 Bandwidth consumption in NCS and NRS when running the “Interaction with external components” test case ........................................................ 79
Figure 55 Content retrieval from a web server using HCN approach ....................................... 80
Figure 56. HTTP response from the NRS to the mobile devices with the Locators ............... 83
Figure 57. HTTP get request sent from the mobile device to the NCS to retrieve the data ...... 83
Figure 58. Link to the application ............................................................................................. 85
Figure 59. UI Android Demo ................................................................................................... 94
Figure 60. Publishing and Retrieving a File ............................................................................... 96
List of Tables

Table 1. Main differences between the ICN approaches .......................................................... 22
Table 2. Data transfer rate of Bluetooth 2.1 + EDR and Bluetooth 3.0 +HS [27] ....................... 24
Table 3. Attributes used in the IOs .......................................................................................... 37
Table 4. HTTP methods used depending on the target component of the control layer .......... 46
Table 5. Suitable providers for the different locator prefixes ................................................... 53
Table 6. MDB table description .............................................................................................. 58
Table 7. GET request parameters (HTTP POST) .................................................................... 60
Table 8. Response code for GET requests ................................................................................ 61
Table 9. PUBLISH requests parameters (HTTP POST) .......................................................... 61
Table 10. Response code for PUBLISH requests ................................................................... 61
Table 11. UNPUBLISH requests parameters (HTTP POST) .................................................... 62
Table 12. Response code for UNPUBLISH requests ................................................................. 62
Table 13 Equipment used for the Connection time test case .................................................... 71
Table 14. Average times for Figure X and figure Y (File size = 2Mbytes) ................................. 77
Table 15 Total times for obtaining the files from a nearby peer ................................................. 82
1. Introduction

1.1 Acronyms
The list of acronyms used throughout the master thesis report is presented in [Appendix 0].

1.2 Problem
In large crowd events (e.g. football matches, concerts, accidents) [see 1.3.1], users tend to experience a decrease in the perceived quality of service due to uplink congestion. The participants in these types of events usually share a common interest (e.g. the band playing in the concert, the same football team, etc.) and there is a high possibility that users request the same content tens of times over the uplink; sometimes even standing right next to each other. This considerable amount of users using the same uplink in order to retrieve over and over the same content could be seen as an inefficient usage of the uplink leaving space for possible alternative solutions.

1.3 Background
When the Internet was invented, the main focus was the ability of connecting remotely located devices to create a communication channel between them for different purposes like file transferring and message exchange. The main idea was therefore to establish a reliable end-to-end byte stream between hosts [1]. Since this communication model was centered in hosts, it was said to be host-centric network (section 1.3.3).

In today’s Internet, accessing information (e.g. videos, pictures, etc.) seems to be the main focus of most of the users. In a research performed by Ipoque\(^1\) on the Internet traffic in 2008/2009, file sharing over P2P networks represented the most common traffic on the network. According to this research: “Peer-to-peer file sharing (P2P) still generates by far the most traffic in all monitored regions – ranging from 43 percent in Northern Africa to 70 percent in Eastern Europe” [2]. In a more recent research, the Cisco Visual Networking Index predicted that: “by 2012 Internet video will account for over 50 percent of consumer Internet traffic.” [3].

These studies on Internet’s traffic have made the research community to start thinking the possible benefits of shifting from the current host centric towards information centric network architecture (section 1.3.4). A number of information centric proposals and models have emerged in recent years (section 2): Content Centric Networking (CCN) [4], Data Oriented Network Architecture (DONA) [5], Publish/Subscribe Internet Routing Paradigm (PSIRP) [6], Networking of Information (NetInf) [7], etc. All these different proposals have different ideas, architecture components, advantages and disadvantages; however, all share the same goal: to create a network architecture focused in the information itself. This information centric architecture could fit better than a host centric architecture to the current internet traffic trends [1].

\(^1\) http://www.ipoque.com/
Another revolution has also been happening in recent years. Nowadays, mobile-cellular telephones have become the most common way to access the internet. According to the Facts and Figures report of the International Telecommunication Unit (ITU) in 2011: “mobile-broadband subscriptions have grown 45% annually over the last four years and today there are twice as many mobile-broadband as fixed broadband subscriptions” [8]. Even though operators are aware of this increase, mobile networks are designed to provide a fixed capacity per area, which is usually an average estimation of the amount of users in that particular location. Therefore, during special events where a considerable number of users gather together in the same location (e.g. concerts, football matches, demonstrations, protests, accidents, etc.), users tend to experience a decrease in the quality of the mobile service [9] [10]. A research conducted by CommProve\(^2\), over a sample of 2000 mobile users in the United Kingdom, found that “one in three people doesn’t expect access to 3G services such as Facebook, Twitter, or web browsing at the Summer’s major events.” [11]. One of the reasons of this decrease in the quality of service is that, in this type of crowded scenarios, every user sharing a common operator has to use the same up-link to request and retrieve the desired information to/from the corresponding server. Since most of the users tend to share a common interest, there is a considerable possibility that the same content is retrieved several times over the same uplink.

The Information Centric Network (ICN) approach has been proposed as a possible solution to the Large Crowd event and to other typical host-to-host centric networks problems [12]. Using an ICN approach in Large Crowd events, popular content could be cached in users’ devices creating new sources of the content within the crowd. Afterwards, these copies could be shared between users using for example P2P-like communication over short range transport technologies available in nowadays’ mobile devices (e.g. Bluetooth, Wi-Fi, etc). In this new content distribution scheme, the more popular the content gets the more chances it could be retrieved from a cached copy within the crowd and, this could lead to a less amount of data transferred over the 3G uplink.

Important concepts and details about technologies used by this master thesis will be introduced below.

1.3.1 Large Crowd Events

The term “large crowd events” referred many times by this document can be defined as a large group of people that gathers together for different purposes such as football matches, concerts, political events, demonstrations, accidents, etc.

The large crowd events can be expected or unexpected (flash crowd events). Nowadays, the increase of network traffic in expected events (e.g. football matches in stadiums) can be handled by solutions such as picocells [13], which can add network capacity in these scenarios with very dense traffic volume. Even though there is a current solution for expected events, there is space for alternatives solutions.

During unexpected events, the other type of large crowd events, there is not the possibility to pre-allocate resources to handle this surprising boost or increase of network traffic. Therefore, for these types of scenarios, there is a need to develop other alternatives

\(^2\) CommProve enables mobile network operators to analyze financial performance, quality of experience and network performance in real-time (http://www.commprove.com/)
strategies that can prevent and avoid network congestions \[9\].

Figure 1 shows a simple scenario where two cell phones try to retrieve content from a server that is somewhere in the cloud. If the up-link is congested due to saturation generated for an expected or unexpected boost of network traffic, those two users will have less chance to be able to get the content desired. One alternative solution for this type of scenarios could be to also allow users to exchange content using P2P-like communication through short range technologies (Bluetooth, Wi-Fi, etc.) available nowadays in mobile devices.

**Figure 1. Event with large crowd scenario.**

1.3.2 **Peer-to-Peer**

In Peer-to-peer networks, clients (PC, mobile phones, or tablets) can exchange content such as data, audio or video between then instead of getting it from a central server. The prototype scenario developed by this master thesis uses a peer-to-peer like communication to allow users to exchange content. Using this type of networks, popular content can be retrieved from any nearby device that has the content using short range technologies instead of retrieving it from a central server. This strategy reduces the amount of data transmitted over the uplink. The more popular the content gets, the higher the chance that the content can be found in nearby peers. Thus, in large crowd event scenarios where most of the people share a common interest, P2P-like communication seems to be a suitable alternative solution for content retrieving purposes.

1.3.3 **Host-centric network paradigm**

The architecture used in in today’s internet is based on the host-centric communication paradigm, where data is tightly coupled with location (IP addresses). In this type of architectures, resources in the network are typically identified with human readable locator (e.g. URL). When users want to access the resources, the request is first forwarded to a Domain Name Server (DNS) that translates this URL into a valid IP address. If this process succeeds, the request is forwarded to the content owner server located in the specified address, which will
send back the desired resource to the user. As an example, in webpages, content is bound to specific IP addresses/Location, see Figure 2.

![Figure 2. Host-Centric Networks](image)

There are a number of issues related with the amount of resources that have to be allocated in the network to satisfy the current users’ needs; resources in terms of bandwidth capacity in a particular link or the capacity to route a packet of data with a sufficiently low delay. Nowadays, millions of new devices have access to internet, users demand content with higher quality (more bandwidth usage) and more users utilize delay critical real time video and audio. These changes will only be handled in today’s internet (based on host-centric networks) with significant investments in network infrastructure (e.g. using CDN). More intelligent solutions like information-centric networks can be used as an alternative to the current internet architecture, using in a more efficient way the resources available in the network [12].

### 1.3.4 Information Centric Networks

New network architectures alternatives are being developed in order to improve the distribution of current content demand; these new architectures are focused on the information itself rather than its storage location. Some of the basic characteristics of these new architectures are [14]:

- **In-network storage**: caching of content at the transport layer allowing different network components (e.g. name resolution services) to have temporal copies of the content. This improves content distribution since future request to the same content can be retrieved from any cached copy in the network. A complete definition of the in-network caching storage system can be found in [15].

- **Persistence and unique naming**: in order to be able to cache copies of the content in the network, the different content copies must have the same name. Therefore, the naming scheme has to be persistent. Also, a unique naming scheme is necessary in order to identify the content in the network regardless of its location.

- **Content Mobility**: in this type of architecture, content is no longer tied to a specific location since it can be cached anywhere avoiding host reachability problems.

These types of networks are known as Information Centric Networks (ICN) and they...
Seem to be a potential alternative solution to handle the current increase of content distribution over internet.

In ICN, the content is not tied to any permanent location but it can be stored anywhere in the network (servers, routers, other users, etc.). When users request content, this content is said to be retrieved from the network i.e. from any cached copy available. In order to allow this type of in-network caching, the content is identified with a unique name within the network (e.g. hash of its content).

Figure 3 shows a simple example of an ICN content distribution, where the information stored in the web server can be retrieved from any content cache node in the network who also has the content, and not only from the web server. In this architecture, the more popular the content gets, the more spread it will be in the network allowing users to retrieve it from a closer copy reducing traffic in the core network and decreasing the latency perceived by the user.

Nowadays, there is an increase of the demand of CDN deployments due to the interest of caching certain contents closer to the end-users in order to provide better content distribution and better user experience. CDN is a solution that is based on contract between content and CDN providers and is mostly applied for applications based on Hyper Text Transport Protocol (HTTP-web based). ICN on the other hand, aims to enhance scalability of content distribution, as mentioned above, through content replication and in-network caching and also making it available for all applications and all types of content [16].

1.4 Motivation

Mobile networks have the challenge to optimize the bandwidth usage during crowd events such as football matches, demonstrations, festivals, accidents, etc. Most of the time, users assisting to these events share similar interests and download the same content thousands of times, unnecessarily saturating the network capacity. Innovative practices like ICN can be applied in these scenarios; the contents requested to the cloud can be cached in users’ devices (local caching) and exchanged using P2P-like communication and short range transport technologies, such as Bluetooth or WIFI Direct. This set-up could be useful in cases of up-link (e.g. 3G) congestion, or when the alternative transport technologies offer benefits for the users.
in terms of higher performance or lower costs.

1.5 Hypothesis
NetInf enabled mobile devices can provide a new control layer that allows the use of alternatives short range transport technologies (e.g. Bluetooth, WiFi Direct) for P2P-like communication using local caches in users’ devices. With NetInf ported in mobile devices, the content distribution in “Large crowd events” can be done in an alternative way reducing up-link bandwidth usage.

1.6 Goal
Develop an alternative, scalable and more efficient solution, based on NetInf ICN approach, capable of reducing the mobile up-link bandwidth for content distribution in large crowd events.

1.7 Objectives
- Reduce mobile uplink bandwidth consumption for content distribution in large crowd events.
- Study of current short range transport technologies available in mobile devices, which could be used for content distribution in large crowd scenarios.

1.8 Task
- Port an existing NetInf ICN approach implementation to mobile devices.
- Provide a fully working content distribution application prototype based on short transport technologies (such as Bluetooth and WIFI Direct) using ICN NetInf approach as a new control layer for mobile devices.
- Define and perform different test cases to evaluate the prototype performance.

1.9 Approach
The project was divided in four main phases: literature research, implementation, testing and documentation.

During the literature research phase, we started by reviewing several ICN related scientific papers to get the necessary background, needed to understand this type of architectures. Afterwards, we evaluated the different ICN Netinf implementations to understand the challenges of porting it to mobile devices. We also studied the capabilities, in terms of transport technologies, available in the different mobile devices, in order to determine how feasible the implementation of the ICN NetInf prototype was over short range transport technologies (e.g. Bluetooth and Wi-Fi Direct).

During the implementation phase, we made the necessary code changes to port the new control layer to mobile devices. Also, some other features/components were added to show compatibility with other NetInf components developed by other Sail partners and at the same time to take advantages of their benefits.

During the testing phase, we defined a number of test cases to show the function ability of the prototype developed over the short transport technologies chosen.
Finally, throughout the documentation phase, we compiled all useful information that was used during the prototype development. This phase started from the very beginning of the project in order to facilitate the documentation of all the process.

1.10 Opportunities

This master thesis had the opportunity to test one of the five application scenarios defined for the NetInf prototyping work in the “(D-B.1) The Network of Information: Architecture and Applications” document [16]. The final prototype represents a valuable input for the research engineers working within SAIL. The result of this thesis shows the possibility to provide an alternative solution for content download problems in these scenarios with limited up-link bandwidth, combining the capability of NetInf to perform local caching and local object resolve, together with the advantages offered by Android devices, such as the innovative WiFi Direct transport technology.

1.11 Target Audience

The outcome of this project contributes to the research area done within SAIL project [7]. “Events with large crowds” is one prototype scenario defined within the WPB work group in SAIL project to study the potential gains of using NetInf ICN approach.
2. Background and Related Work

In this section, different ICN approaches will be introduced (e.g. DONA, CCN and PSIRP); these approaches resulted in valuable contributions for the current ICN NetInf approach. Afterwards, it will be introduced the NetInf ICN approach that was started with the 4WARD NetInf project and that is being extended by the current SAIL NetInf project. Finally, we introduced the different technologies used to develop the prototype.

2.1 Previous ICN Approaches

2.1.1 Data Oriented Network Architecture (DONA)

The Data Oriented Network Architecture (DONA) project suggests that one of the main problems of the host-to-host based architecture used in today’s internet resides in how names are structured and resolved. The three main issues addressed by DONA are: persistence, authenticity and availability of information. To solve such problems, a new naming scheme together with a different name resolution technique is proposed.

The naming scheme is responsible for handling the persistence and the authenticity and it is based on location independent, flat and self-certifying names. This naming scheme is based on owners. Each owner has a set of public-private key, which is used for authenticity and identification. Objects’ names are of the form P:L where P is the cryptographic hash of the owner’s public key and L is a unique label chosen by the owner; this label L is used to uniquely identify each object. Since in this naming scheme object names are no longer tied to its location, the persistence problem is mitigated. Moreover, when a user request an P:L object, the reply will contain a triplet holding the data, the public key of the owner and the signature of the owner. Using this information the user is capable of authenticate the owner and therefore the authenticity problem is solved.

In order to tackle the availability problem, DONA uses a name resolution technique based in the route-by-name paradigm. Instead of using the well-known domain name system (DNS), DONA uses name-based anycast primitives that live above the IP layer. The two main primitives used are: FIND and REGISTER. The REGISTER primitive is used whenever an owner wants to inform the network of the existence of a certain object. On the other hand, the FIND primitive allows users to locate the nearest copy of the object P:L, which has to be registered first. In order to register objects in the network and route the peers’ FIND requests, a new entity called resolution handler (RH) is introduced. The RH uses the information contained in the REGISTER messages to route future FIND requests to that object. [5] [16] [17]

2.1.2 Content Centric Networking (CCN)

The Content Centric Networking (CNN) is an ICN approach proposed by Van Jacobson et. al. in the Palo Alto Research Center (PARC) [18] [19]. According to this project, the current use of the Internet is focused more in what the information is and less in where it come from, therefore a change of paradigm in how the content is requested and delivered is needed. CCN’s authors identified three main issues with today’s internet host-to-host approach: availability of popular content is difficult and costly to obtain, security is based on the channel and sometimes includes relying in untrustworthy location and finally object names are mapped to locations,
which can lead to complicated network configurations when content is moved from its original location. Therefore, CCN propose an architecture where data/information is named instead of naming the hosts.

CCN uses two types of packets in order to allow a peer to retrieve content: interest and data. The interest packet is broadcasted by a client through all its available interfaces to ask the network for a certain type of content. These requests are check at all the receiving peers to determine if the content is available in their content store. If the content is not found, the interest message is forwarded upstream toward content source(s). However, if the content is found it is retrieved to the interested client using the other type of packet: the data packet. The data packet is routed to the requesting client using the same path that the interest packet took; the content is cached on the different peer on its way towards the source to be able to satisfy future request to the same content. Since the content being exchanged is named, forwarding peers are aware of it and can satisfy more than one peer requesting the same content using basic multicast techniques.

The naming scheme used in CCN is hierarchical to allow scalability through name prefix aggregation. CCN also implements several of the main basic mechanisms used in current IP networks including: flow and congestion control, intra-domain and inter-domain routing, etc. Security is based in content instead of in the channel avoiding some of the current vulnerabilities of IP. [19] [16] [17]

2.1.3 Publish/Subscribe Internet Routing Paradigm (PSIRP)

The Publish/Subscribe Internet Routing Paradigm (PSIRP) is an ICN approach that proposes a redesign of the Internet architecture from the host-to-host to a more information centric approach. This project suggests using the Publish/Subscribe paradigm, where content is first publish to the network by its owner (called the “publisher”) and peers that want to obtain such content(called “subscribers”) subscribe to it to be able to get it. This architecture is said to be receiver oriented since subscribers are responsible for initiating the communication; this contrast with today’s architecture, which is sender oriented.

A PSIRP network brings several benefits like preventing most of SPAMs and DoS attacks due to its receiver oriented architecture. Moreover, since publications are immutable and persistent, another benefit from using this paradigm is that multicast traffic can be delivered in more efficient and scalable way, since objects are uniquely identified and can be therefore cached in the network avoiding typical bottleneck problems.

