AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY
Faculty of Electronics, Communications and Automation

Anssi Turkulainen

DELAY-TOLERANT NETWORKING FOR THE OPEN MOBILE SOFTWARE PLATFORM: AN APPLICATION CASE STUDY

Thesis submitted for examination for the degree of Master of Science in Technology

Espoo 25.1.2010

Thesis supervisor and instructor:

Prof. Jörg Ott

AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

ABSTRACT OF THE
MASTER'S THESIS

Author: Anssi Turkulainen

Title: Delay-Tolerant Networking for the Open Mobile Software Platform: An Application Case Study

Date: 25.1.2010 Language: English Number of pages: 8+85

Faculty: Faculty of Electronics, Communications and Automation

Professorship: Internetworking Code: S-38

Supervisor and instructor: Prof. Jörg Ott

This thesis is a study of leveraging a notion of delay-tolerant networking for challenged mobile network environments, where mobile devices face intermittent connectivity. In particular, we utilize in this study the mobile phones that run on Symbian, that is, the largest open mobile software platform in the world today.

We have implemented a novel bundle node implementation, called DTNS60, for the Symbian S60 smartphones. The design and the architecture of DTNS60 are described. The development of new delay-tolerant applications for DTNS60 has been enabled by introducing a common and extensible API for the bundle protocol service. Furthermore, we investigate the resource consumption and the throughput capabilities of DTNS60, and validate the interoperability between DTNS60 and the DTN2 reference implementation using real mobile phones. Also, a delay-tolerant voice application, called DT-Talkie, is studied as a case study.

In addition, critical Symbian-specific software development methods, typical mobile software development characteristics, and basic DTN theory required for the Symbian-based bundle node implementation are reviewed as a literature study.

DTNS60 is not a comprehensive implementation, instead it is more like a proof-of-concept. Therefore, also the evolution of DTNS60 is discussed. We suggest DTNS60 and related delay-tolerant applications to be merged with the Qt application and GUI framework enabling cross-platform support and extending the lifetime of the implementation also for the upcoming mobile phones.

It is feasible to implement the bundle protocol functionality for the stressed mobile networks. However, the proposed DTNS60 implementation requires further development to be more robust for the deployment into the real world.

Keywords: delay-tolerant networking, bundle protocol, DTNS60, DT-Talkie, API, Symbian, S60, interoperability, mobile software development, resource consumption, throughput, Qt

AALTO-YLIOPISTO
TEKNILLINEN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Anssi Turkulainen

Työn nimi: Viive- ja häiriötolerantit Verkot Avoimelle Mobiiliohjelmistoalustalle: Ohjelmistotutkimus

Päivämäärä: 25.1.2010 Kieli: Englanti Sivumäärä: 8+85

Tiedekunta: Elektroniikan, tietoliikenteen ja automaation tiedekunta

Professuuri: Tietoverkkotekniikka Koodi: S-38

Valvoja ja ohjaaja: Prof. Jörg Ott

Tämä diplomityö on tutkimus hyödyntää viive- ja häiriötoleranttien verkkojen käsitettä haastavissa mobiileissa verkkoympäristöissä, joissa mobiililaitteet kohtavaat epäsäännöllisiä yhteydenmuodostamismahdollisuuksia. Hyödynnämme erityisesti Symbian matkapuhelimia. Symbian on tällä hetkellä maailman suurin avoin mobiiliohjelmistoalusta.

Olemme kehittäneet natiivi-Symbianilla uudenlaisen viestisolmutoteutuksen nimeltään DTNS60. Ohjelmiston arkkitehtuuri ja suunnitteluperusteet esitellään. Yhteinen ja laajennettavissaoleva rajapinta viivetoleranteille ohjelmille mahdollistaa uusien ohjelmien kehittämisen viestisolmupalvelua vasten. Tutkimme DTNS60:n resurssien kulutusta, läpisyöttöä, sekä yhteentoimivuutta DTN2 referenssitoteuksen kanssa käyttäen mobiililaitteita. Lisäksi tutkimme käyttötapauksena viive- ja häiriötolerantin puheohjelman nimeltään DT-Talkie.

Diplomityössä esitellään myös kriittiset Symbian-spesifiset ohjelmistontuotantometodit, tyypilliset mobiiliohjelmistotuotannon ominaisuudet ja peruskäsitteet viive- ja häiriötoleranttien verkkojen teoriasta tämän projektin näkökulmasta.

DTNS60 ei ole täydellinen implementaatio vaan enemmänkin konseptin todistus. Siksi diplomityössä puhutaan myös DTNS60:n evoluutiosta. Ehdotamme, että DTNS60 ja siihen liittyvät viivetolerantit ohjelmistot siirtyisivät käyttämään käyttään Qt ohjelmisto- ja käyttöliittymäkehystä, joka mahdollistaa monialustatuen sekä pidentää implementaation käyttöikää myös tuleville mobiililaitteille.

Viestinvälitysprotokollan toteuttaminen on mahdollista haastaviin mobiileihin verkkoympäristöihin. DTNS60 implementaatio vaatii kuitenkin jatkokehitystä ollakseen tehokkaampi ja robustimpi käytäntöön siirryttäessä.

Avainsanat: DTN, viestisolmu, viestisolmuprotokolla, DTNS60, DT-Talkie, API, Symbian, yhteentoimivuus, mobiiliohjelmistotuotanto, resurssien kulutus, läpisyöttö, Qt

Preface

First, I want to thank Professor Jörg Ott for his kind guidance, discussions and offering professional insights during the process of the thesis creation. He provided an opportunity for me to work under this interesting research area in the Department of Communications and Networking (Comnet) for the research projects CHIANTI and REDI, and furthermore, managed the funding to attend related events supporting the thesis work and personal development. Thank you also all the other co-workers for offering such a nice working environment.

Then, I want to thank mobile software development and delay-tolerant networking communities for offering invaluable information for the thesis work. Last year, and the thesis-related work, has been an immense learning experience for me, and hopefully I can contribute back for these communities more in the future.

Especial gratitude goes to my family for supporting and encouraging me throughout my studies and life. I am grateful for my parents, Ritva and Markku, who made all this possible. Not forgetting my brothers, grandparents and other relatives, who have been there for me.

Last, but not least, all my friends deserve my warmest gratitude for offering me lifelong memories in the adventures we experienced during the recent years. Those were the salt of this study era.

Otaniemi, 25.1.2010

Anssi Turkulainen

Contents

Acronyms

AODV	Ad-Hoc On-Demand Distance Vector Routing
API	Application Programming Interface
AVKON	AVKON GUI framework
CPU	Central Processing Unit
DASM	Delay-Tolerant Applications for Symbian Mobile Phones
DLL	Dynamically Linked Library
DTN	Delay-Tolerant Networking
DTNRG	Delay-Tolerant Networkin Research Group
DTNS60	Delay-Tolerant Networking for S60 Mobile Devices
DTN2	DTN2 reference implementation
DT-Talkie	Delay-Tolerant Walkie-Talkie
EID	End-Point Identifier
GPS	Global Positioning System
GUI	Graphical User Interface
IPC	Inter-Process Communication
IrDa	Infrared Data Association
IRTF	Internet Research Task Force
MANET	Mobile Ad-hoc Network
MIME	Multipurpose Internet Mail Extensions
MP3	MPEG-1 Audio Layer 3
MVC	Model-View-Controller
NEP	Nokia Energy Profiler
OLSR	Optimized Link State Routing
OS	Operating System
P.I.P.S.	P.I.P.S is POSIX on Symbian
POSIX	Portable Operating System Interface for Unix
SDNV	Self-Delimited Numeric Value
SMS	Short Message Service
S60	Series 60 application platform
RFC	Request for Comments
TCP CL	TCP convergence layer
TCP CLA	TCP convergence layer adapter
UDP	Universal Datagram Protocol
WLAN	Wireless Local Area Network

1 Introduction

Mobility of systems, users, data and computing has been a world-wide success story. Today there are billions of mobile end-devices out there. It seems that people are carrying their mobile phones, and other mobile end-devices, everywhere they go. Mobility is ubiquitous.

Unfortunately, problems arise when mobile communication rely only on the traditional transport-level end-to-end connections offered by, for example, mobile ad-hoc networks, wireless local area networks or cellular networks. The coverage of the wireless networks offering connectivity for the mobile devices is not ubiquitous today nor in the near future. Technical, economical, legal and social factors restrict the ubiquitous connectivity to become available.

The traditional end-to-end connection in the mobile device fails in challenged network environment where disconnections of links and intermittent connectivity are typical characteristics. Especially, when the mobile device is changing from one wireless network to another, disconnections in connectivity are usual. The traditional application that relies on the traditional end-to-end communication is not usable anymore in this kind of challenged network environment. A novel approach is needed at the application level to cope with the intermittent connections.

For example, when a traveler is traveling on a train, her mobile phone may be connected to the wireless local area network (WLAN) at the departure railway station. But on contrary, when the train is moving between the stations in the middle of nowhere, there is no coverage for the WLANs. To maintain the seamless connection across the railway stations at the application level, delay-tolerant networking approach can be applied. Now, the traveler on the train is able, for example, to send emails with her mobile phone using a delay-tolerant email application while the train is moving, even though she is not connected to any underlying network. When the train arrives at the next station, the traveler achieves connectivity to the WLAN, and email is sent automatically towards the recipient using the delay-tolerant software in a mobile phone. Thus, a notion of delay-tolerant networking offers a solution for the intermittent and opportunistic connectivity problem in the mobile phone.

Background and related work

Delay-tolerant networking (DTN) is a well-studied area in the research communities around the world, for example, in the Delay-Tolerant Networking Research Group [14] in the Internet Research Task Force [28]. DTN was originally designed for the interplanetary internet [29], but can be applied also for the terrestrial wireless networks where challenged conditions for communication are evident - including those where mobile devices operate.

Real-world applications for mobile devices using delay-tolerant and opportunistic communication protocols have been introduced, for example, in Bytewalla[7], KioskNet[25], MobiClique[51], Drive-Thru Internet [48], Haggle[26] and CHIANTI[11] projects. Also, many other projects research the DTN concept in the different target environments, such as, DARPA[43], SAMI network connectivity [61] and N4C[73].

A notion of using DTN-based approach for mobile ad-hoc networking [42] on the Symbian platform [65] has been introduced using WLAN and Bluetooth as connectivity technologies. Furthermore, real-world experiments for the pocket switched networks (PSN) [50] has been already realized using these connectivity technologies on the Windows Mobile platform [77] to study the contact opportunities with other devices in the conference environment.

In this thesis, we take an implementation-based approach to study the delay-tolerant networking concept. The implementations of the DTN-based architecture [10] for the terrestrial mobile networks utilize, in this case, the experimental bundle protocol [62] to offer delay-tolerant message-delivery functions. Convergence layer adapters, e.g. TCP CLA [15] or UDP CLA [36], are used as an interface between the bundle layer, where the bundle protocol operates, and an underlying transport layer. The bundle protocol using the convergence layer forms the basis for the DTN functionality. However, there are many advanced functionalities that are introduced as well, such as, DTN security [70], DTN routing protocols [38, 2], and usage of metadata[69] that can be implemented as extensions for the bundle protocol. The Licklider Transmission Protocol [57], that can serve as a convergence layer adapter, is not considered in the thesis because of being more convenient for the interplanetary communication.