One of the main differences between PSIRP and other ICN proposals is that it proposes to build a network from scratch, i.e. not based IP networks. According to this proposal, IP networks are designed for message exchanging and, even though it is possible to build another layer on top of IP (e.g. peer to peer network), this reduces the flexibility and efficiency of the architecture.

The PSIRP architecture has four main functions: rendezvous, topology, routing and forwarding. The responsibility of each function can be summarized as follow [20]:

- **Rendezvous**: is responsible for creating a link between publishers and subscribers. It matches subscriber’s requests to data published by data sources.
- **Topology**: is responsible for maintaining the information about the physical network
topology.

- **Routing:** is responsible for create and maintain the delivery trees for the different publications. This function is also responsible for do the network caching at the intermediate nodes.

- **Forwarding:** is responsible for delivering the information from data sources to the subscribers using the delivery trees.

Whenever a publisher creates a publication, an identifier will be created by the Rendezvous to be able to identify it; this publication identifier is called a Rendezvous ID (RID). The publications are organized in networks called scopes. Every scope has a Scope Identifier (SID) [20] [16] [17] [21].

### 2.2 Networking of Information (NetInf)

Networking of Information (NetInf) is an ICN approach that was first proposed by the 4WARD project [22] as an information-centric paradigm that was intended to address issues that the current Internet host-centric networks were not designed to deal with [see, Host-centric network paradigm 1.3.3]. NetInf is built upon location independent objects called Information Objects (IO), which are uniquely named allowing in-network caching. The communication process is centred in requesting and retrieving IOs from the network without the need to establish a direct communication with the source of the information itself, or with its possible redundancy or replication sources, as it is in today’s Internet. The authenticity and privacy are provided by information contained in the IO itself since the desired object will not necessarily be fetched from the original source (e.g. it can be obtained from a cached copy). Therefore, current security techniques used in host-to-host network, which are focus in securing the communication channel instead of the information, are no longer applicable.

NetInf was proposed and first defined by the 4WARD project. Afterwards, the SAIL project [7] continued with its development and added some changes based in results of the others ICN projects mentioned beforehand (DONA, CCN, etc.). According to [16], SAIL NetInf is based in the following three main foundations: “the idea of unique naming of information objects without imposing a hierarchical naming structure (similar to the approaches developed by 4WARD and DONA); receiver-oriented transport as in CCN; a multi-technology/multi-domain approach than can leverage different underlying network services and employ different name resolution/name-based routing and transport mechanisms.”

Since SAIL NetInf was designed to be a medium term solution capable of run on top of the existing network, the results from the 4WARD project in naming and name resolution were not only leveraged but also adapted for such purpose. NetInf intends also to allow interconnection of different domain, each one of them with their own: name resolution strategies, routing strategies and underlying network technologies. Such a heterogeneous environment can ease the integration with current host-centric networks and allow a future migration to information-centric networks.

#### 2.2.1 Architecture overview

One of the main advantages of SAIL NetInf is the possibility to have different NetInf domain using different routing/name resolution approaches and different underlying technologies as shown in Figure 4.
The actions a NetInf peer will perform are most likely referred to object exchange within the network. In NetInf domain such objects are called Information Objects. The two main functions a peer will use are:

- **Publish (Information Object):** function used to publish Information Objects to the network.
- **Request (Information Object):** function used to request an Information Object from the network.

Information objects must be published first before they can be requested. However, these two functions are decoupled in time, which means that there is not any specific relationship between the time when the IO is published and the time when it is requested. Moreover, since the IO is published and requested to the network and not to one specific peer, it is likely that the peer replying to the request message is different than the peer that published it. Any peer who has cached this IO previously or any Name Resolution System with caching capabilities could answer a request. Figure 5 shows the interaction between the requester peer and the publisher peer.

As shown in Figure 5, peer #2 (Publisher) publish IO1 to the network. Afterwards, peer
#1 (Requester) make a request for IO1 which is answered by peer #3. As can be seen, the request for IO1, which was published by peer #2, was answered by another peer in the network (peer #3) who did not publish the content but just happened to have a copy of it.

### 2.2.2 SAIL NetInf Naming

SAIL NetInf names [16] are location independent identifiers for the IOs in the network. Beside its identifier function, NetInf name are also used for IO resolution and routing. NetInf names may carry some hierarchical information but this is not a requirement, therefore this naming scheme is considered to be not hierarchically structured. The naming scheme used is URI-based since it is a very well-known scheme in today’s Internet and this can ease the migration process from current URLs (e.g. http://, ftp://, etc) to networking of information names (e.g. ni://). This URI-based names will be used for publish and request IO operations.

The syntax of the SAIL NetInf name contains the following elements [23]:

- **Scheme name (Mandatory):** the scheme name used in in SAIL NetInf is “ni”.
- **Colon and slashes (Mandatory):** literal “://”.
- **Authority (Optional):** optional attribute that can help applications locating the target IO. According to the specification, two NetInf name with the same hash value, one with authority and the other one without authority will almost always point to the same object.
- **One slash (Mandatory):** literal “/”.
- **Digest Algorithm (Mandatory):** the digest algorithm used to create the hash contained in the name. For this master thesis the algorithm used will be sha-256
- **Separator (Mandatory):** literal “:”.
- **Digest Value (Mandatory):** digest value contained in this name. For this master thesis, this value will be the hash of the file content encoded in Base64URL.
- **Query Parameter separator (Optional):** literal “?”. It is the separator symbol used between the digest value and the query parameters if specified.
- **Query Parameters (Optional):** optional parameters included in the object’s name in the tag=value form.

Figure 6 shows an example of a SAIL NetInf name:

\[
\text{ni://example.com/sha256;B_K97zTtF...y6fk}
\]

**Figure 6. SAIL NetInf name example**

### 2.3 Comparison between the different ICN approaches

Table 1 summarizes the main differences between the ICN approaches explained beforehand [14]:
<table>
<thead>
<tr>
<th>Aspect</th>
<th>NetInf</th>
<th>PSIRP</th>
<th>CCN</th>
<th>DONA</th>
</tr>
</thead>
</table>
| **Naming scheme**                  | Flat namespace are used; there is not hierarchical structure that can be used for example for routing purposes | Two flat namespaces are used: rendezvous identifiers and forwarding identifiers.  
Rendezvous identifiers are the names for Redezvous points.  
Forwarding identifiers are used to transport data once the contact has been established | A hierarchical naming scheme is used.  
This scheme allows routing scalability through name prefix aggregation | Flat naming scheme based in PRINCIPAL:LABEL form (P:L). The PRINCIPAL has a global uniqueness and the LABEL is unique only within the PRINCIPAL namespace |
| **Name resolution service**        | In NetInf two resolution mechanisms have been developed: two-step approach and one-step approach.  
In the two-step approach, a resolution services is queried and one or more locators are returned for the desired object. Afterwards, these locators are used to retrieve the object.  
In the one step approach or Integrated resolve/retrieve, objects are directly return without first returning locators | PSIRP has a two-step resolve / retrieve model, where the resolver is called rendezvous point. | CCN has an integrated resolve/retrieve mechanism. Clients ask for an information object by sending interest packets, which are routed toward the publisher of the name prefix using longest prefix matching. | In DONA, nodes that are authorized to serve data, register to the resolution infrastructure. Only once a given content is registered, requests can be routed to it. |
| **In-network storage for caching** | Two types of cached copies are used : cached-copies found in a Name Resolution Server or copies found in nodes, with a cache-aware NetInf transport protocol, on the path to a node known to hold a copy.  
The INTEREST requests are forwarded towards the publisher and the caches of the intermediate nodes are checked for copies of the requested IO. Therefore, the request will be answered by any intermediate node that has the desired IO or in the worst case from the IO source.  
The IO is cached on every node on the return path to the requester | Caching is limited to the scope of the rendezvous point for the identifier associated with an object | Caching is limited to the scope of the rendezvous point for the identifier associated with an object | Any cache can respond to a FIND request and serve the relative IO. |

Table 1. Main differences between the ICN approaches
2.4 Technologies and Programming Tools

2.4.1 Android OS

Android OS was the platform chosen to develop the scenario prototype. The main reasons that led the decision to select the Android OS as our mobile platform were:

- Android OS is an open platform that counts with an extensive and useful documentation. Also, there are open forums where developers all over the world collaborate with new tools to facilitate the development of new applications.

- In the last years Android OS has become the most popular mobile platform. According to [24] 550.000 new mobile devices are being activated each day worldwide. Currently, there are more than 200 million of Android devices activated. In the third quarter of 2011 there were around 2.4 billion of app downloads to Android devices [24]. This facts, give us a great market opportunity to promote the results of our master thesis.

- The Android mobile devices that were chosen for the demo have available the two short transport technologies (Bluetooth and WIFI Direct) used in the prototype scenario. In Android developers website there are available the Bluetooth and WIFI direct APIs that facilitate the development of our prototype. There are also available great examples on how to use these APIs.

- We had previous experience developing in Android OS platform.

Android Platform 4.0

Android platform 4.0 or Ice Cream Sandwich was the version chosen to develop the prototype. The main reasons that led the decision to select the Ice Cream Sandwich as our platform were:

- Ice Cream Sandwich is the latest version of the Android platform for phones, tablets, and more [25].

- Introduce the WIFI direct APIs that is one of the short transport technologies used in the prototype.

2.4.2 Bluetooth

Bluetooth was the first short range transport wireless technology approach taken by this master thesis. The main reasons that led the decision to select Bluetooth as our first transport technology were:

- Our prototype is developed in Android OS, this platform has available Bluetooth APIs that make easier the development of applications that integrate this wireless technology.

- The Bluetooth APIs in Android OS are more mature and the behaviour is more stable than other short range technologies, such as WIFI Direct (The second approach taken).

- Bluetooth is available in most of the Android devices offered in the market at the present time.

Main Features

This technology allows wireless communication between Bluetooth enabled devices. The main
features are robustness, low power, and low cost. Nowadays, there are billions of devices (such as mobile phones, tablets, computers, etc.) using Bluetooth connections to communicate wirelessly and providing to the users solutions such as media content exchange, hands-free headsets for voice calls, printing and fax capabilities, etc.

Any Bluetooth enabled device can connect to other Bluetooth enable device that is located in proximity to one other. Bluetooth devices communicate each other forming ad hoc networks known as piconets. This type of networks (piconet) is established dynamically and automatically as Bluetooth devices enter or leave the radio proximity [26]. Each device can simultaneously communicate with up to seven other devices within that single piconet.

The core specification of Bluetooth establishes a minimum range of 10 meter but the manufactures could tune their implementation to provide the range needed for a specific solution. There are different Bluetooth versions that have been developed: Bluetooth v1.0 and v1.0B, Bluetooth v1.1, Bluetooth v1.2, Bluetooth v2.0 + EDR, Bluetooth v2.1 + EDR, Bluetooth v3.0 + HS and Bluetooth 4.0. The latest version Bluetooth 4.0 has been adopted in June 30, 2010 [27] and includes Bluetooth high speed (based on WIFI) and Bluetooth low energy protocols. All the Bluetooth versions are backward compatible with the versions launched before.

The devices facilitated by Ericsson for the development of the scenario prototype are:

- 2 Sony Ericsson Xperia Arc that come with the version Bluetooth 2.1 + EDR (Enhanced Data Rate) [28].
- 2 Samsung Galaxy Nexus that come with the version Bluetooth 3.0 (without +HS) [29].

As we can see from Table 2 the data transfer rate difference between the version Bluetooth 2.1 + EDR and Bluetooth 3.0 + HS is considerable high. The later version can provide much faster data transfer since the high data rate traffic is carried over 802.11 radio link. The Bluetooth link is used for negotiation and establishment [30].

Many Bluetooth enabled devices contain the 802.11 radio to have access to internet. Most of the smart phones available in the market nowadays contain the 802.11 making possible the integration of the version 3.0 + HS, giving the users the opportunity to take advantages of the benefits that this technology bring with it (Significant increase of data transfers rate). For this master thesis project just devices with Bluetooth v2.1 + EDR and Bluetooth v3.0 will be used. The two Samsung Galaxies used for the prototype scenarios come with Bluetooth v3.0 but not with the major feature added + HS. It means that they do not have the capability to carry data over 802.11 radio link.
Android Bluetooth Manager

The application framework makes possible wireless data exchange between Bluetooth enabled devices. The Bluetooth APIs let developers implement the following functionalities [31]:

- Scan for other Bluetooth devices
- Query the local Bluetooth adapter for paired Bluetooth devices
- Establish RFCOMM channels
- Connect to other devices through service discovery
- Transfer data to and from other devices
- Manage multiple connections

2.4.3 Wi-Fi Direct

Wi-Fi Direct was the second short range transport wireless technology approach taken by this master thesis. The main reasons that led the decision to select Wi-Fi Direct as our second transport technology were:

- Our prototype is developed in Android OS, this platform has available Wi-Fi Direct APIs that make easier the development of applications that integrate this wireless technology.

- We wanted to integrate a second short range transport technology alternative. Wi-Fi Direct is a new feature that came with the Android 4.0 Ice Cream Sandwich platform. This wireless technology was suitable for the prototype scenario designed within this project. It is still an un-mature technology (unstable behaviour) but seems to be a potential wireless technology alternative for content distribution between peers.

Main Features

Typical Wi-Fi networks required the use of wireless access points (AP) to perform functionalities such as physical support for wireless and wired networking, bridging and routing between devices on the network, and service provisioning to add and remove devices from the network. Access point in the “infrastructure mode” works as hub to which the wireless devices are connected.

The use of Wi-Fi in today’s mobile devices has increased in a significant way making even more important the creation of ad-hoc networks that give the users the opportunity to do things like share content, connect to a local printer or even play games wirelessly without the need of an AP.

Wi-Fi Certified Wi-Fi Direct is a certification program from the Wi-Fi Alliance that defines a new way for Wi-Fi devices to connect each other. Some of the main advantages of Wi-Fi Direct are the possibility to print, synchronize or share content among Wi-Fi devices without the need of an AP. All Wi-Fi Direct devices are able to operate as either a device or an access point. The Wi-Fi Direct devices negotiate when they first connect to determine which device acts as an access point. The Wi-Fi Direct connections work within the typical Wi-Fi speeds and range. Wi-Fi Direct devices can work on 802.11 a, g and n Wi-Fi standards. Wi-Fi Direct devices can operate in the 2.4 GHz 5 GHz frequency bands. The connections are protected by WPA-
certified security protocols. The ranges that provide this technology are the same as the any Wi-Fi certify device with ranges up to 200 meters [32].

The Wi-Fi Direct-certified device that acts as the AP manages the creation, admission to, presence and termination of that network implementing the specification underlying the Wi-Fi Direct program. Just as typical Wi-Fi connections Wi-Fi Direct creates IP-based networks between enabled devices, allowing service discovery methods to work just as they do over a wireless LAN today. The specification defines a new pre-association discovery method, giving Wi-Fi Direct-certified devices the ability to discover devices and limited information about device services prior to association (and before having an IP address) [32].

**Android WIFI Direct Manager**

The new Android platform 4.0 comes with the possibility to integrate Wi-Fi Direct enabled devices functionalities to any innovative application that demand the wireless connectivity between devices without an intermediate access point. The Wi-Fi Direct APIs offer the option to take advantages of the faster transfer rate presented in Wi-Fi connection across much longer distances than a Bluetooth connection. For mobile devices, this technology is useful for applications that share data among users [33].

The main methods that are available in the Wi-Fi Direct APIs are [33]:

- Methods that allow you to discover, request, and connect to peers are defined in the WifiP2pManager class.
- Listeners that allow you to be notified of the success or failure of WifiP2pManager method calls. When calling WifiP2pManager methods, each method can receive a specific listener passed in as a parameter.
- Intents that notify you of specific events detected by the Wi-Fi Direct framework, such as a dropped connection or a newly discovered peer.

**2.4.4 Java**

Java was the programming language chosen to develop the scenario prototype in the Android devices. The main reasons that led the decision to select the JAVA platform were:

- Android OS is based on JAVA. There is the possibility to use a NDK toolset [34] that makes possible the development in Android OS using native code such as C or C++. Nevertheless, the Android documentation encourages the developers to use JAVA since most of the examples applications and supported APIs are written in this platform.
- OpenNetInf implementation and applications were written in JAVA that makes suitable the usage of the same platform.
- We had more experience developing in JAVA platform.

**2.4.5 Eclipse**

Eclipse is an open-source that develops open platforms and products. We chose Eclipse as the Java development environment to develop the scenario prototype in the Android devices. The main reasons that led the decision to select the Eclipse as the developing platform were:
- Multiple advantages when developing with Android OS. There is an Android development tool (ADT) that is a plugin for the Eclipse IDE that extend the capabilities of this platform to let developers quickly set up new Android projects, create an application UI, debug the applications, etc. [35]

- We had extensive experience developing in Eclipse environment.

2.4.6 Guice

Guice is used by the OpenNetInf implementation to increase the flexibility and modularity of the project. The reason for continuing using Guice for our Android Prototype was that this tool allows an easy configuration of what services a NetInf node will offer (e.g. Resolution service, Transfer Service, etc). We wanted to continue with the code modularity that OpenNetInf used.

Guice is a dependency injection framework for Java, Python and C++ developed by Google [36]. Figure 7 shows a small example of the usage of Guice:

![Figure 7. Google Guice example](image)

As shown in Figure 7, for a certain interface, there can be several implementations. Using the Guice model, the developer is able to configure which implementation will be used in the program just by configure the Guice Module. In the example, Interface X is bound to Implementation 1 and therefore when this interface is called in the program, the implementation that will be called will be Implementation 1.

For a detailed documentation about Google Guice please refer to [37].

2.4.7 Restlet API

Restlet is an open-source REST framework for the JAVA platform that is suitable for the development of client and server web applications. This API was chosen since made easier the development of our HTTP client and server application. [38]

The implementation of HTTP communication was necessary for the communication between the application and NetInf nodes, and for the communication between the nodes and external servers (NRS, QRcode server, Nicon tent server) that are part of the architecture proposed (further explain in section 4).
2.4.8 OpenNetinf

In order to accomplish with the objectives defined, there were many decisions that had to be taken. At the beginning of the project we had the need to decide which NetInf ICN implementation to use. We had three options that were discussed between us and the supervisors:

- Start from scratch
- Take an existing NetInf ICN implementation called OpenNetinf done by the computer network group at the University of Paderborn, Germany.
- Take an existing NetInf ICN implementation from the NEC group.

After two weeks reviewing the NetInf literature, checking the different code implementation and documentation the decision was to base the prototype carry on by this mater thesis on the OpenNetinf implementation developed at the University of Paderborn-Germany. The main reasons that led us to take this decision were:

- OpenNetinf is open source.
- OpenNetinf comes with good code documentation.
- OpenNetinf is a Java implementation that is the core programing language used to develop in Android devices (OS chose to develop the prototype scenario).
- OpenNetinf has useful prototype scenarios showing the advantages of using NetInf ICN approach.
- OpenNetinf has the basic functionalities that this thesis needed to develop the prototype scenario (such as NetInf node, IO resolution controller, Transfer Dispatcher, etc.)

The other two solutions were not suitable since first the NetInf ICN approach developed by NEC group was based on C (programing language), making more challenging to port it to Android devices. Moreover, to have access to that code some papers had to be signed and that would have taken more time at the beginning of the project. Second, start from scratch was not an appropriate alternative since it would have required more developing time for some components that were already available in the OpenNetinf implementation.
3. OpenNetInf Platform

In this section will be described the main OpenNetInf components that have relevance for the prototype implementation done by this master thesis. The description start describing the main components: node, event service and application and tools. After, more details will be given regarding the node structure, data model, node interface, application interface and IO resolution.

3.1 OpenNetInf Components

In the OpenNetinf implementation there are three main separate and independent components that make possible the communication through information-centric network architecture, see Figure 8:

![Figure 8. Components within NetInf Architecture [39].](image)

3.1.1 Node

The nodes are the first and most important components of the architecture. Nodes can provide three key services:

- **Resolution Service (RS):** provides the capability to store and resolve IO.
- **Search Service (SS):** search within the space of IOs.
- **Transfer Dispatcher (TD):** in charge of transferring the bit-level data (BO) when an IO has been found within the NetInf space.

The communication between nodes can be done via either HTTP or TCP.

3.1.2 Event Service

Event service (ES) is the second component that we can find in the OpenNetinf architecture. ES
can be seemed as publish/subscribe systems that are in charge to inform about the IOs that have been changed, created and deleted. Nodes can subscribe to the ES for specifics IOs changes. The changes are published by other nodes which store these IOs and notice an update to an IO. After changes have been published the ES is in charge of the notification distribution.

3.1.3 Application and tools

These components are situated on top of the node infrastructure. Within OpenNetInf there are a number of applications and tools that were developed to demonstrate the benefits of NetInf ICN approach. These applications, such as IOManagementTool or ShoppingTool, can be found in the project website [40] [39]. For the prototype scenario introduced in this paper was developed an Android OpenNetinf application that communicate via HTTP with the nodes.

3.2 Node Structure

Figure 9 contains the most important functionalities of a NetInf node.

A node can be accessed via NetInf Server, through one of its subclasses the TCP Server or HTTP Server. Accordingly, the communication between nodes can be done via HTTP or TCP. The current OpenNetinf implementation support two message representation: Google protocol buffer messages and XML. These messages are translated into internal messages called NetInf messages. The NetInf server delegates these messages to the NetInf nodes. Each node can provide three different kinds of services:

- Storing and retrieving of IOs, called “Resolution Service”.
- Searching within the space of IOs, called “Search Service”.
- Transferring the bit level Data (BO), called “Transfer Service”.

Figure 9. Node Structure [39]
Each node uses three different controllers to provide and manage the different services. Each controller can have an arbitrary amount of service that can achieve certain task in different ways.

- **Resolution Controller**: provides the capability to store, retrieve, update and delete IOs. Some of the resolution services that the resolution can manage are: local resolution service (to store or retrieve e.g. IOs in the local Data base) or external resolution service through another NetInf node (to store or retrieve e.g. IOs in the local Data base of the other node).

- **Search Controller**: is in charge of providing the search capabilities.

- **Transfer Controller**: manage the transfer service in order to transfer binary files (BO). Different stream providers can be used by the different services to transfer the files (currently supported by OpenNetinf FTP, HTTP).

### 3.3 Data Model

OpenNetInf data model defined in [39] is based in Information Objects (IO). Such IOs contain a set of attribute which can be mandatory or not depending on the type of IO. There are three types of IOs in this data model:

- **IdOs**: the Identifier Objects (IdOs) are used for identification of objects and components in the NetInf network (e.g. resolution services, transfer services, etc.).

- **DO**: the Data Objects (DOs) contain pointers to the real data represented by an IO (e.g. URLs, IP addresses, etc.).

- **IO**: general type of Information Object that can contain any type of information.

Beside these three types of IOs, the objects containing the real raw data are called Bit-level Objects (BOs)

Each IO, regardless of its type, will have a unique identifier that will be compound by identifier labels, see Figure 10.

```
HASH_OF_PK=8c4e559d...9760f4aad371470ccf9~HASH_OF_PK_IDENT=SHA1~VERSION_KIND=UNVERSIONED~UNIQUE_LABEL=MyInformationObject1
```

**Figure 10. OpenNetInf Identifier example**

The possible labels an identifier can have are [39]:

- **HASH_OF_PK** (mandatory for all IOs): contains the hash of the public key of creator of the IO

- **HASH_OF_PK_IDENT** (mandatory for all IOs): contains the hash algorithm used to hash the public key of the IO’s creator

- **VERSION_KIND** (mandatory for all IOs): contains whether or not the IO is versioned.

- **UNIQUE_LABEL**: plain text string that identifies an IO.

Figure 11 shows the relationship between the different types of IOs and the content owner:
Figure 11 shows the owner of a picture called “MyPicture.jpeg” which has a public key and a location (in this example associated with an IP address but could be any other type of locator). Using the public key information, the owner constructs an IdO that will identify him in the NetInf network. Now, using this IdO the owner can create a DO for the “MyPicture.jpeg” file, which is the raw data or BO. The DO will contain: the information contained in the IdO to uniquely identify the owner of the DO in the network, a UNIQUE_LABEL value used to differentiate between different objects published by the same owner and finally a locator attribute that contains the address where the object represented by this DO, in this case “MyPicture.jpeg”, can be found.

3.4 Node Interface

OpenNetInf uses TCP/IP transport protocol for communication between the different components in the network. The messages exchanged between such components are encoded using Google Protocol Buffers [41] or XML [42].

The communication process used in the OpenNetInf project is shown in Figure 12:
Figure 12 shows the node’s interface used for communication between different OpenNetInf components. The component shown in the Figure 12 (e.g. an Android NetInf enabled device), has an open TCP socket waiting for NetInf messages to be received. Whenever a message is received, it will be forwarded from the TCP server to the Communicator instance, which is responsible for asynchronous and synchronous message handling. Since the received message is encoded by the source before sending it, the communicator will decode it, using Google Protocol Buffers or XML decoders, in order to obtain the actual NetInf Message. This NetInf message will then be passed to the NetInfNode receiver handler which will deliver this message to the NetInfNode instance itself. The node will then check if it has a registered service to answer the request: if a suitable service is found, a reply including the result will be sent back to the source, otherwise an error message will be sent back to the source.

3.5 Application Interface

The application interface was developed in order to simplify the access to a NetInf node, making it independent of NetInf messages sent between nodes; this application interface was introduced as a NetInf component. The main advantage of using a separate interface is that application developers will be able to use all the functionality offered by the control layer by interacting with it over a HTTP (a standard well known protocol) instead of needing to understand the different NetInf components and their specific functionality.

3.5.1 Restful HTTP

OpenNetinf implemented an application interface based on RESTful API following the architectural style called Representational State Transfer (REST). The HTTP methods used are listed in the HTTP/1.1 specification (see RFC 26163).

In OpenNetinf three resources can be accessible through HTTP interface: IO resource,
BO resource and Search resource. Below will be explained the two resources that are relevant for the implementation done by this master thesis [39]:

**IO Resources**

This is the first resource in the OpenNetinf implementation. The URI pattern that identifies IO resources is:

```
/io/ni:HASH_OF_PK={hashOfPK}~HASH_OF_PK_IDENT={hashOfPKIdent}~
VERSION_KIND={versionKind}~UNIQUE_LABEL={uniqueLabel}~
VERSION_NUMBER={versionNumber}
```

![Figure 13. IO resource URI Pattern](image)

The request that arrive with that path will be redirected according to the following URI pattern:

```
/io?hash_of_pk={hashOfPK}&hash_of_pk_ident={hashOfPKIdent}&version_kind={versionKind}&unique_label={uniqueLabel}&version_number={versionNumber}
```

![Figure 14. IO resource redirect URI pattern](image)

IO resource permits the use of GET, HEAD, POST, PUT and DELETE HTTP methods.

**BO Resource**

A BO refers the actual data (pictures, videos, music, etc). This resource is in charge of downloading and returning the actual data.

BO resources are identified by the following URI pattern:

```
/ni:HASH_OF_PK={hashOfPK}~HASH_OF_PK_IDENT={hashOfPKIdent}~VERSION_KIND={versionKind}~UNIQUE_LABEL={uniqueLabel}~VERSION_NUMBER={versionNumber}
```

![Figure 15. BO resource URI Pattern](image)

The request that arrive with that path will be redirected according to the following URI pattern:

```
/bo?hash_of_pk={hashOfPK}&hash_of_pk_ident={hashOfPKIdent}&version_kind={versionKind}&unique_label={uniqueLabel}&version_number={versionNumber}
```

![Figure 16. BO resource redirect URI pattern](image)

BO resource permits currently the use of GET and HEAD HTTP methods.
3.6 Resolution

The resolution process involves three main components: resolution controller, resolution selector and resolution services. Figure 17 shows the resolution process and the relationship between these three main components:

![Resolution Process Diagram]

When a resolution request NetInf-message is received by the communication part of the node, it will be forwarded to the Resolution Controller (RC) that is the component that manages all the available Resolution Services (RS). The RC will check and obtain all the RSs capable of replying to the request. If the RC finds one or more capable RSs the Resolution Service Selector (RSS) gets called; otherwise an error message is sent to the source. The RSS will determine, based on a priority value, the order in which the found RSs should be called. The RC will then try to resolve the request using the different RSs in the order determined by the RSS. If at least one of the RS manage to resolve the request, the answer will be sent back to the NetInfNode, which will send it back to the source. If none of the capable RSs managed to resolve the request, an error message will be sent to the source.

Resolution services will have three basic operations [39]:

- **GET**: receives an identifier as input and will return the correspondent IO if it is found.
- **PUT**: takes an IO as input parameter and stores it in the NetInf Network.
- **DELETE**: takes an IO as input parameter and deletes it from the NetInf Network.

Local resolution services will be normally given a higher priority than remote resolution services. In the same way, for different remote resolution services, transfer technologies with higher bandwidths or faster communication establishment times will have higher priorities among the remote resolution services.

The available resolution services a node will have can be easily configured using Google Guice modules (Section 2.4.6).
3.6.1 BO Handling

Bit-level Objects (BO) are handled in the NetInf networks using Data Objects (DO) [39]. These DOs are a specific type of IO introduced early in the 4WARD project that represents a specific BO. DOs contain locator attributes that point to the BO they represent (e.g. HTTP URL), which might be stored in another location different than the DO location.

Once the BO location is found, the BO itself is transferred using the Transfer Dispatcher (TD). The TD is an entity that manages different Stream Providers (SP) and each of them uses a specific protocol; for example there can be: an HTTP SP, a Bluetooth SP, etc. Data will be transferred as a data streams using a specific SP. In order to determine which SP should be used, the TD will use a locator selector that will pick, according to a priority value, which is the best file location. This priority value can be set according to technology preference (e.g. an HTTP transfer can be considered to be better than Bluetooth transfer), location/source preference or other parameters.

Figure 18 shows the BO handling process:

![BO handling process diagram](image)

Figure 18. BO handling process

Figure 18 shows the BO handling procedure for “MyPicture.jpeg” BO. From the locator attributes in the DO, it can be seen that the desired BO can be obtained from two sources: using the http URL provided in the locator attribute 1 or using the Bluetooth MAC address provided in the locator attribute 2. All this information will be handled to the TD that will use the locator selector entity to determine which locator to use. For this example, let’s assume that HTTP connections are preferred over Bluetooth connection because of transfer data rate, therefore the locator selector will indicate to the TD that locator 1 should be used. This will derive in the TD using the HTTP SP in order to fetch the BO. In case the BO cannot be retrieved using the locator 1, the TD will try to get the content using locator 2. If none of this locators work, an error message will be sent to the application.
3.7 Porting OpenNetInf to Mobile Devices (Android OS)

Once we took the decision of using OpenNetinf implementation as starting point for the prototype scenario, based on mobile devices, there were a number of changes that had to be made. These specific changes or additional functionalities added to the implementation are presented below and will be described in detail in section 4:

3.7.1 Data Model Implementation

3.7.1.1 RDF (Resource Description Framework)

Within OpenNetInf the implementation of information objects (IOs) was done in two ways: first using a basic implementation (called impl or Java implementation) and second using a resource description framework (RDF). For our prototype scenario, we took into consideration just the Java implementation of objects since it was a straight forward solution to integrate the basic components that were needed in the prototype. For further implementations the RDF framework (based on RDF library Jena1) can be integrated to the prototype using the available ANDROJENA3 API.

3.7.1.2 Information Objects Attributes

In OpenNetinf implementation the IOs have many attributes that were not considered because are not relevant or out of the scope for this first adaptation done for mobile devices. Some of the attribute that are not taking into consideration are: chunking and security attributes. The attributes used by the IOs used in our scenario prototype are locators (that can be of four different types, see Table 3) and priority. The priority is a random number that is assigned to each locator to avoid retrieving objects from the same peer (for more details see locator selector in 4.1.2)

Table 3. Attributes used in the IOs

<table>
<thead>
<tr>
<th>Locator</th>
<th>Tag</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth locator attribute</td>
<td>nimacbt://</td>
<td>Added by this Master thesis</td>
</tr>
<tr>
<td>WIFI MAC locator attribute</td>
<td>nimacwd://</td>
<td>Added by this Master thesis</td>
</tr>
<tr>
<td>WIFI IP locator attribute</td>
<td>niip://</td>
<td>Added by this Master thesis</td>
</tr>
<tr>
<td>HTTP locator (URL)</td>
<td>nihttp://</td>
<td>Previous used in OpenNetinf</td>
</tr>
</tbody>
</table>

3.7.1.3 Naming Scheme

The naming scheme was one of the main changes in the data model since the new Sail Netinf project defines a simpler way to identify objects. In order to accomplish this, we needed to change the parameters to create objects identifiers. The parameters used in OpenNetinf to create identifiers are:

3 http://code.google.com/p/androjena/
- Hash of the Public Key (hashOfPK)
- Hash algorithm (hashOfPKIdent)
- Version Kind (versionKind)
- Unique Label (uniqueLabel)
- Version Number (versionNumber)

The new parameters used to create the simplified identifiers are:
- Hash Algorithm (hashAlg)
- Hash Content (hashContent)

The implementation done by us, all the NetInf components utilize IOs with the simplified naming scheme.

The communication between the application layer and the nodes are done via HTTP. In section 2.2.2 was explained in detail the naming scheme that was defined by Sail Netinf. But, in order to make possible the deployment of ICN architecture with existing web infrastructures there is a need to map the NetInf messages to HTTP URL. If a request is done using e.g. this scheme (“HTTP GET ni://example.com/sha-256;abcd...”) it will not work. Hence Sail NetInf proposed a mapping scheme through an Internet-Draft called “The Network Information (ni) URI Scheme: Core Syntax” [23]:

For a ni name of the form: “ni://n-authority/alg;val?query-string”

The corresponding HTTP URL would be:


The HTTP requests sent from the application layer to the nodes in the prototype have the following form:


Example:

- http://localhost:8080/.well-known/ni/sha-256/GhMXD38aV6DQ03Sx8Cf9Cr8CWR8URmhe6HmMvl03hFw?METHOD=PUT&BTMAC=F0:E7:7E:3F:D2:43&WIMAC=a2:0b:ba:e0:94:54

3.7.2 Node

We did some changes in the internal OpenNetInf implementation node in order to integrate the components and functionalities that were needed for the prototype.

More details will be given in section [4.1.2.2].

3.7.2.1 Resolution Controllers

For our prototype two resolution services were developed, Local Resolution Service and Remote Name Resolution Service; both of them to retrieve and store IOs.

Local Resolution Service

For this resolution service was used the SQLite data base available in Android OS to store
structured data. The hash algorithm and hash value are the elements used to retrieve and store IOs, and these can be found in the IO identifier.

More details will be given in section [4.1.2.2].

**Name Resolution Service (NRS)**

For the prototype was integrated a name resolution server (NRS), working as a remote resolution service. This server was implemented by NSN (Nokia Siemens Network). With this service the nodes can communicate via HTTP to the NRS to retrieve the list of Bluetooth or WiFi Direct peers that have the content desired by the user.

The use of NRS shows interoperability with other NetInf components that have been developed by other partners within the Sail project. Nokia Siemens Networks was the partner that developed this component. They implemented a niproxy server and niclients for other scenarios prototypes. We adapted their niproxy server to act as a NRS (Register and get locators) and NCS (cache content in an external ni content server). The ni clients used to register, get and delete locators were adapted to the android NetInf nodes developed by us.

More details will be given in section [4.1.2.2].

### 3.7.2.2 Access Server

The android NetInf nodes have integrated three different servers (HTTP, Bluetooth and TCP) for diverse purposes.

**HTTP Server**

OpenNetinf implemented an HTTP server for the communication between the nodes and the application layer. Resolving IOs and transmitting BO between nodes is done through this way. The implementation of the HTTP server that we are using in the android nodes is based on this implementation and below is described the main characteristics:

- The HTTP server is used for the internal communication between the application and the node through a GET request, and between the application and the transfer dispatcher through a POST request.

- The server is redirecting requests to the IO resource in order to search, put or delete IOs. The server is redirecting requests to the BO resource in order to use the transfer dispatcher and retrieve files through Bluetooth, WIFI Direct or 3G using the ni content server (explained in further sections).