The development of the bundle protocol agent and the TCP convergence layer implementations for the Symbian mobile phones, called DASM[41], has been started earlier. We take DASM as an input to the thesis' implementation work, and rename

it to DTNS60 to be more descriptive, and embodying also every new component of the implementation that we have developed. The development of DTNS60 should conform a set of design principles for opportunistic communication in constrained computing environments [45].

The main implementation, called DTN2[18], of the delay-tolerant functionality to-date has been developed earlier as a reference implementation by the DTNRG using C/C++ as an implementation language. The reference implementation contains also a set of basic delay-tolerant applications, for example, for sending and receiving messages between bundle nodes. DTN2 is used in the interoperability testing part of the thesis. The target devices for the reference implementation are also other than mobile handheld devices. As of our knowledge, there is also implementation effort for the bundle protocol (at least) in the DTNRG to develop Java, Python and Objective-C solutions, that are targeted for the mobile devices. Furthermore, also a port to the Android platform [7] has been developed to-date. These other implementations can be used in conjunction with DTNS60 to offer cross-platform bundle protocol functionality for the mobile devices.

A delay-tolerant voice application, called DT-Talkie[31], for Maemo Internet tablets demonstrates the notion of asynchronous voice messaging [27] on the mobile devices. It is a push-to-talk[47] application that can function in the delay-tolerant networks. The Maemo version of DT-Talkie uses DTN2 for the bundle protocol services. In the case study part of the thesis, we use DT-Talkie as a reference to implement a Symbian-specific DT-Talkie implementation that uses the bundle protocol service offered by DTNS60.

Research problem

Traditionally, mobile devices utilize mobile networks with a infrastructure network support, or mobile ad-hoc networks, to establish communication sessions with other devices using underlying transport-layer in an end-to-end manner. Usually, TCP/IP is used as a communication protocol, and routing protocols such as AODV[49] or OLSR[12] are utilized. When a mobile device loses the connectivity, most applications requiring networking capabilities (e.g. web browsers and email applications) will not work anymore on the device.

Applying the DTN-based approach for these challenged mobile networks on the other hand relies heavily on asynchronous communication, where the transport-level end-

to-end connectivity is relaxed. The DTN-based approach offers application-level end-to-end connections using the bundle protocol to deliver messages encapsulated as variable-sized bundles in a hop-by-hop basis from the source to the destination. It operates in a store-carry-forward manner forming an overlay layer and abstracting away the underlying protocol stack, i.e., the bundle layer forms an asynchronous messaging platform. Consequently, an application running on the mobile network that cannot maintain transport-level end-to-end connectivity can be made functional when the bundle protocol has been taken in use in the communication protocol stack. Now, the application can communicate with the remote application, if there is at least one path in time from the source node to the destination node (in the hop-by-hop basis). For example: in the rural areas a person carries in a mobile device other person's messages to the remote town as a "data mule", and transmits the messages when the contact opportunity becomes available, i.e., when arriving to the town and meeting the other person.

Delay-tolerant applications on a mobile device must also be asynchronous by their nature. This means that existing applications, that commonly rely on the synchronous TCP/IP communication, need to be modified to support the bundle protocol layer in their protocol stack enabling asynchronous communication. Delays for delivering an application message, or a "bundle", may vary from a few milliseconds to many hours, or even days or longer in the deep space communication case (space is not our target, though).

Finally, implementing the bundle protocol service for the mobile devices introduces several challenges that are typical for mobile computing but not for fixed infrastructure computing. For example, persistence of bundles, processing power and energy consumption on the mobile devices may bring drawbacks due to limited memory sizes, low-speed CPUs and limited battery-life of the mobile phones.

Objectives and scope of the thesis

The main goals of the thesis are:

- to elicit the requirements and the practices for implementing the bundle node for the mobile software platform, focusing on the Symbian platform;
- to validate the Symbian-specific bundle protocol functionality in the laboratory environment using real mobile phones;

- to evaluate the resource consumption and the throughput of DTNS60, and to validate the interoperability between the DTNS60 and DTN2 implementations;
- to provide a common and extensible application programming interface for the delay-tolerant application developers;
- to analyze the cross-platform delay-tolerant voice application, DT-Talkie, using DTNS60 for the bundle protocol services in the Symbian plaform side;
- to enable further development of DTNS60 by re-architecturing, refactoring and developing further the legacy DASM implementation;
- to gain experience about porting an application from the Linux-based platform to the Symbian platform; and
- to release DTNS60 and DT-Talkie available for the open source community.

The goals are achieved by conducting first a literature study for the requirements of the bundle node implementation for the mobile devices, and for the mobile software development characteristics that are evident in the target Symbian platform. Then, we propose a native Symbian C++ bundle node implementation. We verify the implementation against the requirements specifications, and evaluate the implementation. Furthermore, an application case study, utilizing the bundle node implementation, is performed.

The target network type for the proposed DTNS60 implementation, introduced later in the thesis, is a terrestrial wireless network that cannot maintain end-to-end connectivity at the transport layer, i.e., those that are formed from mobile ad-hoc networks or mobile infrastructure-supported networks, when connectivity opportunities are available. Hence, our target is to investigate the bundle protocol functionality in one network (this can be extended in the future). On the other hand, these mobile networks can contain heterogeneous mobile end-devices (e.g. mobile phones, Internet tablets, laptops and desktop computers) connected to the wireless access points or communicating in an ad-hoc manner through wireless network interfaces. Communicating between different heterogeneous networks requires gateway implementations and is out of the scope of the thesis. Other networks where the DTN architecture may be expanded are Interplanetary Internet [29], sensor networks, satellite networks, and underwater acoustic networks.

Multi-platform support for the bundle protocol is an important goal since there is no overall benefit for developing several bundle protocol implementations, if they can

not communicate with each other in the real world. The interoperability between the DTN2 reference implementation and DTNS60 fulfills this requirement in the scope of the thesis. The bigger picture is to implement a world-wide DTN testbed, including mobile and fixed infrastructure, that is formed from many implementations using, for example, C/C++, Objective-C, Java and Symbian implementations, that can seamlessly communicate with each other. There are efforts towards this goal in the DTNRG.

We choose to develop the legacy Symbian implementation further instead of porting the DTN2 reference implementation to Symbian using Open C/C++ libraries [46]. As of our knowledge, the native environment provides a more efficient solution. Porting the reference implementation for S60 environment instead of developing S60-specific implementation requires further research and is out of the scope of the thesis.

Methodology

Even though there is a heavy tussle among the different mobile software platforms - and the winner remains to be seen - we choose to implement the bundle node on the Symbian mobile phones because of the wide availability of the devices, due to earlier effort with the Symbian-specific bundle protocol implementation, and keeping the scope of the thesis reasonable choosing only one platform. Furthermore, Symbian mobile phones offer many connectivity methods, such as, WLAN, Bluetooth and cellular network connections. Other advanced features, such as a GPS module, a high-quality camera and a touchscreen can also be leveraged by delay-tolerant application development for the Symbian devices in the future. Moreover, Symbian is still the largest open mobile software platform in the world today [65].

The proposed DTNS60 implementation has been achieved by developing DASM further to provide more advanced functionality. In addition, we port the DT-Talkie voice application for the Symbian platform. The porting process is introduced in the case study chapter as porting an application for Symbian is known to be a hard problem in the mobile software development community. We develop also Symbian-specific implementations for the basic tools to send and receive bundles, and to ping bundle nodes, that can be used as a basic tools in the interoperability testing and in the throughput analysis. Analyzing the resource usage of the implementation on the real devices has been achieved by using Nokia Energy Profiler [44].

Also, to continue the lifetime of the DTNS60 implementation, development of 3rd party applications using the bundle protocol service and the further development of the service itself must be made easy to grasp. This requires a careful consideration for the software design and the architecture. DTNS60 provides an extensible and common application programming interface for bundle protocol services that delay-tolerant applications can use, and that is encouraged to be developed further.

An addition of Bluetooth, and other network interface technologies, has been made feasible using a robust design in the software and component-based software engineering. Also, good practices in software development, such as, design patterns and test-driven development have been taken in use to lower the barrier for the further development, and in order to produce quality software.

The underlying network for the bundle protocol in our implementation is TCP/IP at the transport and the network layer, and IEEE 802.11 at the link layer. An option to choose between the WLAN ad-hoc mode and the infrastructure mode has been enabled using the native features of the Symbian phones.

An output of the thesis is a new alpha release version of the DTNS60 bundle node implementation including DT-Talkie as an example delay-tolerant application. The basic tools for sending and receiving bundles, and for pinging other bundle nodes, are also provided. Furthermore, DTNS60 is released truly available for the open source community in hope of attaining contribution, i.e., source code of the software has been made transparent and accessible.

Outline and the structure of the thesis

The following second chapter conducts a literature study describing an overview for the basic delay-tolerant networking characteristics, continued in the third chapter describing typical mobile software development characteristics, and critical Symbian programming paradigms for this project. In the fourth chapter, the design, the architecture and the functionality description of the different components of DTNS60 are introduced. In the fifth chapter, the implementation is validated and evaluated, and the future development ideas are discussed. In the sixth chapter, an application case study using a novel DT-Talkie for Symbian is introduced and real world deployment scenarios are provided. Finally, we look the lessons learnt and make conclusions about the work done. In the end, the future research ideas are discussed.

2 Delay-Tolerant Networking

This chapter is a review for the basic delay-tolerant networking concepts and requirements. First, differences between the traditional Internet-based approach and the DTN-based approach are revisited in the form of motivation for using DTN. Then, the fundamental parts of DTN, that are, the architecture, the bundle protocol, convergence layers, extensions (such as security), and routing protocols are reviewed. The bundle protocol agent using the TCP convergence layer adapter forms the basis for our implementation (keeping in mind that the not-implemented features can be added to the solution in the future). Lastly, examples for challenged terrestrial mobile networks are described. The mobile devices can use the DTN-based approach for communication in these kind of stressed mobile networks.

2.1 Motivation

One important design principle of the traditional Internet architecture is an end-to-end principle [60]. TCP is the most common transport protocol, and uses Internet Protocol at the network layer to interconnect heterogeneous networks together. Hence, reliability has been traditionally implemented at the transport layer using synchronous TCP/IP communication. The connectivity offered by the traditional TCP/IP-based Internet is usually functioning without problems, if there is a robust underlying network infrastructure providing connectivity services and static end-devices, for example, servers that have wired and reliable connectivity methods to the network backbones.

When the mobility of the end-devices is evident, as it is in the current modern societies around the world, another kind of approach is needed to cope with disconnections and disruptions of the mobile communication links caused by the traditional communication paradigm. Moreover, in the developing countries, it is a commonplace that people get the first touch to the Internet via mobile devices, and the fixed network infrastructure is lacking, i.e., mobiles devices are used at the edges of the Internet. As mentioned before in the introduction in the example scenario, connectivity opportunities for the mobile devices are intermittent depending on the coverage of the access points (offering usually also a gateway to the Internet). The traditional architecture fails in these kinds of challenged (and mobile) network environments. We can see, that mobility of the devices that communicate with each other, is the new characteristic in networking. Therefore, it is reasonable to evolve

the traditional communication architecture to support delay-tolerant and mobile characteristics as the usage habits of the architecture itself are evolving from fixed to mobile computing.

The DTN-based approach hides the leaking abstractions, i.e. traditional connection abstractions, from the users of the mobile devices by introducing an overlay network on top of the existing heterogeneous network infrastructure. The overlay approach enables a seamless transformation from the delay-intolerant architecture into the delay-tolerant architecture since both can be used in parallel (no need to change underlying protocol stacks). The overlay network uses the bundle protocol for passing messages as bundles from one node to the another. The bundle protocol forms a store-carry-and-forward overlay network on top of transport layer and it can form conceptual end-to-end connections between delay-tolerant applications, if the message-delivery possibilities from the sender to the receiver are available at some point of time (which may not always be the case, though). DTN leverages asynchronous communication, so the delay-tolerant applications must be also asynchronous by their nature.

The DTN-based approach can be applied to the stressed mobile ad-hoc networks and to the stressed infrastructure-supported mobile networks. DTN-based mobile ad-hoc networking [42] can be applied, for example, in disaster recovery areas, mining work sites, rural areas among indigenous people, or military battlegrounds. Common characteristics for these application areas are that the fixed network infrastructure is lacking, and there are delays, disruptions or re- and disconnections involved in the communication links. End-to-end connectivity may never exist, instead hop-by-hop connections are evident.

In the disaster recovery areas, for example when an earthquake has happened, there is an urgent need to get rapidly a communication infrastructure operational to enable the communication of the aid management. Bringing mobile devices to the area, and building a mobile and delay-tolerant ad-hoc network between the devices is faster than building a new fixed infrastructure from scratch.

In the mining sites, it is cheaper to deliver messages down in the hole carrying the messages in the mobile devices than building an infrastructure network to cover the whole mine. The same kind of situation is in the rural areas where it is not economically rational to build an infrastructure network. For example, in the rural areas helicopters can bring messages (such as news and emails) for the indigenous people when delivering food and other supplies for them. Also, local and virtual

communities can be formed using the asynchronous messaging paradigm to deliver messages among the local people.

In the military battlegrounds, it is common that the hostiles are using electronic warfare: to jam the devices of the other party, and to disrupt the communication links of the enemies to gain an advantage in the battle. Also, the loss of end stations is usual. Moreover, the military operates in challenged environments, where they may be outside of the coverage of the fixed communication infrastructure that would provide, for example, telephony and data-delivery services. Furthermore, mobile and delay-tolerant ad-hoc networking may enable communication in a more convenient way (than using, e.g., satellite phones) also among platoons and squads that are on a mission.

2.2 Architecture

The delay-tolerant networking architecture is described in RFC 4848[10]. The architecture supports intermittent connectivity, long or variable delay, and asymmetric data rates. The purpose of the DTN architecture is to loosen the traditional TCP-based assumptions since the end-to-end path at the transport layer may not exist for the whole duration of the communication session. Also, because the architecture supports interoperability between heterogeneous networks [21], all underlying networks do not need to support TCP/IP protocols on the communication path. This is achieved by the overlay nature of the delay-tolerant network. The overlay network layer is called a bundle layer, that is used to deliver variable-length messages between a sender and a receiver, and it is located at the application layer in the traditional Internet model [64]. The bundle layer is depicted in Figure 1.

The sender nodes and the receiver nodes of the bundles, bundle nodes, are identified by a DTN-specific naming syntax, endpoint identifiers (EID), that enhances interoperability between the heterogeneous networks. The late binding characteristic of the architecture enables that the binding of the identifiers does not necessarily happen at the source bundle node since the destination EID may be re-interpreted at each hop.

Also, the receiver bundle nodes are assumed to have some kind of persistent storage to support store-carry-and-forward functionality over multiple paths. In other words, the storage of the DTN network is distributed to the bundle nodes. The architecture addresses also security mechanism, and coarse-grained classes of service. An analogy

for the DTN network is the postal service. DTN forms an asynchronous messaging platform for the message-delivery operations.

The bundle layer of the architecture also affects the delay-tolerant application design. The delay-tolerant applications should be designed keeping in mind to minimize round-trip times, and prepare the delay-tolerant application with the ability to cope with restarts after the failure while the network transactions remain in the pending state. Also, the delay-tolerant applications should inform the network of the useful lifetime and the relative importance of the data in the bundles to be delivered.

2.3 Bundle protocol

The bundle protocol is described in detail in RFC5050[62] (including the message formats and the processing algorithms). The bundle protocol operates at the bundle layer of the DTN architecture described in the previous section. Conceptually, the bundle protocol builds an overlay network on top of the convergence layer adapters that work on top of the transport layer in the Internet model, i.e., the bundle protocol is located at the application layer.

The main goal of the proposed DTNS60 implementation is to implement a low-level daemon service for the bundle protocol in the mobile phone. This service is able to send, receive and forward bundle messages between bundle nodes. A bundle node is an entity in the delay-tolerant network that has these capabilities. The conceptual components of the bundle node are a bundle protocol agent, convergence layers (described in the next section), and application agents, i.e., components that communicate with the delay-tolerant applications, and that utilize the bundle protocol services. DTNS60 is a proposed implementation for the bundle node for Symbian mobile phones containing these conceptual components.

The bundle protocol agent is a node component that offers bundle protocol services for delay-tolerant applications. It executes the procedures of the bundle protocol, that are, bundle processing phases (e.g., bundle parsing and encapsulation) and administrative record processing. The bundle protocol agent is tied (offers an interface) to the routing engine including algorithms deciding where the bundles are forwarded next. Also, there is an assumption in the bundle protocol that forwarding information bases of the bundle nodes are already populated, e.g., manually in the simplest case.

An application agent has an administrative part and an application-specific ele-

ments. In the proposed DTNS60 implementation, we have developed a few example applications that communicate with the conceptual component that is considered to be an application agent, and is located in the same application process as the the bundle protocol service.

Key capabilities of the bundle protocol are a custody-based retransmission, an ability to cope intermittent connectivity and an ability to take advantage of the scheduled, predicted and opportunistic connectivity. In addition, the bundle protocol has the ability to take an advantage of the continuous connectivity. The custody-based retransmission means that some other bundle node than the original one has taken the responsibility to deliver a bundle to some end-point, i.e., the bundle is delivered in the hop-by-hop basis from the sender to the receiver. The late binding of the overlay network endpoint identifiers to the underlying internet addresses is also a key feature of the bundle protocol supporting the DTN architecture. The late binding means that the end-point identifier of the recipient is not bound to the underlying network address at the source bundle node, because the transmit time may exceed the validity time of the binding; instead the binding is re-interpreted at each hop.

There are several ways to implement the bundle protocol architecture. Our implementation is realized as a daemon process on the mobile phone, that the delay-tolerant applications consume using inter-process communication mechanisms. Other implementation models may be, for example, to implement the bundle protocol functionality as a peer application node, as a sensor network node, or as a dedicated router.

The bundle protocol has been designed to keep in mind security considerations from the beginning. For example, only authorized bundle nodes should be able to send bundles to the delay-tolerant network where resources of the mobile bundle nodes (storage, bandwidth etc.) are scarce.

A message passing through a delay-tolerant network, that uses the bundle protocol, is depicted in Figure 1. The underlying layers beneath the bundle layer and convergence layer adapters may be TCP/IP based, but can also use other protocols. The proposed DTNS60 implementation is also depicted in Figure 1.

The basic message blocks for a single bundle are formed from the primary and payload blocks, but the bundle protocol message formats has been designed to keep in mind extensibility and scalability. Therefore, the messages are encoded using self-delimiting numeric values (SDNV)[19]. SDNVs are encoded using 7 least significant bits and setting the last octet's most significant bit to zero. Every other most

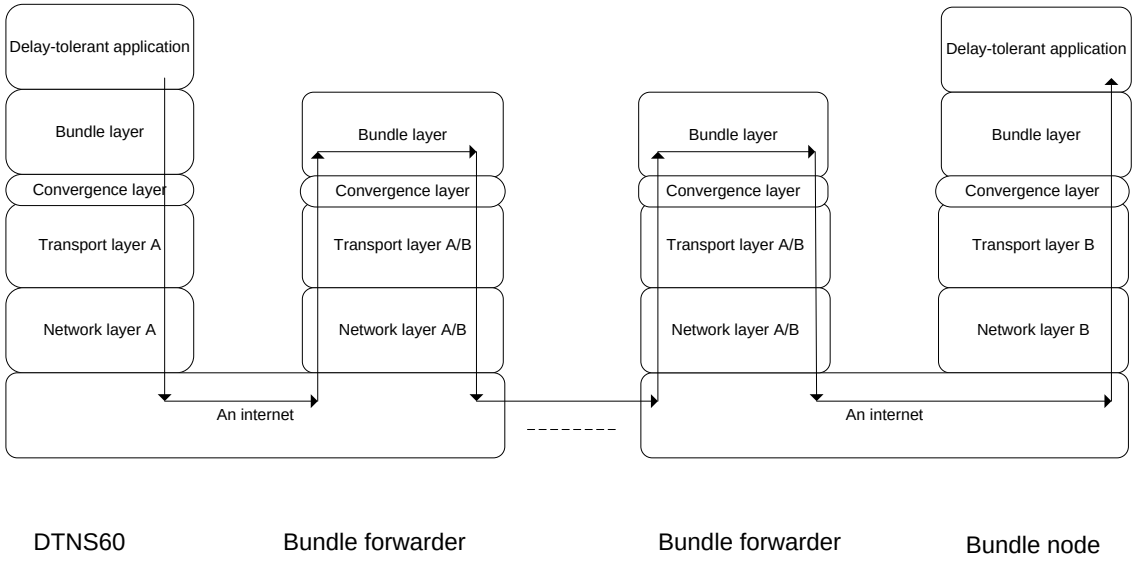

Figure 1: Bundle protocol

significant bit in the subsequent octets is set to one. The usage of SDNVs enables an addition of the extensions to the bundle protocol that may not be known yet.

The bundle protocol requires services from the convergence layer adapters. The convergence layers must provide means for sending a bundle to all bundle nodes in a minimum reception group that are reachable via a particular convergence layer adapter. Also, delivering a bundle, that was sent by a remote bundle node via the convergence layer protocol to the bundle protocol agent of the current node, is provided by the convergence layer adapter.

2.3.1 Extensions

The bundle protocol has been designed so that it can be extended using extension blocks that are all blocks other than primary and payload blocks in a bundle message. If the bundle node is not able to process a particular extension block, it flags the bundle to be unprocessed, and forwards it towards next recipient (that may have the extension block specific capabilities). The extension blocks are not implemented in DTNS60, but those may be features to implement in the future to provide more advanced bundle protocol functionalities in a mobile phone. We introduce briefly below four extension blocks that have been introduced today. An up-to-date list of the available extension blocks is available from the DTNRG.