More details will be given in section [4.1.2.1].

**Bluetooth Server**

We added a Bluetooth server to the Android NetInf node in order to make possible the transfer of content using Bluetooth connections. This server is initialized when the node is started. The main function of this server is to wait for Bluetooth connections. When a connection is established the requesting node sends a protobuf message (short verification process) to verify if the remote node has the content desired. If the remote has the content, it will reply back with the content (ByteArrayStream) or with an error message.

More details will be provided in section [4.1.2.4, Bluetooth BO server].
**TCP Server (WIFI Direct)**

We added a TCP server to the Android NetInf node in order to make possible the transfer of content using WIFI connections. This server is initialized after the WIFI Direct discovery and connection process is finished. When the WIFI Direct connection is established the requesting node will send a TCP connection request and if it succeeds, the node sends a protobuf message (short verification process) to verify if the remote node has the content desired. If the remote has the content, it will reply back with the content (ByteArrayStream) or with an error message.

More details will be provided in section [4.1.2.3.4, TCP BO server].

**3.7.3 Transfer Dispatcher**

For the prototype two new providers were developed for the transfer dispatcher in order to retrieve content using Bluetooth and WIFI Direct. Also, was added the possibility to cache and retrieve content from an external content server through 3G. Below there is a brief explanation of these three elements.

More details will be provided in section [4.1.2.3].

**3.7.3.1 Bluetooth provider**

In order to be able to transfer files using Bluetooth we developed a new provider that is called by the transfer dispatcher. All the Android NetInf nodes have a Bluetooth server waiting for Bluetooth connections. The overall process to retrieve content from a neighbor is:

- First, the IO has to be resolved (i.e. find the locator with the content desired) using some of the resolution service available (e.g. NRS).
- A discovery process is done in order to find out if the peer that has the content is in the Bluetooth range.
- The peer will attempt to connect to the remote node using a Bluetooth connection. It will send a connection socket request.
- When the connection is established, the requesting node will send a Google protobuf message asking for the particular content and finally the remote node will either transmit the data or reply with an error message if the file was not found in the data storage.

More details will be given in section [4.1.2.3.3].

**3.7.3.2 TCP provider**

In order to be able to transfer files using WIFI Direct we developed a new provider that is called by the transfer dispatcher. All the Android nodes that have WIFI Direct capabilities will need to turn on the WIFI Direct in order to listen for WIFI Direct connections. The overall process to retrieve content from a neighbor is:

- First, the IO has to be resolved (i.e. find the locator with the content desired) using some of the resolution service available (e.g. NRS).
- A discovery process is done in order to find out if the peer that has the content is in the WIFI range.
- The peer will attempt to connect to the remote node using a WIFI Direct connection.
First the WIFI Direct has to be established. The node interested in the content will send a P2P invitation and the connection will be set-up.

- After the WIFI connection is set-up, there will be a group IP available for the communication. The remote node will start a TCP server using this IP address and the requesting node will send a connection TCP socket request.

- After the TCP socket is set-up, the requesting node will send a google protobuf message asking for the particular content and finally the remote node will either transmit the data or reply with an error message if the file was not found in the data storage.

More details will be provided in section [4.1.2.3.3].

3.7.4 Ni Content Server

We included a Ni content server as an additional functionality for the prototype. This server will act as an external content server that can be accessed from the mobile devices using 3G. The android NetInf nodes can either cache content in the server or retrieve from it.

This additional functionality aims to show interoperability with other NetInf components that have been developed by other partners within the Sail project. Nokia Siemens Networks was the partner that developed this component. They implemented a niproxy server and niproxy clients for other scenarios prototypes. We use their niproxy server to act as a Ni content server where we can cache and retrieve content using the 3G up-link. The ni clients used to publish and get content were adapted to the android NetInf nodes developed by this project.

More details will be given in section [4.1.5].

3.7.5 Application Interface

We developed an Android application to communicate via HTTP with the Android NetInf node. This application shows the advantages of using ICN NetInf approach. With the application the user has the possibility to create, share, get and delete IOs. More details will be given in section [4.1.1].

The application layer interacts with an external server developed by us, which is used to generate QR Barcodes. The identifiers of the NetInf objects are hash values (i.e. hash-256(content) encoded in base64URL) = GbMXD38aV6DQ03Sx8Cf9Cr8CWR8URmhe6HmMvI03hFw), therefore it was needed a simple way to get that identifier. In the scenario presented by this master thesis project, when a node share or create an IO, a HTTP request is sent to a QRcode server to generate a QR barcode with the hash value that was created. Afterwards, when another node wants to search for an IO, instead of type the whole hash value (unfriendly way), the node just have to scan the QRcode and get the identifier (hashvalue) in a more simple way. More information about this server and its functionality can be found in section [4.1.6].

3.7.6 Caching

3.7.6.1 IO Caching

The caching of IO is done using the SQlite database available in Android OS to store structured data. The hash algorithm and hash value are the elements used to cache IOs, and these can be found in the IO identifier.
3.7.6.2 Content Caching

The caching of content is done using the storage database available in Android OS to store content. The android NetInf nodes will cache all the contents that they request. The name of the files that are cached is the IO identifier (i.e. hash256(content) encoded in base64URL). In the implementation the files are saved in a folder called “MySharedFiles”. An additional external content caching can be done using the Ni content server explained beforehand [3.7.4]. More details will be given in section [4.1.3].
4. Implementation

In this section, it is first presented the architecture proposed including a detailed explanation of all its components. The architecture is comprised by external and internal components. The internal components include: application layer, control layer and internal cache. The external components are: name resolution server, NI content server, QR code server and web server.

4.1 Architecture Overview

The architecture proposed in this project is shown in Figure 19.

As shown in Figure 19, the architecture is comprised by five main elements:

1. The application layer
2. The control layer
3. The Internal cache layer
4. External Mobile Data Storage (MDS) units of nearby devices
5. Different external servers

The application, control and internal cache layers are located within the mobile device.
and will be running in every NetInf-enabled device. The fourth component, the MDSs of nearby devices, will be running in adjacent NetInf-enabled devices running the same architecture. Finally, the external servers are not attached to any particular location; they just need to have IP connectivity to the Internet.

The application layer main functions will be to interact with the user and to be the interface to the control layer. Also, it will use the connection manager in order to participate in the discovery process of Bluetooth and Wi-Fi Direct. All the communication between the application and the control layer is done over HTTP. Finally, this layer will also communicate with the QR-code server (QRS) in order to allow an easy way of obtain the IOs’ names, which are hashes.

In the control layer run the two main components of the architecture:

- Node
- Transfer dispatcher.

The node is the component responsible for any operation related with IOs. It allows components running in the application layer to: get, put, delete and cache IOs. To provide these functions, the node will communicate with two components: one that resides within the device (Mobile Database (MDB)) and one that resides in the external network (Name Resolution Server (NRS)). The MDB will be used to resolve IOs locally and it will contain a list of all the IOs that have been put or cached by the user. The NRS will be an external entity that will work in a DNS fashion allowing NetInf-enabled devices to register IOs and get locators associated with registered IOs.

The transfer dispatcher is the responsible component for all BO handling (transferring, serving, storing, etc). Given a target IO, the transfer dispatcher is capable of retrieving the BO associated with it from a remote peer MDS or from a NI Content Server (NCS) and store it in the local MDS. This component will be also the responsible of listening for other adjacent peers requests for file transferring. Therefore, the transfer dispatcher is in charge of two servers: the Bluetooth server and the TCP server.

Finally, the internal cache layer is responsible of managing two different types of storage: the MDB which is a structured data storage unit based on a SQLite database and the MDS which uses the file data storage capability provided by Android.

The remaining sections in this chapter will explain each layer in detail including all the components developed within them.

4.1.1 Application Layer

The application layer is responsible for the following functions:

1. User interaction: based on a graphical user interface that expose to users the different functionality offered by the control layer.

2. Interaction with the control layer: it allows the users’ request to be redirected to the control layer. It will also receive and display to the user the results of the different requests. All the interaction with the control layer is HTTP based using GET and POST requests.
3. Discovery and Connection establishment: the application layer will be the responsible for performing the discovery process of nearby users for the selected transport technology. This process will be done before attempting to transmit any BO from a remote peer in order to avoid a connection attempt to a user that is not in the range of the selected transport technology. For the specific case of Wi-Fi Direct, this layer will also establish the Wi-Fi Direct connection.

4. Interaction with the QR code Server: the application layer is the responsible for the communication with the QRS. Whenever an IO is cached in or deleted from the local device, a HTTP request will be sent to the QRS. The detailed description of the interaction between the device and the QRS can be found in section 4.1.6.

The application layer is comprised by three main components:

1. User interface
2. Application node Interface
3. Connection manager

These components together with their respective sub-components are shown in Figure 20.

![Application Layer Diagram](image)

**Figure 20. Architecture application layer components**

All the components (User Interface, SendHttpGetRequestTask, SendHttpPostRequestTask, BluetoothDiscoveryTask, WifiDiscoveryAndConnectTask) shown in Figure 20 were contributions from this master thesis project for the application layer. None of them were inherited from OpenNetInf implementation.

**4.1.1.1 User Interface**

This component is in charge of all the interaction with the user. It should allow the user, via a set of input mechanisms, to decide which files will be shared in the network and to request the transmission of files published by other peers. This component is also responsible for keeping the user aware of the progress of the requested operation and notify in case of errors. Finally, it should allow the user to decide which transport technology should be used for file transferring.
4.1.1.2 Application-Node Interface

This component allows the application layer to communicate with the control layer using HTTP methods. This communication uses two types of HTTP methods depending on the component of the control layer that the application wants to communicate with as shown in Table 4.

<table>
<thead>
<tr>
<th>Target Control layer component</th>
<th>HTTP method to used</th>
<th>Input parameter</th>
<th>Passing parameter technique</th>
<th>Return parameter</th>
<th>Architecture component responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>GET</td>
<td>HASH</td>
<td>Included in the target URL</td>
<td>Information Objects</td>
<td>null</td>
</tr>
<tr>
<td>Transfer Dispatcher</td>
<td>POST</td>
<td>Information Object</td>
<td>POST entity</td>
<td>File path</td>
<td>null</td>
</tr>
</tbody>
</table>

As shown in Table 4, the possible return parameters when sending a request to the node can be: an Information object associated to the input hash (GET request success response) or null (when the IO have not been found or a put or delete method have been called). On the other hand, the possible return parameters when sending a request to the transfer dispatcher can be: the file path where the transmitted BO was stored after the transmission or null in case of a transferring error. The BO that the transfer dispatcher will attempt to transmit is the one represented by the IO given as input parameter.

4.1.1.3 Connection Manager

This component is responsible for running the discovery process, using the desired transport technology, before attempting to call the transfer dispatcher to transmit the desired BO. Since the locators registered in the NRS for a certain IO does not necessarily have to be within the transport technology range of the requesting peer, the device must perform a discovery process in order to determine if any of the target locators returned by the NRS for a specific IO is within the transport technology range Figure 21 shows an example:
Figure 21. Discovery process example

The process of retrieving a file from a nearby peer is comprised by the following steps:

1. Request to the NRS the available locators associated with the IO
2. Run a discovery process for the selected transport technology (e.g. Bluetooth) in order to identify nearby peers.
3. Map the list received by the NRS to the list of discovered peers i.e. find peers present in both lists.
4. If there is at least one peer in the range who has the target BO, attempt to connect to it and retrieve the file.
5. If none peer in the transport technology range has the target file, send an error message to the user.

Now, let’s assume that the “Requestor” device would like to retrieve the file represented by “Hash-A” using Bluetooth as the transport technology. In order to do that, the “Requestor” will send an HTTP request to the NRS asking for locators associated with the “Hash-A”. As a result of that query, the NRS will return the Device A’s Bluetooth address since it is the only one that has registered that IO in the NRS. The “Requestor” device will run a discovery process and find out that “Device A” is within its Bluetooth range and thus, the “Requestor” will request the transfer dispatcher to retrieve the file from “Device A”. If the transfer is successful, the “Requestor” will now have the file represented by “Hash-A”.

On the other hand, let’s now assume that the “Requestor” would like to retrieve the file represented by “Hash-B”. This time the response received from the NRS will contain the Bluetooth and Wi-Fi address of “Device B”. If the “Requestor” would like to use again Bluetooth as the transport technology, the discovery process ran by the Connection Manager will not find “Device B” within the Bluetooth range and therefore it will tell the application layer that the file could not be retrieve.

Without the discovery process run by the Connection Manager, the “Requestor” would
have attempted to connect to all devices listed in the locator list for the selected transport technology regardless of their location. This could lead to an inefficient retrieving process in case that several users out of the range of the device will be obtained as possible sources of the file.

Finally, since the Connection Manager alerted the “Requestor” that there is none device within the Bluetooth range that has the desired file, the user attempts to transfer the file over Wi-Fi. As can be seen in the Figure 21, this process will be successful and it will find out that “Device B” is within the Wi-Fi range and that it has the desired file. Therefore, a Wi-Fi connection can be established between “Device B” and the “Requestor” in order to retrieve the file.

**Bluetooth**

When Bluetooth is the transport technology to be used, the BluetoothDiscoveryTask component will be used in order to run the discovery process and match the list of peers obtained by the NRS with the peers within the Bluetooth range. The list of matching peers (peer’s address present in the NRS response and also within the Bluetooth range) will be included in the IO that will be sent to the transfer dispatcher in order to retrieve the desired BO.

**WIFI Direct**

When Wi-Fi Direct is the transport technology to be used, the WifiDiscoveryAndConnectTask component will be used in order to run the discovery process and match the list of peers obtained by the NRS with the peers within the Wi-Fi range. In addition, this component will also be responsible for creating the Wi-Fi Direct connection to the desired peer. After the Wi-Fi Direct connection is setup, the IP address of the Wi-Fi Direct group is added as an attribute to the IO. This address will be used by the transfer dispatcher to retrieve the file from the remote peer.

### 4.1.2 NetInf Control Layer

The NetInf control layer is based on the OpenNetInf architecture components described in section 2.4. This layer is responsible for the following functions:

1. Listen for request from the application layer: this layer is continuously listening for IO and BO HTTP requests from the application layer. Once a request is received, the control layer will forward it to the appropriate subcomponent (e.g. Node or Transfer Dispatcher).

2. IO related operations: this layer is responsible for IO resolution, both locally and remotely, and IO caching and deleting. For local resolution, this layer will interact with the local MDB to determine if the IO it is stored locally. For remote resolution, this layer will use the 3G uplink to interact with the NRS using light weight HTTP requests.

3. Transfer BOs: the control layer offers BO transferring services for the different transport technologies (Bluetooth, Wi-Fi Direct and HTTP). These transfer services can be used to retrieve the target BO from a nearby peer using the desired transport technology.

4. Listen for transferring requests: this layer is responsible for listening for possible transfer request received from adjacent peers.
The NetInf Control layer is comprised by three elements:

1. HTTP Server
2. Node
3. Transfer Dispatcher

These three main components together with their respective sub-components are shown in Figure 22.

![Figure 22. Architecture control layer components](image)

The components shown in Figure 22 that were contributions from this master thesis are: HTTP Server, Local Resolution Service, Remote Resolution Service, Bluetooth Provider, TCP Provider, HTTP Provider, TCP BO server, Bluetooth BO server. On the other hand, the components that were inherited from OpenNetInf implementation were: Node, Resolution Controller, Transfer Dispatcher, Locator Selector.

### 4.1.2.1 HTTP Server

The control layer has an open HTTP server, which is the interface for the application layer for all interaction with the node or the transfer dispatcher. This server runs in the local host address (port 8080) and will process only local requests coming from the application layer. The server can process HTTP GET and HTTP POST request. When a request is received, the HTTP server will extract the parameters from it and call the respective control layer component (node or transfer dispatcher) depending on the type of request. The server was implemented using Restlet library [38].

Figure 23 shows an example the HTTP Server processing 2 different types of request.
In Figure 23, the HTTP server running in localhost port 8080 processes two requests:

1. The first request is an HTTP GET requesting the control layer to “GET” the IO associated with “HASH”. Since the request is an HTTP-GET request, the HTTP server decides that it must be forwarded to the node. Moreover, the two parameters contained in the request (“HASH” and “GET”) are used by the HTTP server to determine that the application is requesting a GET IO operation for “HASH”. The node will then process the request and return, if found, the IO associated with “HASH”. Finally the HTTP Server will send back to the application layer an HTTP response containing the IO found.

2. The second request is an HTTP POST requesting the control layer to transfer the BO associated with the IO sent on the POST entity. Since the request is an HTTP-POST request, the HTTP server decides that it must be forwarded to the node. The HTTP server will extract the IO contained in the POST request and call the get bytes function of the Transfer dispatcher. Once the bytes have been transferred and the file has been stored in the file system, the transfer dispatcher will return the file path where the file was stored to the HTTP server. Finally, the HTTP server will create a HTTP response containing the file path to the application layer.

4.1.2.2 Node

The Node is the component of the control layer responsible for handling all the operations related to resolving, putting and deleting IOs. In order to do that, the Node is comprised by four subcomponents:

1. Node
2. Resolution controller
3. Local Resolution Service
4. Remote Name Resolution Service

Node

This component was inherited from the OpenNetInf implementation and its function is to process every NetInf request in the control layer and determine a suitable controller (e.g. resolution, search or transfer controller) to handle the request is available. For the architecture used in this project the node will only use a resolution controller; however other controllers could be added in the future if necessary.
Resolution Controller

This component was inherited from the OpenNetInf implementation and is the entity that manages all the resolution services available in the control layer. Depending on the priority assigned to the different resolution services that it has registered, the resolution controller will forward the resolution request received from the node to the respective resolution service. This process is iterative, meaning that the resolution controller will attempt to resolve the request using the resolution service with highest priority and, if this query it not successful, the request will be for forward to the resolution service with second highest priority and so on. This process will continue until a successful response has been found for the request or there are no more resolution services registered. As soon as a response have been obtained by one of the resolution services, the response it forward to the node and the resolution process is finished. If none of the resolution service was able to resolve the request, the resolution controller returns “null” to the node.