Security

The bundle protocol can be deployed to the environment where the typical security challenges concerning availability, integrity and confidentiality, are apparent. The bundle security protocol [70] offers data confidentiality and integrity options in form of optional extensions blocks that can be added for each bundle. The security protocol applies to the bundle payload block and other various extensions blocks that may be used in the single bundle enabling the bundle-based communication to be secure.

Metadata

The metadata extension block [69] is intended to provide information for the bundle nodes as they process bundles. The metadata is related to the application-level information, but is used in the bundle processing algorithms in the bundle protocol agents. For example, the metadata block may contain information whether the bundle should be forwarded to the next node, or stored in the persistent memory for future processing.

Previous hop

The previous hop insertion block [67] is designed to include an EID of the previous bundle node, that was added by that previous node when forwarding the bundle to the current node. The intention of this extension block is to prevent routing loops.

Retransmission

The retransmission extension block [68] is designed to prevent duplicate bundles in a delay-tolerant network. It uses some particular policy to delete all bundles that are already forwarded caused by intentional or unintentional routing loops. The custodian, i.e., the bundle node that currently holds a copy of the bundle to be delivered and is responsible for (re-)transmitting the bundle, inserts a retransmission block to the bundle when it is retransmitting that bundle.

2.4 Convergence Layer Adapters

A convergence layer adapter is an interface between the bundle protocol and the underlying network protocol stack. Our proposed implementation uses the TCP convergence layer adapter, and we look in detail, how it works. Also, UDP CLA [36] and other convergence layers can be added later to the implementation, but is out of the scope of the thesis. For every underlying protocol stack, own convergence layer

adapter implementation is required, for example, UDP and TCP requires separate implementations. Convergence layer adapters lay also at the application layer in the Internet model and they usually consume services offered by an underlying operating system (e.g. Sockets API in UNIX and Symbian).

Transmission Control Protocol Convergence Layer Adapter

The TCP convergence layer protocol [15] (TCP CL) utilizes widely-used Transmission Control Protocol as an underlying transport layer protocol. The main functionality of the TCP CL is introduced briefly here because we use it in DTNS60.

The main functions of the TCP CL are: to specify the encapsulation and decapsulation of the bundles, and to describe the procedures for TCP connection setup and teardown. Typically, a node uses the services from the underlying operating system to establish a TCP connection, which is required for the TCP CL connection. A TCP CL connection is formed initially by sending contact headers between the sender and the receiver. After the TCP CL connection has been setup between two nodes, bundles are transmitted in one or more data segments including a header in each data segment that tells the length of the bundle data in the variable-sized SDNV format. The receiver acknowledges all the data segments and allows a possibility to the reactive fragmentation. When the transmission of one or more bundles has been ended the connection is torn down. The typical flow of the TCP CL protocol messages are depicted in Figure 2.

A more detailed description of the TCP CL protocol is described in the specification. For example, message formats, errors and rejections are handled in a TCP CL - specific way.

2.5 Routing protocols

In the DTNRG, there has been designed many routing protocols that can be taken in use in conjunction with the bundle protocol. The intention of the DTN routing protocol is to choose best route from the sender to the receiver in challenged network environments. In the current version of our implementation, that is introduced later in the thesis, we use only static routes. But for the future research, implementing and validating the DTN routing protocols and integrating those to our implementation is an interesting area to look for. The simulation studies for the routing protocols [2] suggest that PropHet[38] is the most appropriate routing protocol to implement first for DTNS60. Other routing protocols are, for example, flooding, epidemic,

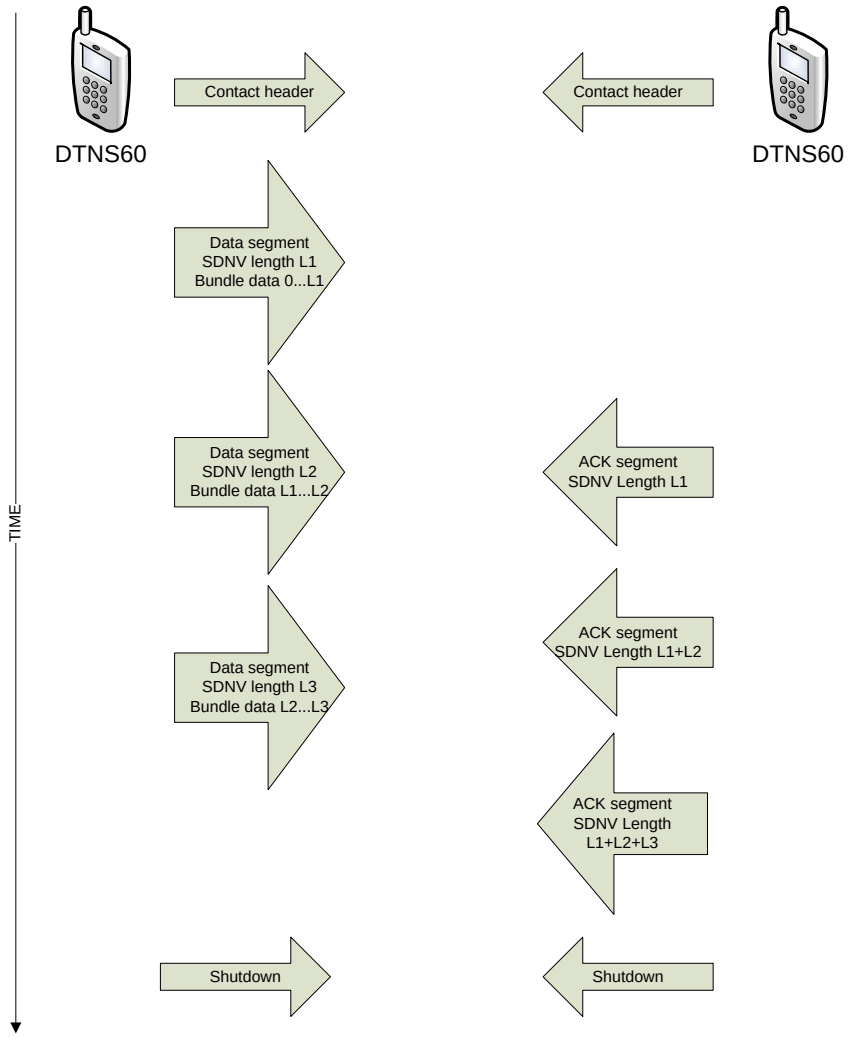

Figure 2: Typical flow of TCPCL protocol [15]

probabilistic, maxprop and spray-and-wait routing protocols.

2.6 Challenged terrestrial mobile networks

An objective for the implementation work of the thesis is to deploy the bundle protocol to the challenged terrestrial mobile networks. These networks are terrestrial wireless networks that cannot ordinarily maintain end-to-end (transport-level) connectivity. There are two ways to form connectivity, when available, in these kind of networks; to use the infrastructure network support by means of using an access point (e.g. WLAN infrastructure mode), or to form a mobile ad-hoc network

(MANET), where mobile devices form a network in a peer-to-peer manner in the infrastructure-less environment.

2.6.1 Infrastructure-supported

In the infrastructure-supported mobile networks, mobile devices achieve connectivity to the network via access point (e.g. via wlan access point or 3G tower). All traffic in and out from the device, encapsulated by the bundle protocol, flows through the access point that is currently in use. WLAN in the infrastructure mode is the infrastructure-supported mode in our implementation. Figure 3 represents an example of the challenged mobile infrastructure-supported network.

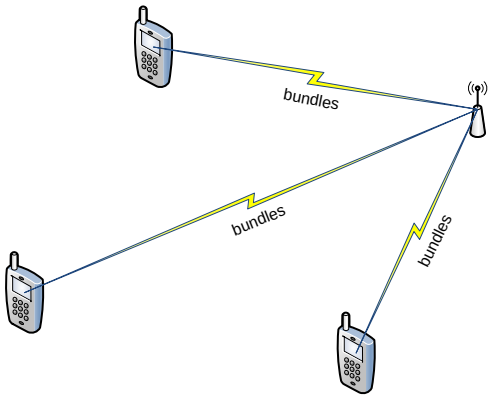

Figure 3: Challenged infrastructure-supported mobile network

2.6.2 Infrastructure-less

Conventional MANETs form an arbitrary networks structures without any help of the infrastructure networks. The connectivity is assumed to be end-to-end and always-connected for the lifetime of the session (like in the traditional Internet). Compared to the conventional MANETs, challenged MANETs may achieve communication opportunities from sporadic and intermittent contacts. In the challenged MANETs, disconnections and re-connections occur frequently and the end-to-end principle is relaxed using the bundle protocol, like in the infrastructure-supported case. WLAN in the ad-hoc mode is the infrastructure-less mode that our implementation supports. Figure 4 represents an example of the challenged mobile ad-hoc network.

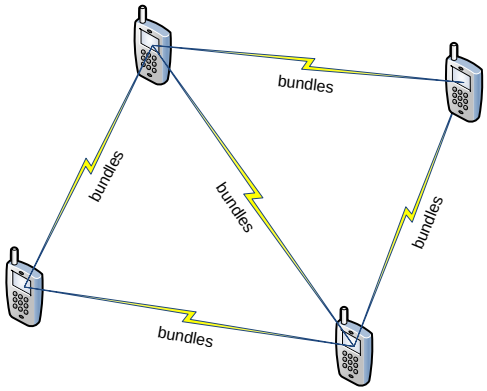

Figure 4: Challenged ad-hoc mobile network

2.7 Summary

In this chapter we reviewed the basic DTN concepts and the differences between the DTN and the traditional Internet concepts. The chapter forms an overview for the requirements of the bundle node implementation, even though more detailed requirements are found from the requirements specifications. The next chapter takes a look in the characteristics of mobile computing and in the general mobile software development practices used in the Symbian-based project, that are needed to take in account when implementing a bundle node for the Symbian device. In the subsequent chapters we propose a bundle node implementation for the Symbian S60 mobile phones, and investigate it in more detail.

3 Mobile software development

In this chapter, the characteristics of the mobile software development from our implementation point of view are studied. Therefore, our focus is on the Symbian software platform, but most of the characteristics can be applied also for the other popular mobile software platforms, that we briefly review in this chapter. Software development for Symbian, and especially for the S60 application platform, has some typical characteristics and capabilities in addition to the generally good software development practices (such as, commenting the code and the coding conventions). The client-server framework, the active object framework, the Avkon GUI framework, the ESOCK communications framework and the multimedia framework are introduced since those are the main frameworks for implementing DTNS60. Nokia Energy Profiler (NEP) is reviewed because we use it for measuring the resource consumption of DTNS60. We describe, how NEP can offer more deeper integration to DTNS60 in the future. In the end of the chapter, test-driven development, beneficial design patterns, component-based software engineering and event-driven programming are briefly discussed in order to create quality software, and because those are the essential software development paradigms for the proposed implementation introduced in the following chapter.

3.1 Constraints on software

Mobile software development generally, and especially for the mobile devices apparent in the challenged mobile networks, are restricted by a set of constraints that are distinct compared to the desktop and enterprise environment. We describe here the typical constraints [33] in more detail keeping in mind that we are developing delay-tolerant software for the Symbian-based devices. However, most of these constraints are universal for the mobile software development on every mobile software platform.

Most importantly, mobile devices have (today) constrained hardware due to the nature of the mobile device; the device needs to be small enough that it can be carried always in a pocket. Also, economical reasons prevent devices from being more powerful. More efficient hardware is more expensive. When a mobile manufacturer ships millions of devices each month, even a small cost savings in hardware costs add up to a large sum eventually.

The power supply of the mobile device is dependent on the battery, that is a scarce

resource. This is not the case in the enterprise servers and desktop computers using a fixed electrical charge. The power usage of the device must be minimized so that the phone can stay active as long as possible with one battery charge. The battery life is typically the most valuable requirement for the end users, and especially valuable for the challenged environments where charging possibilities may not be available all the time. The goal is to minimize the activity time of the mobile phone's hardware and software modules, for example, activity of the screen, antennas, or daemon processes (e.g. the bundle protocol service) running on the device. We look at the energy usage of DTNS60 in the chapter 5.

The users of the mobile device expect high reliability of the software running on the device. One reason for this is the capability of the mobile phone to act as a life-saving tool, for example, by placing an emergency call, or allowing rescue services to locate the user in case of an accident (e.g. using GPS).

Symbian OS is an open operating system. This means that the codebase is transparent, but it also means that the operating system allows the software on device to be upgraded and extended after it has been shipped from the factory. Consequently, the software cannot be optimized for specific tasks, but rather the software needs to be flexible, adaptable and allow future changes. An exception for this rule are tailored mobile phones for one special purpose (e.g. mobile devices reporting health information automatically to the hospitals). On the other hand, if the delay-tolerant software is going to gain importance among the typical end-users, it should support parallel operations and upgradings in the software and handle correctly the consequences of these upgradings also in the delay-tolerant software.

Mobile device end users, device manufacturers, retailers, network operators and suppliers of software, services and content expect the mobile device to be secure against the software attacks. Mobile devices usually contain private data, that must be secured. The risks for the Symbian mobile devices are more significant because Symbian allows software to be installed on a mobile phone. Also, devices and multiple communication channels of the devices support many services that are charged from the user increasing the importance of the security to prevent usage violations, and huge surprising costs from the users. Mobile devices in a DTN network must be especially secure because of the scarce resources, e.g., the mobile device should not forward by default any unwanted traffic that drains the battery.

The user interface of the mobile device suffers from the small screen sizes and minimal keyboards. These constraints pose restrictions for the applications, including

delay-tolerant ones, on a mobile device. In addition, the user interface of the delay-tolerant application has a challenge to be more responsive, because a response for a request may come after some arbitrary time after a request has been made, or it may come never. It is clear that delay-tolerant applications have even more restrictions to the user interface design than the traditional applications.

3.2 Forces for software development

Forces for the mobile software development [33] that need to be taken in account are reviewed next. These forces are caused by the mobile device constraints that we described in the previous section. Moreover, the forces need to be taken in account when implementing DTNS60, and mobile software generally to provide the end-users a pleasant user experience.

Firstly, it is reasonable to reduce the memory usage of the mobile devices due to limited amount of read-access memory (RAM) available. Also, reducing the RAM can result in the less power usage, since power is required to store data in RAM. By contrast, the execution time may increase, if a lot of caching techniques (that consume memory) are reduced, and therefore, there is a trade-off between the RAM usage and the execution time. The three main contributors to the RAM usage are the call stack, the heap and the code of the application (if not executed-in-place). In addition, Symbian C++ offers the cleanupstack-mechanism that effectively prevents memory leaks before deploying the software (and leads to more reliable software, e.g., when a memory leak does not consume all the memory unexpectedly).

On the other hand, RAM will eventually run out (e.g. when multitasking programs), and it is important to predict this. For example, when placing an emergency call, the function should not fail because the software was not able to allocate enough memory due to less important other applications that were consuming all the memory. Predicting the RAM usage will also lead to more reliable software.

The code, read-only data and persistent data are stored in the secondary storage, i.e. in the flash disk or the hard drive. Accessing to the secondary storage is much slower than accessing to the RAM, and the secondary storage space is also limited compared to the desktop or enterprise computers. Therefore, this data should be minimized on a mobile phone. On the other hand, a delay-tolerant network relies heavily on the persistent storage available in the network. This affects to the policy that is used to accept custody of bundles in a mobile phone, and also which mobile

phones are used; it does not make sense to use low-storage phones in conjunction with the distributed storage requirements of the delay-tolerant network.

Minimal execution time for software may mean, for example, starting an application or responding to a key press from the end user. The main concern is to execute a particular use case in the shortest execution time as possible. A use case is limited always by a bottleneck, and in the delay-tolerant software, the network bandwidth is the bottleneck. Therefore, the goal is to make every other component of the software execute efficiently and not create another bottleneck that is more severe than the network connection. An unwanted scenario is that the connectivity opportunity is missed because the software is processing other functions, and is not able to take advantage of the available connection, i.e., the function to open the connection has been blocked.

Real-time responsiveness, that is related to the minimal execution time, is important for delay-tolerant software, even though bundles cannot meet real-time constraints when sending or receiving those from the remote bundle node. However, software can still be separated to real-time and non-real-time operations, where some functions can meet real-time responsiveness requirements. For example, routing decisions in a bundle node should work in a real-time manner.

Reliability is the ability of the software to successfully perform the required functions in a specified period of time. The drawback for creating the reliable software is that it is a time and resource consuming process, that turns out as increased cost and effort for development and testing. Therefore, only a proof-of-concept implementation can be developed in the scope of the thesis.

Security in general, in addition to the security of the bundle protocol, is important for the delay-tolerant software. Secure software prevents unwanted actions to happen in a mobile phone, for example, to accidentally delete important files that are required to keep software running. Symbian has many Symbian-specific security mechanisms, such as, data caging and signing the software. Drawbacks to bring security may be increased development effort and more limited usability due to limited user actions.

The software should be designed to minimize the total development effort, especially in the proof-of-concept implementation. Creating the design and the architecture for the software, that are created early in the software development project, affect implementing, testing and maintaining phases. Therefore, also DTNS60 must be designed and architected carefully before starting the implementation phase. On the other hand, DTNS60 has a legacy burden, that is described later in the thesis,

that restricts the designing phase (which would be more flexible if started from scratch).

Testing is used to assess the quality of the software. It provides a level of confidence for the software. Testing environment should be made easy to establish assurance of the software. We introduce test-driven development method for Symbian later in the chapter.

Maintainability is described as: "The ease with which a software product can be modified in order to correct defects, meet new requirements, make future maintenance easier or cope with a changed environment" [35]. Maintainability is especially important for the bundle protocol implementation, that is experimental and may change in the future.

Good encapsulation of the software provides well-defined interfaces for software objects. An object in a software is like a black-box; there is no need to see inside the box, instead only the input and the output are important. Improving encapsulation improves reliability, maintainability and reduces the testing cost.

3.3 Mobile software platforms

Currently, there is no efficient way of coding once and deploying for every major software platform the same code. Therefore, platform-specific implementations are needed. To keep the scope of the thesis reasonable, we choose to implement the bundle protocol functionality in the protocol stack of Symbian, that is, the most popular mobile software platform today. In order to achieve a bigger picture where the Symbian bundle protocol implementation lies, we introduce other main alternatives for Symbian, that are, Maemo, iPhone, Android, RIM, WinMo and Limo. A brief description of the each platform is described below from the software development point of view. The characteristics of Symbian, and the Symbian-specific frameworks used in our implementation, are described in more detail after this section.

Symbian is an open mobile software platform [65]. There are many implementation language alternatives to use on the Symbian platform. This is based on the many runtimes that Symbian offers on top of the native environment. Symbian C++, C/C++, Python, .NET, Java ME and FlashLite applications can be developed for Symbian phones. Also, web technologies (e.g. html and css) can be used to develop widgets (small applications). On the other hand, it is

always more efficient, and a wider set of APIs are available, when developing a native Symbian C++ solution.

Maemo platform is built largely from the open software components [40], and has Linux as an underlying operating system. Future versions of Maemo will use Qt as a main application and GUI framework, and this enables also cross-platform support between Symbian and Maemo devices, i.e., the same source code can be deployed on both platforms in the future.

Linux mobile is also built around an open operating system [37]. It offers a secure run-time environment to support downloadable applications, and is based on modular plug-in architecture. C and C++ are most popular implementation languages.

iPhone is a closed source mobile platform [30]. A native implementation language used in the iPhone platform is Objective-C.

Android is an open source mobile software platform [1]. It offers Java as means for implementing applications.

RIM mobile platform [58] is most famous of the Blackberry mobile phones. It is closed source. Java is used as the implementation language.

Windows Mobile mobile platform [77] intends to bring many powerful desktop technologies, such as, Windows Presentation Foundation [78] into the mobile devices. WinMo is a closed source mobile software platform. Visual C++ and Visual Basic are used as the implementation languages.

As of our knowledge, Android and iPhone are gaining popularity in the mobile software development community, but Symbian remains as the largest mobile software platform by far [65]. Also, Symbian provides a lot of low-level accessibility and multitasking capabilities that may not be apparent on most of the other platforms. Furthermore, Symbian is going to have multi-core mobile devices available in the near future, that provides means for creating more efficient ways to do multitasking (e.g. daemon processes use separate core than other applications). Finally, being an open mobile software platform, Symbian may enable more innovation in the future with the delay-tolerant software.

There is a heavy tussle going on among the different mobile platforms today, but our belief is that Symbian will live many years onwards, and is therefore a reasonable

platform to implement also novel technology concepts, such as, the delay-tolerant networking concept.

The goal of the thesis is to demonstrate a proof-of-concept implementation on a mobile phone, and therefore one platform is enough for now. However, a next step, is to have the coverage of the bundle protocol implementations to cover more mobile software platforms interoperating seamlessly with each other in an ad-hoc manner, or with help of infrastructure access points. This would require many platform specific-implementations. The generic abstraction layers in a mobile phone for the delay-tolerant software is depicted in Figure 5.

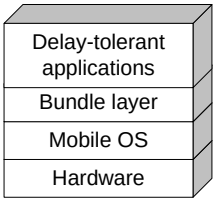

Figure 5: Abstraction layers on a mobile phone

Many implementation programming language possibilities are enabled providing many runtimes on the mobile software platforms. On the other hand, it makes sense to implement native implementations instead running the code binaries on a runtime. It is a matter of future research to evaluate different implementations, i.e., to compare implementations that are running on a runtime and native implementations with each other.

To reduce the development effort, it should be noted, that more cross-platform software development support is better. We discuss later about cross-platform development using the Qt application and the UI -framework when introducing the future development possibilities of DTNS60. Finally, we considered only the "native" platforms here. There exists also Java ME [34] runtime that offers also cross-platform development possibilities, and that is not tied to some particular manufacturer's phones. Drawback for Java ME is a restricted set of APIs available to access lower level functionalities in the mobile phone.