In our implementation, two resolution services are registered and managed by the Resolution Controller: the Local Resolution Service and the Remote Name Resolution Service. The priority of the Local Resolution Service has been set higher than the Remote Name Resolution Service one; therefore all requests will try to be resolved locally before attempting to use a remote resolution service.

Local Resolution Service

The local resolution service is used to register and resolve IO locally in the MDB. Therefore, this service can be seen as the control layer interface to the MDB. This service offers three operations: get IO, put IO and delete IO. The information registered into the MDB to put an IO is just the hash and the hash algorithm of the IO as can be seen in Figure 24.

![Figure 24. Local resolution service putting and IO into the MDB](image)

The get IO operation offer by the Local Resolution Service will search for the existence of the request hash in the MDB. If there is a match in the MDB, an IO is creating using the hash and the hash algorithm register in the MDB and afterwards, this IO is returned as the result of the query. If not matches are found in the MDB, the Local Resolution Service will return null.

Finally, the delete IO operation is used to delete entries from the MDB associated with the hash sent in the request, if any.

Remote Name Resolution Service

The Remote Name Resolution Service is used to register and resolve IO remotely in the NRS. Therefore, this service can be seen as the control layer interface to the NRS. This service offers
three operations: get IO, put IO and delete IO. The communication with the NRS is done over HTTP using the 3G uplink using light weight request. The messages type used are HTTP POST messages that contain a multipart entity where all the identification parameters of the IO together with the locators associated with the registering peer are put. The parameters included in the HTTP POST to register an IO are the following:

1. **NI URL**: a ni URL containing the hash and the hash algorithm of the IO to be registered. For example:
   
   ```
   Hash = GhMxD38aV6DQ03Sx8Cf9Cr8CWR8URmhe6HmMvI03
   Hash Alg = sha-256
   NI URL = ni:///sha-256;GhMxD38aV6DQ03Sx8Cf9Cr8CWR8URmhe6HmMvI03
   ```

2. **Message Id**: id number used by the server to keep track of the message and its replies.

3. **Bluetooth locator**: the Bluetooth MAC address of the peer registering the IO if available. The MAC address will be prefixed with the “nimacbt://” (NI MAC Bluetooth).

4. **Wi-Fi locator**: the Wi-Fi Direct MAC address of the peer registering the IO if available. The MAC address will be prefixed with the “nimacwd://” (NI MAC Wi-Fi Direct).

Figure 25 shows an example of the Remote Name Resolution Service putting and IO into the NRS:

![Figure 25. Remote Name Resolution Service putting and IO in the NRS using HTTP POST](image)

As can be seen in Figure 25, the Remote Name Resolution Service uses the Information object, the Bluetooth MAC address and the Wi-Fi Direct MAC address in order to build the HTTP POST request. If any of the addresses is not available (e.g. adapter turned off, technology not supported in the device), this field will not be present in the POST request. Therefore, is completely possible to register and IO with only the Bluetooth or Wi-Fi Direct address. A POST request without any locator will be rejected by the NRS. The communication between the Remote Name Resolution Service and the NRS is done using the 3G link over HTTP.

The Remote Name Resolution Service “get IO” operation allows the control layer to
retrieve IOs registered by other users in the NRS. This operation also uses HTTP POST method, but the request entity only contains the target NI URL. All the locators registered in the NRS for the requested NI URL will be returned in the HTTP response as a JSON Array. If the array contains at least one locator, the Remote Name Resolution Service will create an IO using the hash and hash algorithm contained in the NI URL and it will add all the received locators as attributes to this IO. A random local priority value will be assigned to every locator; this priority will be used by the Locator Selector (Transfer Dispatcher). The resulting IO will be returned to the resolution controller as result of the query.

Finally, the delete IO operation is done using the same procedure and parameters as the put IO operation.

4.1.2.3 Transfer Dispatcher

The Transfer Dispatcher is the component of the control layer responsible for handling all the operations related to serving and retrieving BOs. In order to do that, the Transfer Dispatcher is comprised by four subcomponents:

1. Transfer Dispatcher
2. Locator Selector
3. Providers
4. BO servers

4.1.2.3.1 Transfer Dispatcher

This component was inherited from the OpenNetInf implementation and its function is to select the suitable provider to transfer a BO, given a certain IO as input. A provider is an entity capable of retrieve the byte array associated with a BO using a specific technology (e.g. Bluetooth). The different providers are registered when the Transfer Dispatcher is initialized. The architecture proposed in this master thesis includes three different providers: Bluetooth provider, TCP provider and HTTP provider.

When the transfer dispatcher is called from the HTTP Server, it receives an IO as input. Depending on the locators attributes contained in the IO, the transfer dispatcher will select a suitable provider to transfer the file. The prefixes of the locators are used to determine the suitable provider for a given locator as shown in Table 5.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>nimacbt://</td>
<td>Bluetooth provider</td>
</tr>
<tr>
<td>niip://</td>
<td>TCP provider</td>
</tr>
<tr>
<td>nihttp://</td>
<td>HTTP provider</td>
</tr>
</tbody>
</table>

As can be seen in Table 5, the prefix “nimacwd://” associated with the Wi-Fi Direct MAC addresses is not
present. This is due to the fact that after the “WifiDiscoveryAndConnectTask” has successfully connected to the remote peer, the protocol used to transfer the files is TCP using the IP of the group owner instead of the Wi-Fi Direct MAC address.

4.1.2.3.2 Locator Selector

This component was inherited from the OpenNetInf implementation and it is used in case that the IO received as input contains more than one locator for the target file. The locators will be used in a certain order depending on a priority attribute that each locator has; the Locator Selector is the responsible component for arrange the locators according to their priority.

The assignment of the priority for the locator is implementation dependent, meaning that different implementation can have different criteria for assigning the priority. For the implementation done in this master thesis, the priority is a local random value used in order to avoid that all the peers that want to retrieve a file use the same source. This priority is assigned by the remote name resolution service when the NRS returns the set of locators for a certain IO and it is a local temporary value used only for the device requesting the IO. Since it is a local value, even the same IO can have different locators’ priorities in different devices as shown in Figure 26.

![Image of Locator Selector](image)

**Figure 26. Use of the locator priority by the Locator Selector**

Figure 26 shows two different devices that want to retrieve the same BO identified by “hash1”. Both devices send a request to the NRS asking for all the locators associated with “hash1”. As can be seen from the NRS table, “hash1” has three locators associated with it; these three locators will be returned to each of the devices as result of the request. When “Device A” receives the list from the NRS, it assigns a local random priority value to each of the locators. “Device B” will do the same procedure but it will have its own local priority values. When “Device A” calls the Transfer Dispatcher to transfer the file, the Locator Selector will pick...
as first option for transferring the target devices with “BluetoothMAC-2”. On the other hand, “Device B” Locator Selector will pick the target devices with “BluetoothMAC-3”. This mechanism will work in a load balancing fashion avoiding overloading one device with all requests for the same file.

### 4.1.2.3.3 Providers

Once the list of locators contained in the IO have been arranged by their priority by the Locator Selector, the Transfer Dispatcher will select, based on the locator prefixes, which provider will be used for the file transferring (Table 5).

A provider is a component responsible for transferring the byte array corresponding to a certain BO from a remote peer. It receives two input parameters:

1. **Hash**: the hash of the BO that will be transferred from the remote peer
2. **Locator**: the address (e.g. Bluetooth MAC, IP, etc.) of the remote peer that has the requested BO

If the BO has been successfully transferred, the provider will return the byte array stream containing the BO’s bytes; this byte stream will be used to create the file that represents the BO.

The providers developed by this master thesis are the following:

1. **Bluetooth provider**: creates a Bluetooth client socket that will connect to the remote peer and will transfer the BO over Bluetooth. This provider will be used to transfer files from external MDSs when Bluetooth is the selected transport technology.
2. **TCP provider**: creates a TCP client socket that will connect to the remote peer and will transfer the BO over the TCP connection. This provider will be used to transfer files from external MDSs when Wi-Fi Direct is the selected transport technology.
3. **HTTP provider**: this provider will send a HTTP GET request to an HTTP server and it will receive the BO in the HTTP response. This provider will be used to transfer files from the NCS when 3G is the selected transport technology.

### 4.1.2.3.4 BO servers

So far, it has been explained the components of the Transfer Dispatcher that are used when the devices have the client role. However, in order to be able to connect to other peers to retrieve BOS, every node needs a server that is waiting for requests from other nodes; these servers are called BO servers.

The BO servers will have a server socket listening for possible connections. Once a socket connection is established, the BO server will received BO-transfer requests from the client that start the socket connection. When the client sends a request for a certain BO, the BO server will check for the existence of the file and, if it is found in the local MDS, the file will be sent to the requestor over the socket connection.

Once the socket connection is established, the provider and the BO server need a way to agree on which BO the requestor wants to retrieve from the server. The implementation in this master thesis has created a small signalling protocol to allow clients and servers to exchange
BO requests and response. The signalling protocol uses the “Transferring Messages” which were designed and created using Google’s protocol buffers [41]. A Transferring message is comprised by two fields:

1. Request code: code that identifies the type of message. It has three possible values: REQUEST, REPLY_OK and REPLY_ERROR.
2. Data: a string value used for clients to include the hash of the file to be retrieved.

Figure 27 shows an example of how the signalling protocol is used:

![Diagram of communication process between Device A (Requestor) and Device B (Server)](image)

Figure 27 shows the communication process between the provider of “Device A” and the BO server of “Device B”. The process is comprised by the following steps:

1. The transfer dispatcher will give as input to the provider the hash of the target BO and the target locator, which for this example will be “Device B” address. (e.g. Bluetooth MAC or IP address)
2. Using the locator, the provider will attempt to connect to the server socket of “Device B”.
3. The BO server of “Device B” will accept the connection and pass the connected socket to the connected socket handler, which will now wait for transferring messages from the provider.
4. The provider will create a transferring message of type “Request” and will include the hash of the file that it wants to retrieve.
5. Once the “Request” transferring message is received by the Connected Socket handlers, it will check in the MDS if the requested file exists. If the file is found, it will respond back to the provider with a transferring message of type “Reply_OK”. If the file is not found, it will respond back to the provider with a transferring message of type
“Reply_ERROR” and the socket will be closed.

6. If the file was found, the BO server will transfer the BO’s byte array to the provider.

7. Once the byte array has been successfully transferred, the socket connection is closed and the provider will return the byte array to the transfer dispatcher.

The BO servers developed by this master thesis are the following:

1. **Bluetooth BO server**: BO server that listens to Bluetooth socket connections and transfer BOs over it to the respective provider. This BO server will be used when Bluetooth is the selected transport technology.

2. **TCP BO server**: BO server that listens to TCP socket connections and transfer BOs over it to the respective provider. This BO server will be used when Wi-Fi Direct is the selected transport technology.

### 4.1.3 Internal Cache

The internal cache layer is responsible for the following functions:

1. **Provide structured data storage**: this layer provides structured data storage by a SQLite database used for storing IOs. This component allows querying and deleting functions.

2. **Provide file storage**: this layer is responsible for BO storage using the device’s available storage system.

The Internal Cache Layer is comprised by two elements:

1. **Mobile Database (MDB)**
2. **Mobile Data Storage (MDS)**

These two components are shown in Figure 28:

![Figure 28. Use of the locator priority by the Locator Selector](image)

The development of the Mobile Data Base was a contribution from this Master thesis, while Mobile Data Storage functionality was taken from the Android API.

**Mobile Database (MDB)**

This component is a local database where all the IOs that the application has put or cached will be stored. Using the SQLite built-in capabilities of the Android operative system, we developed the database; the decision of what parameters are stored in the database and the format in which these values are stored is a contribution of this master thesis. The database contains only one table with the following fields:
Table 6. MDB table description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Mandatory</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Integer</td>
<td>YES</td>
<td>Auto incremental value that is assigned to every entry in the table</td>
</tr>
<tr>
<td>HASH_ALG</td>
<td>String</td>
<td>YES</td>
<td>Contains the hash algorithm used to create the hash of the content</td>
</tr>
<tr>
<td>HASH_CONTENT</td>
<td>String</td>
<td>YES</td>
<td>Hash of the content represented by the IO</td>
</tr>
</tbody>
</table>

The MDB offers the following three possible operations:

1. **Put**: used to create a new entry in the table
2. **Get**: given a certain hash as input, this operation allows the control layer to check if there is an entry in the table with the same hash.
3. **Delete**: used to delete entries from the table given a specific hash

The control layer interface to the MDB is represented by the local resolution service, which is the only component capable of interacting with the MDB.

### 4.1.3.1 Mobile Data Storage (MDS)

The MDS is the file system where BOs are stored. It is implemented simply by using the android file management system.

Files that are shared or cached by the NetInf application created in this master thesis are copied to a specific directory in the external storage public directory (SD card) called “MySharedFiles”. This folder allows the user to keep track of which files have been shared to the network.

Figure 29 shows an example of the usage of the MDB and the MDS when the NetInf application developed in this master thesis wants to share a picture to the NetInf network.
In Figure 29, it is shown an example of what happens in the internal cache layer when a user uses the “Share” function of the NetInf application in order to share a picture to the network. The user will start by picking the file to be shared. Once the user has selected the file, the application will hash the content of it and do the following two steps:

1. It will send and HTTP request to the control layer requesting to PUT and IO. Once the control layer has received the request, it will create an IO containing the hash of the picture and the hash algorithm used and it will create an entry in the MDB with those values.

2. The application will also store a copy of the file in the “MySharedFiles” folder which is part of the MDS.

### 4.1.4 Name Resolution Server

The NRS is used as a remote resolution service to resolve IOs. Android nodes can register and retrieve IOs and the list of locators associated to them.

In Figure 30, it is shown how two different android NetInf nodes are registering the same object but with different MAC address. The content in the first node is available through the WIFI Direct interface (see the prefix, nimacwd://) and in the second node is available through Bluetooth interface (see the prefix, nimacbt://). Both entrances are stored by the NRS under the same niname both with the two locators. If a third node wants to retrieve that object and is resolving IO with the NRS, the reply will be a <list of locators> (in this case two MAC addresses). See Figure 31.
Figure 31. NRS (GET requests and GET/Response request)

The communication between the NRS and the android NetInf nodes is through HTTP and is following the “HTTP NetInf Convergence Layer” a document written as part of the work done within the WPB Sail project group. The document specifies how NetInf protocol messages are mapped to HTTP and how NetInf nodes are expected to process them:

GET/GET-RESP

The Get/Get-RESP request/response pair messages (used when a NetInf node wants to get locators or actual content (BO) are mapped to an HTTP POST requests with the request parameters in a form data set. The response message can either contain the requested content, a list of locators or an error code. Table 7. GET request parameters (HTTP POST)Table 7 shows the parameters of this type of request:

Table 7. GET request parameters (HTTP POST)

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Data Type</th>
<th>Optional/Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>String</td>
<td>MANDATORY</td>
<td>The ni name for the requested Object</td>
</tr>
<tr>
<td>Msgid</td>
<td>String</td>
<td>MANDATORY</td>
<td>The message ID</td>
</tr>
<tr>
<td>Ext</td>
<td>String</td>
<td>OPTIONAL</td>
<td>Place holder for protocol extensions</td>
</tr>
</tbody>
</table>

Table 8 shows the response code for the GET request:
Table 8. Response code for GET requests

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK—object provided in message body</td>
</tr>
<tr>
<td>404</td>
<td>Not Found—have no information on specified object</td>
</tr>
</tbody>
</table>

If the response code is 200, the message body MUST provide either the actual content (BO) or a list of locators. The list of locators is represented as a JSON list of String. The NRS will response just with locators, further it will be introduced another external entity called NI content server where nodes can either cache or retrieve content (using the same server implementation and communication process).

**PUBLISH/PUBLISH-RESP**

The Publish/Publish-RESP request/response pair messages (used when a NetInf node wants to publish either locators or content) are mapped to an HTTP POST requests with the request parameters in a form data set. The request message can either contain content to be published or a list of locators. Table 9 shows the parameters of this type of request:

Table 9. PUBLISH requests parameters (HTTP POST)

<table>
<thead>
<tr>
<th>Form Field</th>
<th>MIME Type</th>
<th>Optional/Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>String</td>
<td>MANDATORY</td>
<td>The ni name for the object to be published/registered</td>
</tr>
<tr>
<td>Msgid</td>
<td>String</td>
<td>MANDATORY</td>
<td>The message ID</td>
</tr>
<tr>
<td>Octets</td>
<td>Application/octet-stream</td>
<td>OPTIONAL</td>
<td>The bits of the object to be published</td>
</tr>
<tr>
<td>Loc1</td>
<td>String</td>
<td>OPTIONAL</td>
<td>A locator for the object to be registered</td>
</tr>
<tr>
<td>Loc2</td>
<td>String</td>
<td>OPTIONAL</td>
<td>A locator for the object to be registered</td>
</tr>
<tr>
<td>Fullput</td>
<td>String</td>
<td>OPTIONAL</td>
<td>If present, signals that the message contains an object</td>
</tr>
</tbody>
</table>

Table 10 shows the response code for the PUBLISH requests:

Table 10. Response code for PUBLISH requests

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK—object/locators accepted</td>
</tr>
<tr>
<td>404</td>
<td>Not Found – TBD</td>
</tr>
</tbody>
</table>
UNPUBLISH/UNPUBLISH-RESP

The Unpublish/Unpublish-RESP request/response pair messages are used when a NetInf node wants to unpublish locators from the NRS. These requests are mapped to an HTTP POST requests with its parameters in a form data set. The request message will contain the list of locators to be deleted from the NRS. Table 11 shows the parameters of this type of request:

Table 11. UNPUBLISH requests parameters (HTTP POST)

<table>
<thead>
<tr>
<th>Form Field</th>
<th>MIME Type</th>
<th>Optional/Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>String</td>
<td>MANDATORY</td>
<td>The ni name for the object</td>
</tr>
<tr>
<td>Msgid</td>
<td>String</td>
<td>MANDATORY</td>
<td>The message ID</td>
</tr>
<tr>
<td>Loc1</td>
<td>String</td>
<td>OPTIONAL</td>
<td>Locator to be unpublished for the object</td>
</tr>
<tr>
<td>Loc2</td>
<td>String</td>
<td>OPTIONAL</td>
<td>Locator to be unpublished for the object</td>
</tr>
<tr>
<td>LocN</td>
<td>String</td>
<td>OPTIONAL</td>
<td>Locator to be unpublished for the object</td>
</tr>
</tbody>
</table>

Table 12 shows the response code for the UNPUBLISH requests:

Table 12. Response code for UNPUBLISH requests

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK—locators unregistered</td>
</tr>
<tr>
<td>404</td>
<td>Object Not Found</td>
</tr>
</tbody>
</table>

4.1.5 Ni Content Server

As mentioned beforehand [3.7.4], the Ni Content Server (NCS) is used to externally cache and retrieve content. This external entity provides an additional functionality for cache content in an external server through 3G. When users desire to share content to the network, they can select if they want to make this content available only in their device or in their device and the NCS.