3.4 Symbian mobile software platform

Symbian is our target platform for the proposed bundle protocol implementation, and therefore, we will introduce in this section the S60 application framework, and

the main frameworks for implementing a bundle node implementation for the S60 mobile phones.

3.4.1 S60 application platform

The S60 application platform is part of the world's leading software platform for mobile smartphones, and it is based on Symbian OS. Consequently, it is a convenient choice to implement delay-tolerant applications for the S60 platform because of the widespread availability of the S60 mobile phones. S60 offers a rich set of application programming interfaces also for the lower level services [59].

S60 provides many frameworks that ease solving particular kind of problems, e.g., concurrent programming and communication services related problems. Furthermore, the main release of the legacy DASM implementation has been targeted to the S60 platform, so it is a natural path to continue development with that release instead of doing everything from scratch.

The attractiveness of the S60 has been increased since the Qt was announced to be merged with S60 [65], and to replace the S60 user interface layer. Qt for S60 lowers the barrier to develop cutting-edge delay-tolerant applications on the S60 platform.

The S40 platform would also be a popular choice. But, because there has not been any earlier development effort concerning DASM implementation on that platform, it is ignored for now. The development on S40 must be made using Java ME so the implementation would have to be made from scratch.

However, S40 is a good choice to keep in mind for the future, especially because of the cheap price of the S40 phones and popularity in the developing countries, where challenged network environment are apparent because of the lack of the basic network infrastructure.

The first versions of DASM supported also S80 platform, but since development of that platform has stopped entirely, also further development of S80 version of DASM is dropped. The main focus is now on the S60 version.

The target S60 mobile phones for DTNS60 have at least Symbian OS v9.1, for example, Nokia E90 and Nokia N95 8GB, that we use in the evaluation and testing, are running Symbian OS v9.2. The design goal for DTNS60 is that it is written in a way that it is directly reusable beneath the UI layer on different supported versions of Symbian OS. At least in theory, the DTNS60 should be runnable on all S60 phones that have Symbian OS v9.1 and onwards, but need to be tested on every

target phone (that is out of the scope of the thesis), if willing to use other than the two we use. S60 3rd edition feature pack 2 software development kit is used in the development of DTNS60, even though also newer SDKs (S60 5th edition and N97 SDK) exist today. This is due to usage of the non-touchscreen devices that use the older version of Symbian OS. It should be noted also that our test devices are natively S60 3rd edition feature pack 1 devices, but that also support the feature pack 2.

3.4.2 Client-Server framework

The purpose of the client-server framework is to provide services from the server program to the multiple client programs [3]. The server handles resources on behalf of the clients. These services are, for example, Windows Server, File Server, Socket server and font and bitmap server. When the server runs in its own process it provides modularity and protection from the ill-behaving clients.

The main purpose to adapt the client-server framework for the bundle protocol implementation is to separate the bundle protocol layer from the delay-tolerant application layer in the communication protocol stack. But there are also other reasons for the client-server framework. The framework is usually chosen to be the design principle if the management of the shared resource, asynchronous services, or protection offered to run the server in separate process than the clients is needed. It is the "Symbian way" to provide services to applications, i.e., it is based on the robust microkernel operating system architecture [71].

In DTNS60, the asynchronous services are required, because of the whole nature of the asynchronous communication in the DTN network. When several delay-tolerant client applications are running simultaneously they share underlying communication resources. Moreover, if the clients are just waiting for bundles, using asynchronous server to handle receiving, the user interface does not seem like jammed and can be shown to be more responsive for the end user.

There are four key concepts in the client-server framework that are the server, the session, the sub-session and the message. These all have their base classes provided by the framework.

The server is responsible for handling any requests from clients to establish connections. The session and the sub-session represent the channel of communication between the client and the server.

The message represents data structure passed between the client and the server. In DTNS60 the data that is passed between the client and the server is application data that is encapsulated as bundles in the server, and sent further to the delay-tolerant network; or data that is decapsulated from the bundles in the server and sent to the local destination delay-tolerant application.

Fragmentation and reassembly of the bundles are done in the server-side so that only the whole bundle payloads are passed between processes. The basic class diagram that must be fulfilled using the client-server framework in DTNS60 is provided in Figure 6. This is slightly updated from the original design [41] (that was not implemented in the legacy implementation). The architecture for DTNS60 supporting the client-server framework is provided in the chapter 4.

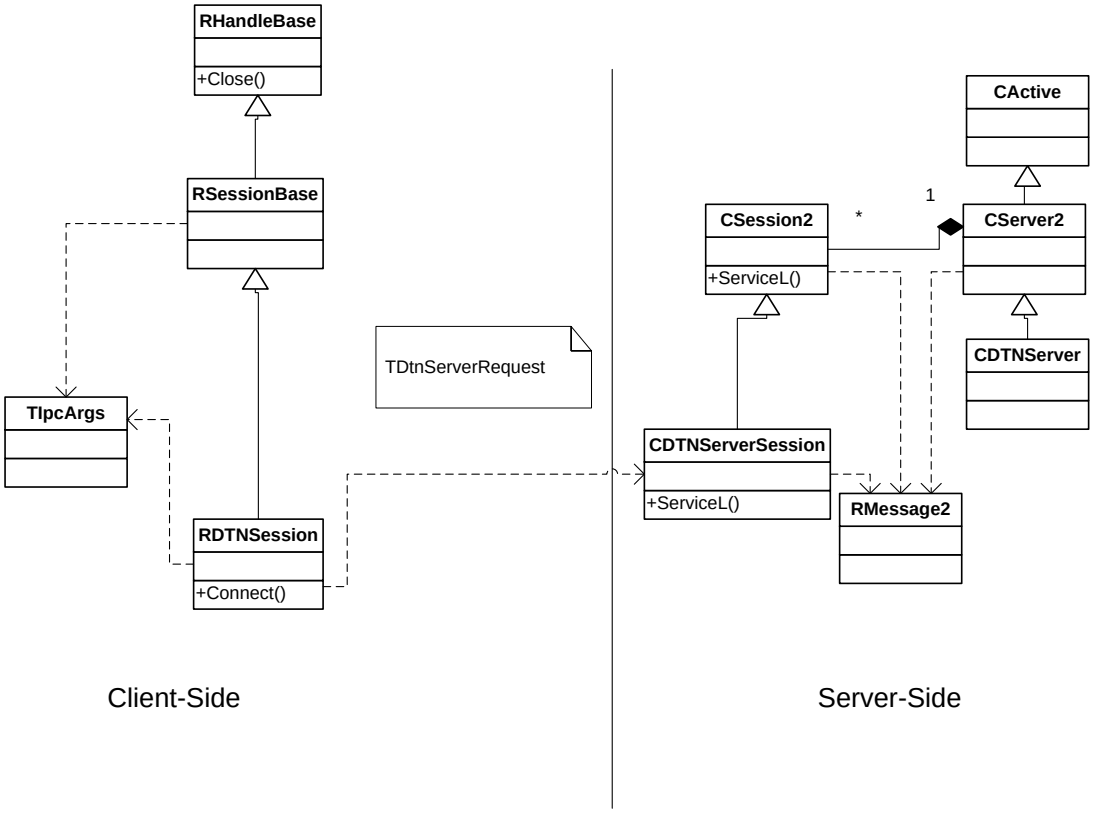

Figure 6: Client-Server framework for DTNS60

3.4.3 Active object framework

The active object framework [3] provides means for the asynchronous processing of operations and simplifies it for the developer. The framework handles multiple

asynchronous tasks in the same thread using an active scheduler per thread. Also, the framework encapsulates a service request and a response of the completion. Furthermore, active objects provides means for event-driven programming, when the framework calls the right method after the completion for the request has occurred (e.g. a bundle has been received in the DTNServer). The active objects are favored in concurrent programming instead of threads in a Symbian phones due to resource-saving reasons. Also, there are only one-core processors available in the market today, and the context-switching between threads is more expensive than using active objects, when the context-switching is not needed.

3.4.4 Avkon GUI framework

In native Symbian, there are currently three design approaches to create application user interfaces. Those are using of the traditional Symbian control-based architecture, the dialog-based architecture or the Avkon view-switching architecture [20].

We use the last Avkon-based approach in our example delay-tolerant applications since it allows us to create multiple views conveniently for a limited-sized mobile device screen, even though the view-switching architecture is more complicated than the other two.

The view-switching architecture makes application navigation between local and external (other applications) views seamless. For example, when a user receives a bundle from some contact that she have in her contactbook, but whose end-point-identifier she wants to add to the contact's details, she would need to start the contacts application on a mobile device and add that particular contact to her contact list. Without view-switching it would be necessary to start the contact application manually, and manually navigate to that contact entry and then enter the details of the end-point identifier.

The Avkon view-switching architecture uses the MVC design pattern for creating GUI-based applications, i.e., an application logic, views and data are separated from each other in the application structure.

3.4.5 ESOCK communications framework

ESOCK[8] provides two main services for an application developer. The first one is the sockets API for creating connections with the remote systems and transferring data. The other one is the connection management API for selecting, configuring

and monitoring connections. For example, core parts of the Bluetooth stack, most of IrDA, most of the TCP/IP stack and the SMS stack are accessed through the ESOCK APIs. Figure 9 depicts how ESOCK fits in the delay-tolerant software architecture in a Symbian phone.

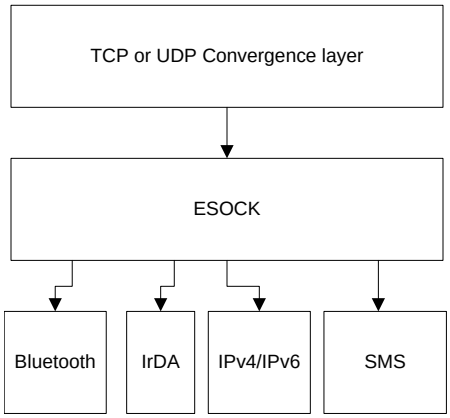

Figure 7: ESOCK

3.4.6 Multimedia framework

The multimedia framework [59] is used for recording and playing audio in the DT-Talkie voice application introduced later in the case study. The Symbian phones offer different level of access for the different codecs depending if the application developer has a partnership contracts to access lower level hardware accelerated codecs, such as, MP3 codecs. The multimedia framework can be used also for playing and recording video content.

3.4.7 Design patterns

Design patterns describe solutions for the common programming problems. They do not give the actual implementation in any specific language. Using the design patterns prevents reinventing the wheel in situations that many other developers have faced before. The design patterns also increase the quality of the software. Several design patterns are used to speed up the development of the DTNS60 implementation. Active Object, Event Mixin, Client-Server and MVC design patterns [33] are considered to be useful in the next release of DTNS60, and we describe those briefly. For further development of DTNS60 the investigation of using additional design patterns should be made. For example error-handling strategies, resource lifetimes,

security and optimizing execution time may be the goals to choose additional design patterns.

Active Object

Symbian OS is a heavily asynchronous operating system. When a program requests a service, the service is performed either asynchronously or synchronously. To avoid blocking in the user interface an asynchronous service is a more convenient way to do the request. Symbian OS provides co-operative multitasking within a process by using active objects. In the case of DTNS60, the active objects enable the use of multiple client applications simultaneously, and nonblocking behavior of the GUI-based application, when the application is, for example, waiting for the incoming bundle. The concurrent execution is done more effectively using the active objects instead of the threads (if there is a single-core processor in use) because there is no need for the context-switching between threads. Furthermore, the active objects provide memory consumption savings and battery power savings. The Symbian OS Active Object framework, introduced earlier, eases the use of the active objects in the applications.

Client-Server

The intent of the client-server design pattern is to securely police access to a shared service from multiple clients. The server component defines an API and offers services through that API to the client applications. The design pattern supports high cohesion and low coupling. High cohesion means that subsystem includes only those functions that are related to each other and low coupling means that interaction between the different subsystems are kept at minimum. Changing one subsystem should affect only that one and others stay unchanged. The client-server design pattern is used in the Symbian OS client-server framework.

Event Mixin

The intent is to save power and decouple components by defining an interface through which event signals are sent to another component in the same thread. An event generator notifies any state changes of their subject to other objects, usually by calling one of their methods defined in the interface. The event mixin pattern is used in DTNS60, for example, to notify of a received bundle or a bundle fragment in the bundle protocol implementation's core classes. The event mixin is also known as observer. In general, this pattern can be applied to the event-driven programming paradigm, described in the end of this chapter.

Model-View-Controller

The model-view-controller design pattern is a central part of the delay-tolerant GUI-based applications for the S60 platform. It defines the structuring of the application to a model, a view and a controller components. The model component contains and manipulates the data of the program. The view defines how the model is presented to the user. The view also forwards the requested command to the controller. The controller defines how the user interface reacts to the received commands and requests.

3.4.8 Test-Driven development

Test-driven development is a technique for the software development, where the development iterations are based on the pre-written tests. Each new function is tested right away providing rapid feedback to the developer to fix errors in the early stage. The tests also track the real progress of the development effort. Creating well-tested software increases the quality of the software and it gives confidence of the reliability of the software.

Symbian OS Unit [66] is an open source unit test framework for Symbian applications. Using this framework, test-driven development can be utilized in the development of DTNS60. Therefore, the framework is integrated in DTNS60 to produce more quality code. Also, delay-tolerant applications can be tested using the same test component.

The goal is to test every functionality and write the tests before the implementation of the actual function. The test project is integrated in the build system, but acts as a separate project from DTNS60, and is not going to be in the final phone release.

3.4.9 Nokia Energy Profiler

Nokia Energy Profiler (NEP) [44] is the main tool for doing resource measurements for DTNS60 in the validation part of the thesis. It provides tools to measure energy consumption and other resource usage in real-time on the target mobile phones. The measurement data can be exported for later use and plotting.

Different metrics, that can be measured using NEP are listed below.

Power - Shows the power consumption over a measurement period.

Current - Measures the current draw from the battery.

Processor - Shows the CPU load over a measurement period.

RAM Memory - Shows the memory usage over a measurement period.

Network Speed - Shows the download and uplink speeds through the IP stack.

WLAN - Shows received signal strength when the device is connected to the WLAN.

Signal Levels - Shows the cellular signal levels as RX and TX levels.

Energy - Shows the cumulative energy consumed over the measurement period.

Voltage - Shows the battery-voltage levels.

3G Timers - Shows the 3G-network-data inactivity timers.

We utilize the energy and the processor usage in the validation of the resource consumption of the implementation.

Finally, NEP provides APIs that can be used in an application to measure a real-time resource usage. It is possible to modify the behavior of the implementation depending on the resource usage when integrating NEP in to the implementation. This would be convenient in the challenged environments, where resources can be scarce and their usage should be optimized, e.g. going in the idle state when connectivity opportunities are not available, and waking up when the network coverage or ad-hoc contacts has been discovered. Therefore, it would be reasonable to add real-time resource usage APIs in DTNS60 in the future to enable more efficient behavior of the implementation in challenged environments. NEP also allows to developer to create customized measurements.

3.5 Component-based software engineering

A software component is a modular, deployable, and replaceable part of the system that encapsulates implementation and exposes a set of interfaces [52]. In other words, the software component offers services to the other software components. The abstraction level of the component is higher than in the object and components do not share state as objects do. Messages are used in between the communication of the different components. DTNS60 uses this practice and separates different parts

of the bundle node system to different reusable components. Those components are roughly: The DTNServer process and delay-tolerant applications processes that use the bundle protocol services of the server. These two component types can be further decomposed to different library components, such as, TCP CLA dynamically linked library. Using this kind of modular design the system can be developed more efficiently and different parts can be replaced with more efficient solutions, if necessary, or adding new functionality is possible without affecting the other components.

Four basic design principles apply when using components [52]. The Open-Closed Principle tells that the module should be open for extension but closed for modification. The Liskov Substitution Principle tells that subclasses should be substitutable for their base classes. The Dependency Inversion Principle tells to depend on abstractions and not depend on concretions. The Interface Segregation Principle tells that many client-specific interfaces are better than one general purpose interface. Applying these principles in the design and development of DTNS60, the quality of the software increases. More about the design of DTNS60 is discussed in the next chapter.

3.6 Event-Driven Programming

The software running on the mobile phone has a direct impact on the power consumed by the CPU, because the CPU uses power for each executed CPU cycle. The fewer the CPU cycles executed, the less power is consumed.

Polling is a mechanism where a thread periodically executes some command, e.g., checks whether new bundles are received in the bundle node. This is wasteful for the CPU, and consequently may be reduce the battery-life significantly. Lengthening the polling period reduces the CPU usage, but creates unresponsive software.

The alternative for the polling mechanism is event-driven programming [33]. It allows the CPU and other hardware components to be used in most power-efficient and responsive way.

Events are some significant occurrences, for example, receiving the bundle from the remote bundle node, or keystroke in the user interface. An event signal is a notification of the event. The event signal indicates that the event has occurred at some software component. An event consumer is an entity that receives the notifications for a particular events. An event generator is an entity that produces events and sends event signals to the event consumers. Many software components

can act at the same time as event consumers and event generators. Also, forming an event chain to accomplish some task is possible.

Using the event-driven programming paradigm in the delay-tolerant software is considered to be a good practice. Because the communication is asynchronous, it is impossible to know exactly when the event occurs, for example, when the bundle is received. Therefore, CPU or other hardware components are utilized only when an event occurs - instead of polling - and this creates more power-saving software. The drawback is, that one event loop is needed anyway, that has a direct impact on the performance of the software (depending on the polling period). The event loop is polling for new events that can be any kind of action happened in the bundle node system.

3.7 Summary

In this chapter we reviewed the critical technologies and paradigms for the development of the Symbian-based delay-tolerant software. We also reviewed constraints of mobile computing and forces to the software development under these constraints. We briefly reviewed different mobile software platforms and explained why we chose Symbian as the target platform. The Nokia Energy Profiler tool was introduced to provide means to measure the proposed implementation described next. NEP can be integrated in the future to DTNS60 to offer more efficient resource-saving behavior in the challenged environments where resources are scarce. Now, we have have studied the main requirements for the bundle node implementation, and the main methods to implement it on the Symbian mobile software platform.

4 Implementation

This chapter introduces a novel bundle node implementation, called DTNS60, for the Symbian S60 mobile phones. DTNS60 is composed from many software components, i.e., the implementation includes the bundle protocol service and also a few example applications - or tools - for the end users. These tools use the common bundle protocol service offered by the separate DTNServer daemon process. The bundle protocol service itself consists of the bundle protocol agent [62] and the TCP convergence layer [15] library components. We introduce the design, the architecture and the functionality of the bundle protocol service and the tools. Also, the delay-tolerant application programming interface of the bundle protocol service is described. However, first the legacy for this software is discussed.

4.1 Legacy

The legacy for the DTNS60 implementation is originated from the implementation called DASM[41]. The latest DASM version 0.3.0 contains a bundle protocol agent, a TCP convergence layer adapter, and a GUI-based file transfer application for transferring MIME messages using the bundle protocol. The MIME-message implementation is custom-made for the Symbian environment, i.e, it is not provided natively by the Symbian platform services.

This version of DASM has some fundamental flaws in the implementation that differ from the original design decisions [41]. The original intention was to develop a client-server framework -based separation of concerns for the different conceptual components, but the architecture of DASM does not fulfill this decision and is the most severe defect in this release. In other words, the bundle protocol is not separated programmatically from the application processes. This leads to the situation where ill-behaving clients can prevent other delay-tolerant application to use the bundle protocol service, and the development of the 3rd party delay-tolerant applications becomes difficult.

Indeed, all the functionality is tightly-coupled into the same component in the single executable. This leads also to an inefficient and not scalable solution. It is, nevertheless, visible in the DASM version 0.3.0 that the implementation is not complete (also the version number indicates that). Because DASM is only partially implemented against the requirements specifications [62, 15, 10], further development is needed to get more coverage of the requirements fulfilled, and to create the software

architecture more robust.

Running more than one delay-tolerant applications would facilitate to the binding of the same bundle protocol implementation over and over again in the different delay-tolerant applications. It is generally not a good idea to introduce duplicated code in a memory-constrained device, and maintaining possible different copies of the same bundle protocol service for the individual applications is overwhelming, and not reasonable, when there is a possibility offered by the software platform to apply the multitasking of the applications in a more convenient fashion using a singleton[22] service in the mobile phone.

The bundle protocol agent in the legacy implementation is based on one main event-loop. This event-loop processes bundles as those are received from the remote bundle nodes or from the application agents to the local message queues. The polling period for this event-loop, for which the whole bundle protocol implementation is based on, affects to the efficiency of the implementation (as can be seen in the next chapter in the throughput capability measurements since the next release has also this feature originated from this release). Choosing a too long period leads to an unresponsive software, and choosing a too small period drains battery of the mobile device quickly. Polling is generally inherently evil in a portable battery-powered device because it prevents the system for going to lower power modes, and hence reduces the battery-life. Therefore, it is important to keep only the one main event-loop and not to introduce more polling mechanisms in the future releases in other parts of DTNS60 system. Every event for the bundle protocol should be originated from this event-loop and message queues that it handles. There are currently three message queues. One for received bundles from the remote bundle nodes, one for bundles to be transmitted to the remote bundle nodes, and one queue for processing different events, such as, receiving a bundle from the convergence layer and passing it to the message queue that handles received bundles. The queues are processed consecutively one after the other (that may introduce processing delays).

The bundle protocol agent provides an interface, than can be partly adapted for the next release (for receiving and sending application data). The existing bundle processing logic offering basic application data bundling, bundle parsing and delivery functions can be adapted straightforwardly for the next release. But more advanced features, such as, registrations or reactive and proactive fragmentations and persistence of bundles are missing from DASM.