The NCS component can be used by content distributors as a bootstrap node for content distribution. As an example, the stadium owner could set up a NCS an upload to it content related to the ongoing event. Then, users could start fetching the content from the NCS using 3G since the NCS has IP connectivity over the internet. Once content start to become popular in the crowd, user can start fetching the content from nearby peers using short range transport
technologies instead of using the 3G uplink.

The implementation used is based on the niproxy server developed by Nokia Siemens Network, also utilized for the name resolution server. The behavior is similar with the only different that the PUBLISH requests done by the Android devices include the bits of the content attached in the body of the request. See Figure 32.

<table>
<thead>
<tr>
<th>Ni name</th>
<th>&lt;list of locators&gt;</th>
<th>Content</th>
</tr>
</thead>
</table>
| sha-256;Mylo0:0e54mmn9G6jvLzye1vGW0ip8oq342XXowwr_w | nimaciw://67:87:98:HG:7G | ![content]

The Niproxy server, also utilized for the name resolution server, performs the following functions:

- The PUBLISH requests done by the Android devices include the bits of the content attached in the body of the request.
- The behavior is similar to the niproxy server.

Figure 32. Ni Content Server (Publish requests, including content)

When Android devices cache the content in the NCS, the content becomes available for other peers from the android device who cache the content but also from the external server (which has IP connectivity over the Internet). Therefore, if another device wants to retrieve that content, it will have the option to retrieve either from the Android device using Wi-Fi Direct or Bluetooth, or from the external server using 3G.

As shown in Figure 32, when content is cached in the external server, a request has to be sent to the NRS to register the locators available for this IO.

Figure 33 shows an example of a node that does not have any mobile device in the Bluetooth or Wi-Fi Direct range with the content desired, and therefore it will use the NCS as the content source.
In Figure 33, the user will first send a request to the NRS in order to find out the available locators for the target IO. For the example shown in Figure 33, the user receives a Wi-Fi Direct locator and a NCS locator. Assuming that there are not any peers in the Wi-Fi Direct range for the user, the only possibility for the user to retrieve the content will be using the NCS. Therefore, the Android device will create a request for the NCS containing the hash of the desired IO. Finally, since the content is available in the NCS, the content will be sent back to the user in the HTTP response.

The communication between the Android NetInf node and the NI content server is done following the same HTTP convergence layer explained beforehand when the NRS was introduced. The different in the response for the GET request is that the message body will provide a BO (content) instead of a list of locators.

4.1.6 QR code Server

As was explained beforehand [3.7.5] within this master thesis was developed an external QRcode server that interact with the android NetInf node to generate QRbarcodes. The content generated by this server is then showed in a webpage. This functionality is useful for the nodes to read the hash value or identifiers when a node wants to perform a search request.
The QRcode Server is a HTTP server that was developed using the RESTLET framework. The main functions of the server are:

- Generate QRBarcode from a hasValue input (sent as GET HTTP request from the nodes). The server creates a .jpg file with the QR barcode image.
- Create a .txt file with the Object that have been created and the corresponding Bluetooth MAC address, WIFI Direct MAC address and the external cache URL. The .txt together with the .jpg files generated with the QRcode is used by a webserver (further explained) that is showing the content in a webpage.
- Delete the objects created in the .txt when a node has unpublished an Object to the NRS.

The server has two server resources associated, BOServer and DeleteServer:

**BOServer Resource**

This resource is in charge of generating the QRBarcode image and the .txt with the object created. The requests that will be redirected to this resource have the following form:

```
http://{serverAddress}/.well-known/ni/{hash_alg};{hashvalue};{macBTAddress};{macWIFIAddress};{httpServer}
```

*Figure 35. Form of the request redirected to the BOServer resource*
Example:

http://192.36.165.136:8182/.well-known/ni/sha-256;GhMXD38aV6DQ03Sx8Cf9Cr8CWR8URmhe6HmMvI03hFw;F0:E7:7E:3F:D2:43; a2:0b:ba:e0:94:54;192.36.165.136:8183

Figure 36. Example of a request redirected to the BO server resource

Figure 36 shows the form of the GET request that the nodes send to the QRcode server in order to generate the QRbarcode and .txt file. The first field 192.36.165.136:8182 is the address and port of the VM running the server. Then sha-256 is the algorithm that was used to hash the content. The third field is the hash value. Afterward, it can be found in strict order the Bluetooth MAC address, WIFI Direct MAC address and finally the IP address of the external content cache. The MACs (BT and WIFI) and IP address of the content cache server can have the value “NONE” when the object is not accessible for some of the interfaces or is not available in the external cache server.

The QRcode image and .txt files are created in the “C:\inetpub\wwwroot” folder that is being used by the web server to upload the content to the website. As was mentioned before the web platform used is based on the internet information server (IIS 7 in windows).

The request showed in Figure 36 will create a .txt file with the following entrance:

Figure 37. .txt file created from the entrance showed in Figure 36.

Also, the QR code generated from the request showed in Figure 36 is (you can try scanning the barcode with a standard barcode reader and check that the result will be the hasValue):

Figure 38. QRcode generated from the request showed in Figure 36
Delete Server Resource

This resource is in charge of deleting an object when an unpublished request is sent by the android nodes. The requests that will be redirected to this resource have the following form:

\[
\text{http://(serverAddress)/.well-known/ni/delete/{hash_alg};{hashvalue};{macBTAddress};{macWIFIAddress}}
\]

**Figure 40. Form of the request redirected to the Delete server resource**

Example:

http://192.36.165.136:8182/.well-known/ni/delete/sha-256;GhMXD38aV6DQ03Sx8Cf9Cr8CWR8URmhe6HmMv03hFw;F0:E7:7E:3F:D2:43; a2:0b:ba:e0:94:54

**Figure 41. Example of a request redirected to the Delete server resource**

The request showed in Figure 41 will remove the WIFI and Bluetooth MAC registered to that object.

Let’s say that there is an object created and two nodes have cached that content.

<table>
<thead>
<tr>
<th>QR Code</th>
<th>Hash Value</th>
<th>Bluetooth MAC Address</th>
<th>WiFi Direct MAC Address</th>
<th>External Cache Serv</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="QR Code" /></td>
<td>GhMXD38aV6DQ03Sx8Cf9Cr8CWR8URmhe6HmMv03hFw</td>
<td>[F0:E7:7E:3F:D2:43, F1:E7:7E:3F:D2:43]</td>
<td>[a2:0b:ba:e0:94:54, a1:0b:ba:e0:94:54]</td>
<td>192.36.165.136:8182</td>
</tr>
</tbody>
</table>

**Figure 42. Example of an object with two MAC address associated**

If a node sent a request showed in Figure 41 the two MAC address will be deleted and
the result showed in the web site will be:

![QR Code](image)

<table>
<thead>
<tr>
<th>QR Code</th>
<th>Hash Value</th>
<th>Bluetooth MAC Address</th>
<th>WiFi Direct MAC Address</th>
<th>External Cache Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>C: \ inetpub \ wwwroot</td>
<td>19.2.16.150.136:8</td>
<td>F1: E7: 7E: 3F: D2: 43</td>
<td>a1: 0b: 0e: 94: 54</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 43. Result after the delete request showed in Figure 41*

### 4.1.7 Web Server (IIS7)

Internet information server (Windows IIS 7) is the web server that is used to show the QR code generated and show the objects that have been created. IIS7 is a web server application and set of feature extension modules created by Microsoft for use with Microsoft Windows. It can support HTTP, HTTPS, FTP, FTPS, SMTP and NNTP.

IIS7 is not turned on by default when Window is installed. The process to turn on the web server is:

- Go to “Control Panel”
- Click on “Program and Features”
- Click on the left panel “Turn Windows features on or off”
- Click “Internet Information Services”, and click OK (The server will be turned on)
- After the installation the folder “C:\inetpub” will be created. Inside this folder there is the folder C:\inetpub\wwwroot where you will find the index html file.

The HTML file written by this master thesis can be found in the wwwroot folder. A small piece of JavaScript code was included to give more dynamic behavior to the content shown in the web site.

The Web Site has a delete button that sends a request to the QRcode server to delete the whole content shown in the Web Site. This feature is useful to clean the screen when many objects have been created. To be able to send the delete request from the Web server to the QRcode server (running in another port) was necessary to turn on and install another IIS7 features:

- Enable ASP.NET (Turn Windows features on or off-> Internet Information Services-> World wide web services-> Application Development Features-> ASP.NET)
- Install URL Rewrite Module (version 2.0)
- Install Application Request Routing version 2.0
- When the modules and features mentioned before are installed, it is necessary to configure the rules to rewrite the requests. In the Internet Information Service (IIS7) Manager Directory there is the “URL rewrite option” that is used to configure the rules:
- Click URL Rewrite and add a new rule. To redirect the delete HTTP requests (delete button pushed) to the QR Server is necessary to add the following rule (Figure 45):

![Figure 45. Delete HTTP request redirected to QR server.](image)

In the implementation shown by this master thesis the web server was running in the port 82 and the QRcode server in the port 8182.

### 4.2 Code Overview

Within appendix [10.3 Code Package Overview], it can be found the content of the different Java packages of the application. Also, it is detailed through UML diagrams the connection between Java classes and methods.
5. Prototype Evaluation

The aim of the section is to evaluate the prototype created using the implementation described in Section 4. First, the aspects to be evaluated in the prototype will be defined. Afterwards, each of these aspects will be exemplified with a test case that will be used to discuss the performance of the prototype in terms of advantages and disadvantages.

The prototype developed within this master will be analyzed in terms of:

- **Connection time between two peers over the different short range transport technologies**: the connection time is defined as the interval that starts with the creation and initialization of the socket in the requesting peer side and, finishes when the connection have been setup and bytes can be exchanged between the two peers using the socket’s streams. The discovery time is not included in this value. The connection time will be evaluated for the three different transports technologies implemented in the prototype: Bluetooth 2.1, Bluetooth 3.0 and Wi-Fi Direct.

- **File transfer time between two peers over the different short range transport technologies**: the file transfer time is defined as the interval that starts when the requesting peer sends the “Transferring message” to ask the serving peer to start the target file’s transfer and, ends when the last byte is completely received by the requesting peer. The file transfer time will be evaluated for the three different transports technologies implemented in the prototype: Bluetooth 2.1, Bluetooth 3.0 and Wi-Fi Direct.

- **Interaction with external components**: the interaction of the prototype with the external servers (NRS, NCS and QRS) that comprised the solution proposed in terms of: bandwidth consumption measured in the NRS and NCS and times involved in the IO retrieving procedure.

5.1 Connection Time

In order to evaluate the prototype connection time between peers, a test case where two devices establish a connection over the different transport technologies (Bluetooth and Wi-Fi Direct) was run. Afterwards, the result of the test case is used in order to discuss the advantages and disadvantages and derived conclusions.

5.1.1 Connection time test case

5.1.1.1 Setup

The equipment used for the connection time test case is summarized in Table 13
Table 13 Equipment used for the Connection time test case

<table>
<thead>
<tr>
<th>Technology</th>
<th>Equipment</th>
</tr>
</thead>
</table>
| Bluetooth v2.1 + EDR   | – Two Sony Ericsson arc S LT18i (OS Android 4.0 ICS)  
|                        | – Eclipse Debugging mode                       |
| Bluetooth v3.0         | – Two Samsung Galaxy Nexus GT-I9250 (OS Android 4.0 ICS)  
|                        | – Eclipse Debugging mode                       |
| Wi-Fi Direct           | – Two Samsung Galaxy Nexus GT-I9250 (OS Android 4.0 ICS)  
|                        | – Eclipse Debugging mode                       |

Based on the equipment presented in Table 13, the setup used for the connection time for the Bluetooth case is shown in Figure 46 and for the Wi-Fi Direct case is shown in Figure 47.

Figure 46 Bluetooth connection time test case setup

Figure 47 Wi-Fi Direct connection time test case setup
As shown in Figure 47, for the Wi-Fi Direct case the connection time involved an extra interval beside the socket connection; the Wi-Fi Direct group connection time. This is the time taken from the moment the requesting peer has discovered a Wi-Fi Direct peer and the connect method of the Wi-Fi Direct API [33] is called, until the Wi-Fi Direct connection has been setup and the IP address of the group owner is available for TCP sockets connection. For the Wi-Fi Direct case, the total connection time is defined as the sum of the Wi-Fi Direct group connection time and the TCP socket connection time.

In order to measure these times, several time stamps were placed inside the code in the beginning and end of the code block that is associated with the connection establishment procedure. The logcat Android tool provided by Eclipse [43] was used in order to extract the time values used to calculate the connection time.

In this scenario, a file of 2Mbytes was transferred ten times for each of the target technologies (Bluetooth v2.1 +EDR, Bluetooth 3.0 and Wi-Fi direct). Afterwards, the average connection time was calculated for each of the cases.

5.1.1.2 Results

The average connection time obtained in this test case run for the different transport technologies are presented in Figure 48:

![Connection time comparison between Bluetooth v2.1 +EDR, Bluetooth v3.0 and Wi-Fi Direct](image)

**Figure 48** Connection time comparison between Bluetooth v2.1 +EDR, Bluetooth v3.0 and Wi-Fi Direct

5.1.1.3 Analysis

Figure 48 summarizes in one graph the average connection times from the transport technologies use in the prototype tests.

The results show that Bluetooth v3.0 is the transport technology that establishes the faster connection process. It can also be seen the huge different in the connection times between Wi-Fi Direct and the other two Bluetooth transport technologies. For Wi-Fi Direct, the process that takes most of the time is the Wi-Fi Direct group connection (AVG. 7.1345 seconds), while the TCP connection time takes only an average of 0.0191 seconds.

The reason why the Wi-Fi Direct connection takes so long is that in order to establish a Wi-Fi Direct group connection with another device the below steps have to be done:
- The requestor has to send an invitation to the adjacent device to join the Wi-Fi direct group and this has to be accepted by the remote device’s user. This invitation message can be shown using different types of configuration [44]. The configuration used in the implementation done in this master thesis is the Push Button Config (PBC), which shows a pop-up dialog to ask the user if he/she wants to establish a Wi-Fi Direct group connection with the requesting peer.

- In addition to this delay, once the remote peer has accepted to participate in the Wi-Fi Direct group, some other processes like: group owner negotiation and IP address assignments will also consume time.

For the Wi-Fi Direct case, the fact that the user needs to accept the connection introduces a human error in the measurements. For the connection process attempts, the Push Button Notification was accepted as soon as it popped up (one of the team members was holding the device just waiting for the dialog to appear). The average connection time could, however, be considerably longer if the user that holds the device receiving the connection request does not have his/her focus on the application (e.g. the user might be using another application, the user has the phone on silence mode, the user reads the notification but decides to answer it later, etc).

In contrast, for the Bluetooth case the only connection that needs to be established is the Bluetooth socket connection. This connection does not need any type of authentication from the user and therefore, it does not introduce any human error. However, allowing devices to establish connections without the user’s awareness can derive into security and privacy issues. These issues were out of the scope of this master thesis.

When we did the tests for WIFI Direct, it is also important to mention that the mobile devices were giving connection problems after two or three reconnections. The Wi-Fi Direct functionality has a bug that has been confirmed by Google developers [45] [46]. After two or three successfully connections with nearby peers using Wi-Fi Direct, the mobile devices cannot establish new connections until the devices are restarted. Therefore, in order to accomplish the 10 transfer attempts was necessary to restart the devices at least 4 times. There seems to be a problem with the Wi-Fi Direct adapter driver; this bug is expected to be fixed for the upcoming platform Android releases. The bug can also be seen while using the Wi-Fi Direct demo application created by Google [33].

Figure 48 also shows that there is not big connection time difference between Bluetooth v2.1 + EDR and Bluetooth v3.0. This is due to the fact that Samsung Galaxy nexus used for the tests just have the newer Bluetooth v3.0 but without the HS (high speed) feature added. Without this capability, the mobile devices cannot gain the benefits to carry data over the 802.11 radio link. The small difference that can be seen (a bit faster for the Bluetooth v3.0) is due to faster processing power capacity of the Samsung Galaxy Nexus. It is assumed that the connection request are processed faster with the Galaxy Nexus that have 1.2GHZ dual core over the single core 1GHZ processor that the Sony Ericsson arc S has.
5.2 File Transfer Time

In order to evaluate the prototype transfer time between peers, a test case where two devices transfer a file over the different transport technologies (Bluetooth and Wi-Fi Direct) was run. Afterwards, the result of the test case is used in order to discuss the advantages and disadvantages and derived conclusions.

5.2.1 File transfer time test case

5.2.1.1 Setup

The equipment used for the file transfer time test case was the same as the one used for the connection time test case (Section 5.1.1) and is summarized in Table 13. Based on the equipment presented in Table 13, the setup used for the file transfer time for the Bluetooth and Wi-Fi Direct case is shown in Figure 49.

![File transfer time test case setup](image)

Figure 49 File transfer time test case setup

In order to measure these times, several time stamps were placed inside the code in the beginning and end of the code block that is associated with the file transfer procedure. The logcat Android tool provided by Eclipse [43] was used in order to extract the time values used to calculate the connection time.

In this scenario, a file of 2Mbytes was transferred ten times for each of the target technologies (Bluetooth v2.1 +EDR, Bluetooth 3.0 and Wi-Fi direct). Afterwards, the average connection time was calculated for each of the cases.
5.2.1.2 Results

The average file transfer time obtained in this test case run for the different transport technologies are presented in Figure 50:

![Figure 50 Average file transfer time for the different transport technologies](image)

5.2.1.3 Analysis

The successful execution of this test indicates that the prototype was able to transfer files using the different transport technologies. As shown in Figure 50, the transfer speeds obtained show an expected behavior where Wi-Fi Direct has a faster transfer speed than Bluetooth. The difference perceived between the Bluetooth technologies is due to processing power capacity between the Sony Ericsson Xperia arc S and the Samsung Galaxy Nexus, as explained for the results obtained in the connection time test case.

Using together the results obtained in the two experiments (connection and file transfers time) a more interesting conclusion can be obtained. Based on Figure 48 and Figure 50, we could state that:

- The connection time for Wi-Fi Direct is higher than Bluetooth v2.1 + EDR and v3.0.
- The transfer file rate for Wi-Fi Direct is higher than Bluetooth v2.1 + EDR and v3.0.

Therefore, for the prototype implemented in this master thesis and assuming the connection times shown in Figure 48, it could be concluded that for transferring large amount of data the most suitable technology will be Wi-Fi Direct since once the connection is established, the file transfer speed is almost 10 times faster than for the Bluetooth technologies. However, in cases with short amount of data to be transferred, the difference between the technologies will not be considerable since the total time needed to transfer the file (connection time and transfer time) will be comparable between Wi-Fi Direct and Bluetooth technologies. An example of this behavior can be seen in Figure 51 and Figure 52:
Figure 51 Connection and transfer times for Bluetooth v3.0

Figure 52 Connection and transfer times for Wi-Fi Direct
Figure 51 and Figure 52 show the total connection and transfer times for Bluetooth and Wi-Fi respectively. The times shown correspond to the ten attempts of transfer a file with the prototype used in the file transfer test case. The size of the transfer file was 2Mbytes (picture taken with the device camera).

As mention beforehand, it can be seen that when using Bluetooth as the transport technology, the connection time is relative short in comparison with the time that takes to transfer the file. On the other hand, we see a complete opposite behavior for Wi-Fi Direct. For this technology, we can see that the time that take to establish the connection is relative higher than the time that take to transfer the file. Table 14 shows the total average time:

Table 14. Average times for Figure X and figure Y (File size = 2Mbytes)

<table>
<thead>
<tr>
<th></th>
<th>Bluetooth</th>
<th>Wi-Fi Direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average connection time (sec)</td>
<td>1.169</td>
<td>7.1536</td>
</tr>
<tr>
<td>Average transfer time (sec)</td>
<td>8.5201</td>
<td>1.275</td>
</tr>
<tr>
<td>Average total time (sec)</td>
<td>9.6891</td>
<td>8.4286</td>
</tr>
</tbody>
</table>

As shown in Table 14, the average time for the two technologies when transmitting the same file just differs in 1.26 seconds. From a user perspective, a difference in only one second could be seen as a negligible difference and, therefore, the performance of the two technologies will be similar. However, if a user were to transfer a bigger file like an HD recorded video, a single file can sometimes reach even hundreds of megabytes. For this type of files, the difference perceived in the total average time between Wi-Fi direct and Bluetooth would be much more considerable and, assuming again the connection times shown in Figure 48, using Wi-Fi direct will give a faster response to the user.

5.3 Prototype interaction with external components

In order to evaluate the prototype interaction with the three external components (QRS, NRS and NCS), a test case where four devices obtain an IO, and its respective BO, using the NetInf control layer was run. The transport technology used for this experiment was Bluetooth. The result of the test case is used in order to discuss the advantages and disadvantages and derived conclusions.

5.3.1 Interaction with the NRS and the NCS test case

5.3.1.1 Setup

The equipment used for the interaction test case was:

- 5 Android mobile devices with 3G connection, running the Android ICN NetInf approach application developed by this master thesis project.
- 1 NRS (to remotely resolve IO, see section 4.1.4)
- 1 NCS (to retrieve content from an external cache server, see section 4.1.5)
- 1 QRS (to obtain the hashes associated with the IOs, see section 4.1.6)
- Wireshark network protocol analyzer.

![Wireshark Diagram](image)

Figure 53 Interaction with external components test case

**Test case description**

- Wireshark was used in order to analyze the incoming packets (HTTP requests) to the NCS and the NRS.

- To obtain the Graphs that shows the bandwidth usage (Bytes/Sec), the Wireshark IO Graph functionality was used.

- The starting point of this scenario was when “Mobile 1” published a 2.8MB picture in the NCS. The content is cached in the “Mobile 1”’s cache but it is also cached in the external NCS. Now the content is available to all other devices at the NCS via 3G. “Mobile 1”’s Bluetooth MAC address and the IP address of the NCS are registered by “Mobile 1” in the NRS as available locators.

- A request for creating a QR code for the picture hash is sent by “Mobile 1” to the QRS. This request results in a new entry in the form of Figure 42. This hash will be called “target object hash”

- “Mobile 1” is put out of the Bluetooth and Wi-Fi range of all other mobile devices

- “Mobile 2” scans the QR associated with the “target object hash” and tries to obtain an IO associated with this hash. Since “Mobile 2” does not have a local copy of that IO, the
request is forwarded to the NRS. The NRS answers back with the locators available for the “target object hash” ("Mobile 1”‘s Bluetooth MAC and NCS IP)

- Since “Mobile 1” is out of “Mobile 2”‘s Bluetooth range, “Mobile 2” attempts to retrieve the content from the NCS. The content is successfully retrieved over the 3G network and a new locator ("Mobile 2”‘s Bluetooth MAC address) is registered in the NRS.

- Now, “Mobile 3” scans the QR associated with the “target object hash” and tries to obtain an IO associated with this hash. Since “Mobile 3” does not have a local copy of that IO, the request is forwarded to the NRS. The NRS answers back with the locators available for the “target object hash” ("Mobile 1”‘s Bluetooth MAC, “Mobile 2”‘s Bluetooth MAC address and NCS IP)

- Since “Mobile 2” is within “Mobile 3”‘s Bluetooth range, the transfer of content is done over Bluetooth. The content is successfully retrieved and a new locator ("Mobile 3”‘s Bluetooth MAC address) is registered in the NRS.

- The same retrieval procedure used by “Mobile 3” is used by “Mobile 4” and “Mobile 5”. It is important to mention that “Mobile 2”, “Mobile 3”, “Mobile 4” and “Mobile 5” are within the same Bluetooth range.

- In total, 1 HTTP request was sent to the NCS (2,8MB) and 8 IO resolution requests (1KB each) were done to the NRS.

5.3.1.2 Results
Figure 54 shows the bandwidth consumption measurements perform in the NCS and NRS when running the test case.

![Figure 54 Bandwidth consumption in NCS and NRS when running the “Interaction with external components” test case](image)

5.3.1.3 Analysis
The successful execution of this test indicates that the prototype was able to interact with the
different external components used in this master thesis. Moreover, it allowed us to evaluate the prototype in terms of:

- Bandwidth consumption reduction
- Times involves in IO retrieval procedure

**Bandwidth consumption reduction**

Figure 54 shows the IO graph captured with Wireshark when the test case was performed. The scale used in the “Y” axis is logarithmic and represent the bytes/second (bandwidth). The “X” axis represents time measured in seconds.

In the graph, it can be seen two different color series, the blue one representing the NRS and the red one representing the NCS. Both series represent the bandwidth usage in the servers. In the NCS, it can be observed just one wide curve that represents the only content request that is sent to the NCS. Therefore, it is the only time in this test case where a representative amount of data is sent over the 3G up-link; the other file transmissions were done over Bluetooth instead. In the NRS series, it can be observed 8 peaks that represent the different GET and REGISTER requests that are sent to the NRS to either resolve the IOs or register the locators. As shown in the graph, these peaks are light-weight HTTP request (the heaviest of them was 6.886 Kbytes).

If we would have used the current content distribution, based on host-centric networks, to retrieve the same file from a web server, all mobile devices would have needed to use the 3G uplink in order to retrieve the content. We tested this scenarios and the result of this type of content distribution is shown in Figure 55.

![Figure 55 Content retrieval from a web server using HCN approach](image-url)
Figure 55 shows the IO graph captured with Wireshark when the host centric network scenario was performed. The scale used in the “Y” axis is logarithmic and represent the bytes/second (bandwidth). The “X” axis represents time measured in seconds.

In the graph, it can be seen 4 curves that represent the bandwidth usage in the web content server when the 4 content requests are sent from the mobile devices and respond from the server.

By comparing the results showed in Figure 54 and Figure 55, we can see a clearly benefit of using ICN NetInf architecture together with short range technologies for content distribution. The 3G UP-link bandwidth usage was reduced since the heavy data requests handled by the content server (NCS in the ICN case and Web Server in the HCN case) were reduced to just one for the ICN approach instead of the four needed for the HCN approach. This reduction in the bandwidth consumption can be translated into less network resources consumption and less processing power resources needed in the server side.

It is important to highlight that the results obtained in this scenario were obtained using four devices in the same Bluetooth range. Therefore, after the first device caches the object from the NCS, all the other devices were able to find at least one nearby peer to retrieve the content from instead of going to the NCS again. In the real scenario were users are scattered all over the event, it can be expected that when new content is available, most of the first requests to such content will end up being answered by the NCS. The more popular the content gets, the higher is the chance that the requestor will find a nearby peer from which it can retrieve the content instead of going to the NCS. As soon as peers start finding adjacent devices that have the desired content, the up-link usage reduction will start to be seen.

Deeper tests have to be done to simulate a more complete and complex scenario with much more devices participating in the content distribution and different strategies of content retrieval. Possible retrieval strategies that could be of interest would be:

- Non-sequential - where node requests may overlap in time
- Non-adjacent – where some nodes are not within the same range of the transport technology (e.g. not within the same Bluetooth range)
- Non-static nodes - where some nodes move around the space and become available from time to time

These simulations were out of the scope of this master thesis, however, within the Sail WPB group there is a discussion about possible simulation of more complex scenarios where the prototype developed by this project could be included.

**Times involves in IO retrieval procedure**

In Figure 54, two types of retrieval procedure are presented:

- Retrieval process from a nearby peer
- Retrieval process from a NCS

These two types of procedures share the same stages: getting the locators from the NRS, obtain the file from target source and register new locators in the NRS.

For both retrieval processes, the time involves for getting and register locators is similar,
between 1 and 2 seconds for each operation. However, the time for obtaining the file from the target source varies between the two processes.

For the retrieval process from a nearby peer, the total times for obtaining the files are shown in Table 15:

<table>
<thead>
<tr>
<th>Device</th>
<th>Total time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile 2</td>
<td>23</td>
</tr>
<tr>
<td>Mobile 3</td>
<td>25</td>
</tr>
<tr>
<td>Mobile 4</td>
<td>24</td>
</tr>
</tbody>
</table>

As seen in Table 15, the average time for obtaining a file of 2.8 Mbytes using Bluetooth was of 24 seconds. The times involves in this procedure are: peer discovery time, connection time between peers, file transfer time and delay introduce by the application. The time taken in each of these stages is:

- The average measured discovery time was of 7.13 seconds
- The average measured connection time was of 1.2 seconds
- The average measured transfer file time was of 11.928 seconds
- The average measured delayed introduced by the application was approx. 4.042 seconds. This delayed can be attributed to: database accesses, file storage delay, request processing, etc.

The NCS component was used for this project as bootstrap component, which initiates the content distribution for the test case defined in Figure 53. In Figure 54 we can see that the time for retrieving the file from the NCS was 45 seconds. This time was count from the moment when the HTTP Get request was sent to the NCS (approx. in sec 9, see Figure 54), until the data was transferred completely (approx. in sec 54, see Figure 54).

When analyzing these times intervals we noticed a strange behavior when retrieving files from the NCS. In Figure 54 we can see that as soon as the mobile device gets the locators from the NRS, the HTTP get content request towards the NCS is sent immediately after. However, when the NCS receive the get request, the data transfer started approximately 8 seconds after. We have confirmed this behavior repeating the tests and verifying that there is a delay inserted by the NCS when processing the requests.

In Figure 56 and Figure 57 there are shown two small traces highlighting the time when the NRS response with the locators, the time when the HTTP get request is sent to the NCS, and finally the time when the NCS starts sending back the data. This complementary information is showed to demonstrate that the delay is inserted by the NCS server and not by the prototype (developed within this project) running in the mobile devices.
In Figure 56 we can see that the NRS response back the HTTP get request with the locators in the second 46.62 (approx.). Afterward, we can see in Figure 57 that the mobile device sent the HTTP get request towards the NCS in the second 47.05 (approx.). However, it is until second 55.49 (approx.) that the NCS server starts transmitting the data towards the mobile device.

The delay inserted by the NCS is an undesired behavior that makes slower the content downloads that are done from this external entity. Since our main focus is the content distribution over the short range transport technologies integrated in the prototype developed, a detailed time analysis of this external component was not considered (out of the scope of this master thesis). The NCS component was developed by Nokia Siemens Networks (Sail NetInf partner) and it was used by this project as bootstrap component only.
6. Results

In this section, it will be presented and discussed the general results achieved throughout the development of this master thesis project.

The prototype developed within this master thesis project could be used in Events with Large Crowds scenarios as an alternative solution for optimizing content distribution. Our solution implements/ports a new control layer, based on NetInf ICN approach, for mobile devices. This control layer offers the users the possibility to resolve and retrieve information objects using p2p-like communication over different underlying technologies like Bluetooth and Wi-Fi Direct. When users retrieve information objects, these objects are cached in their mobile devices creating new sources of the content within the crowd. Other users trying to retrieve the same information object can make use of a nearby device copy instead of fetching the content again over the 3G uplink. Using this new content distribution scheme, the more popular the content gets the more chances it could be retrieved from a cached copy within the crowd and, this could lead to a less amount of data transferred over the 3G uplink.

Below it is summarized the different thesis’s contributions.

6.1 OpenNetinf in Mobile Devices

It was successfully ported a NetInf ICN approach implementation, OpenNetinf, to mobile devices (Android OS for our particular prototype). With OpenNetinf ported in mobile devices, users can take advantage of a new control layer for content distribution based on ICN architecture. By using this new control layer, users can retrieve the desired content from available local copies (if any) and, therefore, using less resources of the 3G uplink.

6.2 Architecture overview

The overview architecture that was proposed within this master thesis project can be found in section 4.1. The architecture is comprised by external and internal components. The internal components were developed by this project based on the OpenNetInf architecture. Depending on their functionality, each of these components belongs to one of the following layers: application layer, control layer and internal cache. The external components used in the architecture are: name resolution server (NRS), NI content server (NCS), QR code server and web server.

6.2.1 Internal components

The components that belong to the application layer are responsible for the user interaction, interaction with the control layer, discovery and connection establishment and finally the interaction with the QR code Server. All the components that belong to this layer are a contribution of this project:

- For the user interaction functionality, an Android application was developed in order to expose to the user the different functionality offered by the control layer. The .apk (Android application) can be downloaded from the following url: http://dl.dropbox.com/u/40610185/AndroidNetInf.apk. The object published when using the application can be found in the following web server URL=http://192.36.165.136:82.
- For the interaction of the control layer, we implemented the possibility of sending HTTP request (GET/POST) to a local HTTP server.

- For the discovery and connection process, the connection manager was created. It is important to highlight that within the OpenNetInf architecture, there was not any available component for this purpose. Therefore, the design decision and strategies used were a complete contribution of this project.

- Finally, the interaction with the QR server, done over HTTP, was also implemented by this project.

The components that belong to the control layer are responsible for: listen for request from the application layer, IO related operations, transfer BOs and listen for transferring requests. For this layer, some basic components of the OpenNetInf architecture were ported to the prototype: node, resolution controller, transfer dispatcher and locator selector. In addition, a set of resolution services, providers and BO servers were implemented by this master thesis in order to suit with the different requirements of the project. The components implemented by this master thesis were:

- A local resolution services capable of store and retrieve IOs in the SQL database provide by the Android system.

- A remote resolution service capable of interact with the Name resolution server (NRS) implementation of Nokia Siemens Network (NSN)

- Two new providers: the Bluetooth provider and the TCP provider. These providers allow the transfer dispatcher to create sockets connections over Bluetooth or Wi-Fi Direct.

- Two new BO servers: the Bluetooth and the TCP BO server. These servers allow the application to listen from other peers request to transfer BOs.

Finally, the components that belong to the internal cache layer were responsible for provide both structured data storage and file storage. For this layer, the only component implemented within this project was the SQL table used to store the IOs.

6.2.2 External components

The Name Resolution Server (NRS) is used as a remote resolution service to resolve IOs. Android nodes can register and retrieve IOs and the list of locators associated to them. The NRS was developed by Nokia Siemens Network and adapted to our prototype. The integration of the NRS shows interoperability between different components that have been developed by the Sail partners working in NetInf.
The NCS was also developed by Nokia Siemens Network and used by our prototype. This external entity provides an additional functionality for cache content in an external server through 3G. The main reason for adding this component was to allow us to bootstrap the content distribution process.

Finally, within this master thesis was developed an external QRcode server that interact with the Android NetInf node to generate QR barcodes. The content generated by this server is then showed in a project webpage. This functionality is useful for the nodes to read the hash value or identifiers when a node wants to perform a search request. In conjunction with this server, it was used a web server in order to be able to show the content that is being created (it shows QR codes, hash values, and dynamically add and delete MAC address from nodes exchanging content). The webserver used was windows IIS7. The webpage was developed using HTML and JavaScript.

6.3 Short range transport technologies
Bluetooth and Wi-Fi Direct were the two short transport technologies that were integrated to the Android NetInf ICN prototype. Bluetooth was the first short range transport technology integrated and its behaviors is stable (i.e. no error in the connections nor the in file transfers). Wi-Fi Direct was the second technology integrated and there are still cases when the connection process fails. Wi-Fi Direct is an immature technology that was released with the last android version platform 4.0 ICS.

It is important to highlight that within the OpenNetInf project there was not any support for Bluetooth nor Wi-Fi Direct. Therefore, all the functionality required to allow the peers to communicate over this technologies is a contribution of this project.

6.4 Prototype Evaluation
The performance of the prototype developed within this project was analyzed in terms of:

- Connection time between two peers over the different short range transport technologies.
- File transfer time between two peers over the short range transport technologies.
- Interaction with external components/servers (QRS, NCS and NRS). The interaction was analysed in terms of: bandwidth consumption measured in the NRS and NCS and the times involved in the IO retrieving procedure.

In section 5 we provided and discussed all the results obtained from the different tests cases defined to evaluate the prototype’s performance.
7. Conclusion

The main idea of this master thesis was to create an alternative solution for content distribution in large crowd events based on Netinf. This new alternative leverages the different characteristics of this ICN approach (e.g. in-network caching and unique naming) to create new copies of the content within the crowd. After this new local sources have been created, users are able to retrieve the content from the local copies instead of using the 3G uplink; this leads to a reduction of the bandwidth consumption.

To implement this new strategy, we have developed a prototype based on mobile devices and short range transport technologies (Bluetooth and Wi-Fi Direct). The solution presented implements a new control layer based on NetInf ICN approach in mobile devices. This control layer offers the user the possibility to resolve and retrieve information object using different underlying technologies like Bluetooth and Wi-Fi Direct.

In order to implement the NetInf ICN approach, the OpenNetInf platform was ported into mobile devices. Based on this platform, this project proposed an architecture comprised by internal and external components. This project made all the necessary implementation and changes in order to be able to run the different OpenNetInf components in the target mobile platform (Android). The IO transfer over the short range transport technologies was not supported in the OpenNetInf platform and it is a complete contribution of this project. Moreover, some of the external components used in the architecture were implemented by Nokia Siemens Network (NSN) and the interoperability with these components show the easy integration of our prototype with other NetInf projects developed within SAIL.

The prototype implemented was evaluated in terms of: connection time between peers, file transfer time between peers and interaction between the internal and external components. The results showed the different advantages and drawbacks of using our prototype.

The successful implementation of this prototype confirmed our hypothesis since we were able to port a NetInf implementation into the target mobile platform. Two short range transport technologies were successfully implemented for IO sharing between peers: Bluetooth and Wi-Fi Direct. By using these technologies to transfer the content instead of retrieving the files from the 3G uplink, our prototype is capable of reducing the bandwidth consumption over the uplink.

The prototype developed can be used as a base implementation for further development of the NetInf functionality in mobile devices. In addition, more complex experiments, simulations or evaluation could be performed to analyse the performance of the prototype when using: a considerable number of devices, different IO retrieving strategies, etc.
8. Future Work

8.1 OpenNetInf Platform
For this prototype were ported the basic components from OpenNetinf: node and transfer dispatcher. More OpenNetinf components could be integrated to the prototype e.g. Event Service that is a publish/subscribe system in charge to inform the subscribed nodes about the IOs that have been changed, created and deleted.

8.2 Wi-Fi Direct stability
As shown in the prototype evaluation, the current Wi-Fi Direct support in Android OS presented some problems when establishing connections between peers; this bug is expected to be solved in future versions of the Android OS. When this future releases become available, the connection establishment test case should be run again to determine if the problem disappears. These future implementations of Wi-Fi Direct may imply also son changes in the API functions that might require some changes in the source code as well.

8.3 Media files supported
The current implementation of the prototype can just exchange pictures. The application could be extended to allow support for other media files like videos, music, etc.

8.4 QR code server improvement
The current functionality implemented in the QR code and web server does not include the possibility of adding a description tag to objects being published in the web site. Therefore, unless the user knows exactly the hash of the object he/she wants to retrieve, it is not possible to differentiate between published objects. A possible way forward could be to add a human-readable tag to the object when publishing it on the website; users could make uses of these tags to identify the objects he/she wants to retrieve.

8.5 Control layer implementation
For the prototype implemented in this master thesis, the Android Netinf control layer is implemented as a part of the Android application. One of the drawbacks of this approach is that the control layer can be used only by the running application and it is not available to other applications. The control layer should be implemented as a service offered by the Android OS and should be available to every application on the device. Moreover, since the application is currently running the complete control layer, it consumes a lot of memory in run time and this behavior is not desirable.

8.6 Further areas of study
This project was focused in porting a NetInf implementation to mobile devices. During this implementation, we found some areas associate with the prototype were further studies are needed. Among these areas we find: privacy and security, battery consumption, chances of congestion of the wireless channel when a considerable amount of users are using the Wi-Fi and Bluetooth at the same time, etc. Studies on these areas are left as part of the future work for this project.
9. Bibliography


[17] María Alduán et al., "Architecture for Future Media Internet,". 


[22] 4WARD project EU. 4WARD project home page. [Online]. http://www.4ward-project.eu/ 


http://www.4ward-project.eu/index.php?s=Deliverables


## Appendix

### 10.1 List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAIL</td>
<td>Scalable and Adaptable Internet Solutions</td>
</tr>
<tr>
<td>ICN</td>
<td>Information Centric Networks</td>
</tr>
<tr>
<td>NetInf</td>
<td>Network of Information</td>
</tr>
<tr>
<td>CCN</td>
<td>Content Centric Networking</td>
</tr>
<tr>
<td>DONA</td>
<td>Data Oriented Network Architecture</td>
</tr>
<tr>
<td>BO</td>
<td>Bit Level Object</td>
</tr>
<tr>
<td>DO</td>
<td>Data Object</td>
</tr>
<tr>
<td>ES</td>
<td>Event Service</td>
</tr>
<tr>
<td>IdO</td>
<td>Identity Object</td>
</tr>
<tr>
<td>IO</td>
<td>Information Object</td>
</tr>
<tr>
<td>NRS</td>
<td>Name Resolution Server</td>
</tr>
<tr>
<td>NCS</td>
<td>Ni Content Server</td>
</tr>
<tr>
<td>QRS</td>
<td>QR Code Server</td>
</tr>
<tr>
<td>MDS</td>
<td>Mobile Data Storage</td>
</tr>
<tr>
<td>MDB</td>
<td>Mobile Data Base</td>
</tr>
<tr>
<td>RC</td>
<td>Resolution Controller</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
</tbody>
</table>
10.2 Demo

The prototype created for this project is an Android application that makes use of the NetInf control layer in order to exchange pictures between nearby peers using short-range technologies like Bluetooth and Wi-Fi Direct. Peers using the application are allowed to share content to the network by creating IOs that will represent BOs on the NetInf network. The demo also interacts with different components implemented in this master thesis: NRS, NCS and QR code server. The four main functions the application allows user to do are: share, get, delete and NCS publish.

Share

Using this function, users are able to share pictures from their devices that can be obtained from the phone’s camera or the build-in gallery. When pictures are shared, it will mean that an IO will be created and stored in a local database of the phone for that picture; this IO will be the BO’s unique representation in the NetInf network. A particular BO’s IO will contain a locator value where the file represented by this IO can be found. This locator value can be the MAC address of the Bluetooth and/or the Wi-Fi Direct adapter depending on the technologies available on the IO’s publisher device.

In order to make this IO also available to other users in the network, an HTTP request will be sent to the NRS in order to “PUBLISH” the IO. Published IOs are visible to other peers in the network and can be retrieved using the locators contained within it.
Get

Peers will be able to get IOs published by other peers using this application function. There are five main steps that are involved in the “Get” functions:

- **Finding the hash of the desired IO**: in NetInf networks, IOs are identified by hashes which are used as unique identifiers that allow in-network caching capabilities. For this prototype, the hash of the content is used as the IO identifier value. Therefore, whenever a peer wants to retrieve an IO the first step is to get the hash of it. This application uses QR-codes to allow a user-friendly way to obtain such hash; the ZXing barcode scanner [47] is used as QR-code scanner. Figure

- **Querying the NRS**: once the hash of the desired object has been obtained, an HTTP query request is sent to the NRS. If any other peer has already published the requested IO, the NRS will have an entry for it and will return the list of locators for the IO. If none entry is found for that IO, an error message will be sent back to the requester.

- **Getting the IO**: in the case that a list of locator is received from the NRS, the requestor peer will run a discovery process using its available transport technologies (Bluetooth and/or Wi-Fi) in order to create a list of all the surrounding peers. Afterwards, the list of discovered devices will be matched to the locators list retrieved from the NRS and all the addresses present in both lists will be selected as potential retrieving sources; these potential sources will be arranged using a certain priority (e.g. Wi-Fi over Bluetooth or vice versa).

- **BO retrieving**: using the potential retrieving sources list, the requester will attempt to create a Bluetooth or Wi-Fi connection to these different sources in order to try to retrieve the desired BO. If none of the sources is able to provide the BO an error will be sent to the application.

- **Caching IO**: if the BO is successfully retrieved from any of the available sources, an IO will be created for it and stored locally. In addition, the peer caching the content will send an HTTP request to the NRS in order to publish this new location of the object in the network.

When users use the Get function, they are allowed to pick which technology it will be used in order to retrieve the content:

1. Bluetooth
2. Wi-Fi Direct
3. NCS (using HTTP over the 3G link)

**Delete**

Using this function, users are able to delete pictures previously shared files using the share or publish to NCS functions. When a user deletes a picture, it will mean that the local IO created for that BO will be deleted from the database. In addition, in order to no longer make this IO available to other users in the network from this user, an HTTP request will be sent to the NRS in order to “UNPUBLISH” its locators (e.g. MAC addresses) for that particular IO. If other users have cached already the IO, the content will still be available to new users but not from the ones who have unpublished their locators.
NCS Publish

The NCS publish function is similar to the share function, the only difference will be that besides creating the local IO and registering the locators in the NRS, the BO will be cached in the NCS. Therefore, this content will now be available not only using the short range technologies but also through the NCS.

Figure 60 shows an example of the complete procedure of publishing and retrieving a file using the architecture proposes in this master thesis:

- Peer 2 uses the “SHARE” function in order to create an IO that will represent photo1 (a.k.a. the BO). A hashing algorithm (sha-256 was used in this paper) is used to hash the content. Using this hash, an IO is created and stored in the IO local database of the device. We will refer to this IO representing photo1 file as IO1.

- IO1 is “PUBLISH” to the Name Resolution Server (NRS) in order to make it available to other peers in the network. If the registration is satisfactory, the NRS will send a “REPLY(OK)” message.

- Peer 1 uses the “GET” function to scan a QR-code that contains photo1 file’s hash. This hash is used to create and send a “GET” request to the NRS for IO1.

- The NRS will REPLY with a list of possible locators for IO1. In this prototype it could contain the MAC address of Peer 2’s Bluetooth adapter and/ or the IP address of Peer 2’s Wi-Fi adapter. For this example let’s assume that the MAC address of the Bluetooth adapter is used as the locator value received from the NRS.

- Peer 1 will run a discovery process and it will find that Peer 2 is in its Bluetooth range. Since Peer 2 is in both addresses list (discovered devices and locators from the NRS), it
will be selected as a potential retrieving source for the BO represented by IO1. A Bluetooth connection will be established between Peer 1 and Peer2. Afterwards, Peer 1 will request the BO represented by IO1.

- Peer 2 will send to Peer 1 the BO (photo1) over the same Bluetooth connection. This file will be stored by Peer 1.

- If the BO is received successfully by Peer 1, and IO will be created for it and stored in Peer 1’s IO local database, i.e. Peer 1 will cache the IO.

- This new locator for the BO is “PUBLISHED” to the NRS.
### 10.3 Code Package Overview

<table>
<thead>
<tr>
<th>Package name</th>
<th>Overview</th>
<th>Layer</th>
<th>Associated architecture component</th>
</tr>
</thead>
<tbody>
<tr>
<td>netinf.android</td>
<td>Main package containing Android Activity which initializes the UI, starts the Control layer and listens for user’s actions. It also contains the DisplayResult activity which is used to show the BO requested by the user. Finally, this package also contains the NetInfBroadcastReceiver which is listening for messages sent from the Android OS related to Wi-Fi Direct and Bluetooth (peers discovered, connection status, etc.)</td>
<td>Application</td>
<td>User Interface</td>
</tr>
<tr>
<td>netinf.android.access.bluetooth</td>
<td>This package contains the BluetoothBOServer that will be used by the transfer dispatcher to listen for transferring request. It also contains the Bluetooth server that should be initialized in case that the Bluetooth remote resolution service will be used.</td>
<td>Control</td>
<td>Bluetooth BO server</td>
</tr>
<tr>
<td>netinf.android.access.rest</td>
<td>This package contains the classes used to initialize the HTTP server that will be used by the control layer to listen for request coming from the application layer. Redirection rules and parameter extraction configuration are defined in this package</td>
<td>Control</td>
<td>HTTP Server</td>
</tr>
<tr>
<td>netinf.android.access.rest.resources</td>
<td>This package contains the IO resource where all HTTP request are processed. It can process GET and POST request. According to the information in the request, the appropriate function of the node or the transfer dispatcher will be called from this class.</td>
<td>Control</td>
<td>HTTP Server</td>
</tr>
<tr>
<td>Package</td>
<td>Description</td>
<td>Type</td>
<td>Used in:</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>netinf.android.access.tcp</td>
<td>This package contains the TCPServer that will be used by the transfer dispatcher to listen for transferring request once a Wi-Fi Direct connection has been setup.</td>
<td>Control</td>
<td>TCP BO server</td>
</tr>
<tr>
<td>netinf.android.application.http</td>
<td>This package contains the two asynchronous tasks used by the activity to create HTTP GET and POST request. These classes also process the HTTP response from the server and, if necessary, call the appropriate component to continue with the process (e.g. BluetoothDiscoveryTask, DisplayResult, etc.)</td>
<td>Application</td>
<td>Application-Node Interface</td>
</tr>
<tr>
<td>netinf.android.bluetooth</td>
<td>This package contains an asynchronous task responsible for discover the nearby Bluetooth peers.</td>
<td>Application</td>
<td>Connection Manager</td>
</tr>
<tr>
<td>netinf.android.common.communication</td>
<td>This package contains the Bluetooth connections necessary for implementing the remote resolution between two mobile devices over Bluetooth (Not used in the final prototype)</td>
<td>Control</td>
<td>-</td>
</tr>
<tr>
<td>netinf.android.common.communication.protobuf</td>
<td>This package contains the .proto file used to create the Transferring Messages. It also contains the Transferring Messages java file create by the protobuf compiler</td>
<td>Control</td>
<td>Used in: Bluetooth and TCP providers</td>
</tr>
<tr>
<td>netinf.android.common.datamodel</td>
<td>This package contains the declaration of the different IO’s attributes. It also contains a declaration of the labels used in the IO’s identifiers</td>
<td>Control</td>
<td>-</td>
</tr>
<tr>
<td>netinf.android.common.security.hashing</td>
<td>This package contains the class the is used to create the Base 64 enconded hash</td>
<td>Application</td>
<td>User Interface</td>
</tr>
<tr>
<td>netinf.android.dialog</td>
<td>This package contains the different dialog classes used by the application (e.g. Alert Dialog, progress dialog, etc.)</td>
<td>Application</td>
<td>User Interface</td>
</tr>
<tr>
<td>Package</td>
<td>Description</td>
<td>Application</td>
<td>User Interface</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>netinf.android.input.qrscanning</td>
<td>This package contains the two classes used for QRcode processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>netinf.android.module</td>
<td>This package contains the Guice module used to configure the Control layer</td>
<td>Control</td>
<td></td>
</tr>
<tr>
<td>netinf.android.resolution</td>
<td>This package contains all the resolution services implementations (e.g. Local Resolution Service)</td>
<td>Control</td>
<td>Local Resolution Service, Remote Name resolution service and Bluetooth Remote resolution Service</td>
</tr>
<tr>
<td>netinf.android.resolution.local.database</td>
<td>This package contains the SQLite helper class used to interact with the database</td>
<td>Internal cache</td>
<td>MDB</td>
</tr>
<tr>
<td>netinf.android.transferdispatcher.</td>
<td>This package contains the implementation of the Transfer dispatcher and the Locator selector</td>
<td>Control</td>
<td>Transfer Dispatcher and Locator Selector</td>
</tr>
<tr>
<td>netinf.android.transferdispatcher.providers</td>
<td>This package contains the implementation of the different providers implemented (e.g. Bluetooth provider)</td>
<td>Control</td>
<td>Bluetooth provider, TCP provider and HTTP provider</td>
</tr>
<tr>
<td>netinf.android_wifi</td>
<td>This package contains an asynchronous task responsible for discovering the nearby peers and connects to the Wi-Fi direct group and the NetInfWifiDirectManager. The latter is used to manage all the calls to the WiFi Direct API (e.g. discovery, connect, etc.)</td>
<td>Application</td>
<td>Connection Manager</td>
</tr>
</tbody>
</table>
10.4 UML Diagrams

10.4.1 Bluetooth Discovery Task

```
BluetoothDiscoveryTask

getNRNRSolvedPeers()

while (counter < discoveryTimeOut)

getBluetoothDiscoveredDevices()

bluetoothDiscoveredPeers

matchingPeers, retainAllDiscoveredPeers()

deleteUnknownLocation(matchingPeers)

clearBluetoothDiscoveredDevices()

SendHttpPostRequest.execute()

NetInfoBroadcastReceiver

startDiscover()

BluetoothDevice.ACTION_FOUND

bluetoothDiscoveredPeers.add(device.getAddress())

BluetoothDevice.ACTION_FOUND

bluetoothDiscoveredPeers.add(device.getAddress())

AndroidSystem

```
10.4.2 WIFI Discovery and connection task

```
WIFIDiscoveryAndConnectTask | NET.WIFI.DiscoveryManager | WFPManager | WIFI secrecy Receiver | Android System
```

```
startDiscovery(handler)
while(counter < discoveryTimeout)
setDiscoverySuccess(true)
connectToDevice(allowedPeers)
while(counter < connectTimeout)
setConnectSuccess(true)
```

```
donPeersAvailable(peers)
connect(channel, config, ActionListener)
```

```
discoverPeers(channel, ActionListener)
requestPeers(channel, NetInfoManager)
```

```
WIFI_P2P_PEERS_CHANGED_ACTION
WIFI_P2P_CONNECTION_CHANGED_ACTION
```
10.4.3 GetIO
10.4.4 PutIO

- putListener
- startActivityForResult(Intent, "PICK_FROM_CAMERA")
- onActivityResult(int requestCode, resultCode, data)
- startActivityForResult(Intent, "PICK_FROM_FILE")
- onActivityResult(int requestCode, resultCode, data)
- createHTTPRequestFromByteArray(String, byte[], fileArray, String, filePath)
- encodeResult()
- String hash
- copyFileToMySharedFileFolder(String, String, hash)
- execute(new String[]{requestType, hash, filePath})
- resultDialog