A routing information base and a forwarding information base are statically config-

ured to the configuration files that must be given separately outside the program in the mobile phone's file system. To be more precise, the routing information base is actually lacking since no routing protocols are implemented in this legacy implementation. Using static files is a error-prone way to do configurations for routing, and is not scalable, and not viable if the mobile devices receive underlying addresses dynamically from the access point. The routing configurations should be done in a more dynamic fashion, or even automatically leveraging ad-hoc peer discovery mechanisms. However, there is no ad-hoc peer discovery implemented in the legacy release, i.e., a user must know beforehand all the endpoints and configure them to the forwarding information files in a static fashion. The ad-hoc peer discovery mechanisms and the routing protocols are features to be implemented in the future releases.

A TCP convergence layer adapter is implemented in the legacy version. Implementing Bluetooth should be an easy task since the streaming-based base classes are implemented already. Those provide many common features for stream-based convergence layers, i.e., both the WLAN and the Bluetooth implementations can use the same base classes. Furthermore, both use the ESOCK communications framework offered by the Symbian platform as an underlying service.

In the legacy implementation, there has been used the observer (also known as event mixin), and the active objects design patterns in the current release in the bundle protocol agent and the tcp convergence layer. The observer has been used to manage events and displaying feedback for the end-user in the user interface. The active objects are used for concurrent processing, such as, for listening incoming bundles and sending bundles at the same time.

In spite of the shortcomings, DASM implementation offers a groundwork to continue further development of the native delay-tolerant software for the Symbian phones offering lot of objects that can be reused in the bundle node implementation fulfilling the RFC5050[62].

We list a more detailed list of the features implemented in the legacy implementation in Appendix A.

Finally, it should be noted that the DTN2 reference implementation has had an impact on the legacy BPA and TCP CLA implementations, for example, in structuring of the TCP CLA.

4.2 DTNS60

DTNS60 is a novel bundle node implementation for a Symbian mobile phone, and it is a next release of DASM (and is renamed at the same time). The current version of the DTNS60 release, as the time of writing the thesis, is 0.4.1. The remainder of the thesis is based on this release.

DTNS60 is a suite of several software components that work in conjuction with each other to offer the bundle protocol functionality in the communication protocol stack. The main components are: the DTNServer daemon process and the delay-tolerant application process(es), that are, the components supported by the Symbian-specific client-server framework introduced in the chapter 3. The server process is a singleton on a mobile phone, and multiple delay-tolerant applications can use the bundle protocol service of the server at the same time. These client and server processes load their functional engine components from the dynamically linked libraries at runtime. A lot of additions, changes, re-architecturing and refactoring have been made to enhance the implementation from the legacy codebase. However, the bundle protocol agent and the TCP convergence layer components are still heavily based on the legacy code, even though those are split into the different library components.

Thus, the architecture of DTNS60 has been completely re-architected, even though conceptual components remain the same as in the legacy implementation. The new architecture allows multitasking of different delay-tolerant applications, and introduces a common application programming interface (API) for the bundle protocol service. Consequently, the development of new applications against this common DTN API has been enabled. Only the API has been exposed for a delay-tolerant application developer, so he or she does not need to be aware of the underlying layers utilizing the bundle protocol and traditional heterogeneous networks. The architecture is designed so that the solution can be extended, and that the further development of DTNS60 is enabled and encouraged.

4.2.1 Design considerations

Designing software for the mobile devices has constraints, and forces on the software development caused by these constraints, that we described earlier in the chapter 3. Also in that chapter, the component-based software engineering principles and event-driven programming paradigm were introduced that are essential paradigms in the current release of DTNS60.

We should design delay-tolerant software to work in conjunction with the traditional software to offer seamless transformation from the traditional software to the delay-tolerant software. In other words, the end-user should not be aware what kind of communication paradigm has been implemented beneath the user interface layer. Therefore, we divide the applications on a mobile phone to three categories from the software development perspective: delay-tolerant, traditional, and hybrid applications. These applications use delay-tolerant APIs, traditional APIs, or both first-mentioned APIs. In the hybrid applications, changing the mode between the traditional and the delay-tolerant mode can be seamless for the end-user.

Because we use the legacy implementation as the basis for our work, the goal is to reuse it as much as possible. The interface for the bundle protocol agent can be applied using an adapter class between the client-server framework classes and the classes from the legacy implementation, and to push the bundle protocol -specific code in to the separate library components. This design also allows us to replace every legacy component in the future releases, if necessary.

The DTNS60 implementation should provide different conceptual layers for different components and clear interfaces for these layers. Two layers must be provided: the bundle protocol layer (or the DTN layer), and the delay-tolerant application layer.

Finally, we want also to make unit testing of the software easy, and encourage the test-driven development in order to create quality software.

4.2.2 Architecture

A generic architecture for the bundle protocol agent and convergence layer adapters is depicted in Figure 8, that has been slightly modified from the original version [42].

In Figure 8, the traditional users use the traditional applications that utilize the services offered by the underlying operating system (usually TCP/IP communication services).

The delay-tolerant users use the delay-tolerant applications that are developed against the DTN API to enable delay-tolerant functionality. The delay-tolerant functionality has been implemented between the operating system and the delay-tolerant applications - in the bundle layer. It is important to not involve the developer of the bundle protocol to choose different configurations on behalf of the end-user. Therefore, configuring the bundle protocol core logic must be enabled for the users, if choosing the right functionality, for example, choosing the right convergence layer

adapter, has not been automated in the code.

The hybrid users and the hybrid applications are in-between the other two. They choose on-demand the bundle protocol or the traditional protocols for communication. The hybrid applications have been extended from the traditional ones to use also the DTN API, when intermittent connectivity is apparent; or from the delay-tolerant applications to support also the traditional communication protocols, that may be more efficient than using the bundle protocol, even though the bundle protocol supports also the continuous connectivity. An option to choose between a delay-tolerant mode and a conventional mode (normal socket communication) will be enabled in the future releases of DTNS60 to create hybrid applications. Moreover, the usage of the traditional or delay-tolerant APIs should be automated, i.e., choosing the right API depending on the present network conditions, where the applications is running.

More applications are available using the traditional protocols (usually TCP/IP), and therefore, porting from the traditional APIs to use the DTN API is more convenient way to develop delay-tolerant applications, and furthermore, can reduce the development time compared to developing the delay-tolerant applications from scratch. On contrary, the traditional applications may not support asynchronous messaging offered by the bundle layer; instead near real-time requirements are needed.

The DTNS60-specific component model of the bundle node architecture is depicted in Figure 9. The delay-tolerant application engine and GUI components form the delay-tolerant application layer. The DTNServer, the bundle protocol agent and the TCP CLA form the bundle layer in the DTN architecture.

The picture illustrates also a larger context where the implementation can reside in the DTN testbed. The goal is to combine fixed and mobile infrastructure networks to form an overall testbed for the delay-tolerant networks using the bundle protocol in several software platforms. The other platforms, in addition to Symbian, may be, for example, Maemo, Mac, or Windows platforms. The intention is that the bundle protocol abstracts away the platform, and also mobility of the node, so applications can seamlessly communicate with each other, in spite of in which bundle node they reside. The picture does not show more than one delay-tolerant network, but it is also possible to combine more than one challenged network to communicate between each other through gateways (this was not our goal in the scope of the thesis, though).

The software architecture supports the Symbian-specific client-server framework functionality, that is; a Symbian-specific way of implementing asynchronous ser-

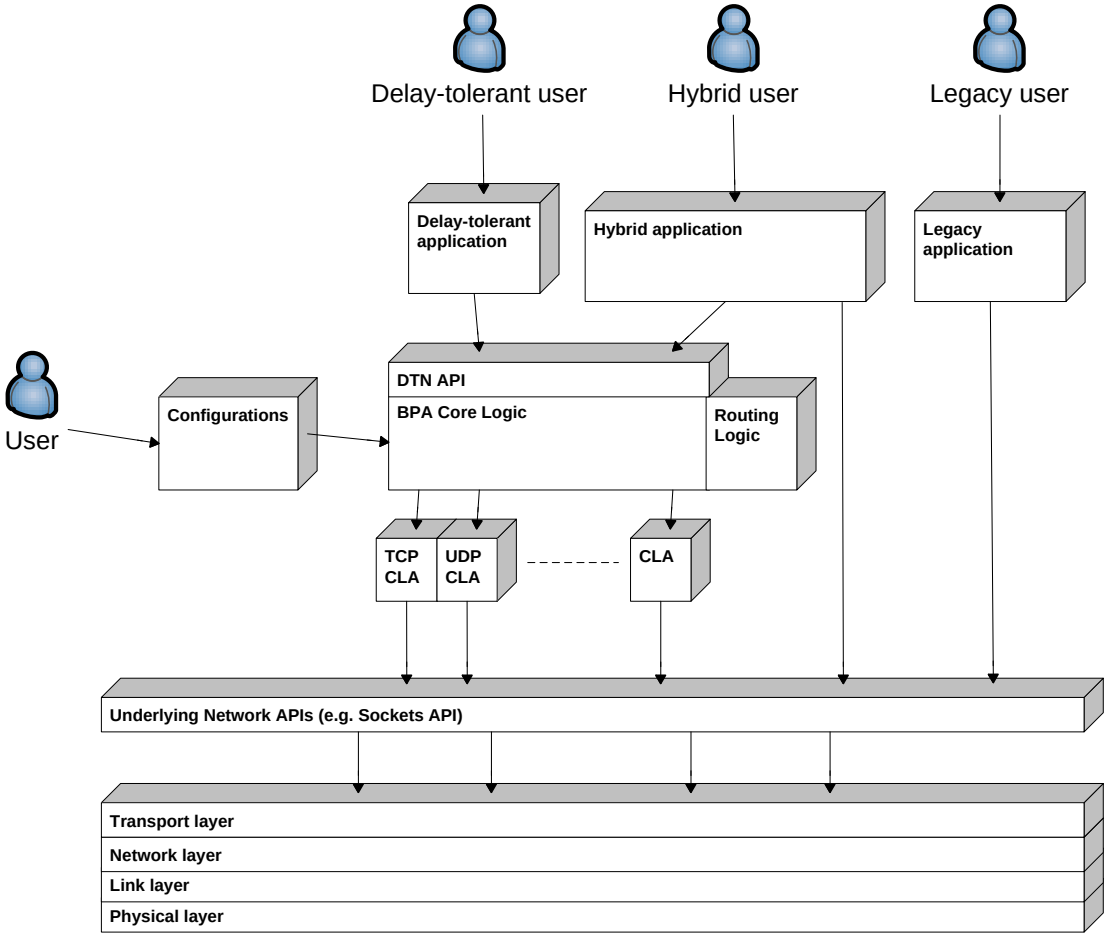

Figure 8: Generic architecture

vice providers [33], and conforing to the microkernel architecture structure [71]. Communication between the client delay-tolerant applications and the DTNServer daemon process has been implemented using the Symbian-specific inter-process communication mechanisms offered by the framework. Also, separating a graphical user interface from the application logic engine as a good practice has been taken into account. This enables the code of the applications to be more portable, for example, for touchscreen-supported mobile devices in the future. Our goal with this kind of architecture is to continue the evolution of the Symbian-specific delay-tolerant software. Adding new components to this kind of architecture is more convenient because new functionality can be developed as separate components (that can be completely independent projects) providing only an interface to the functionality offered by a particular component. The components of the current release, and their functionalities are explained in more detail in the following subsection.

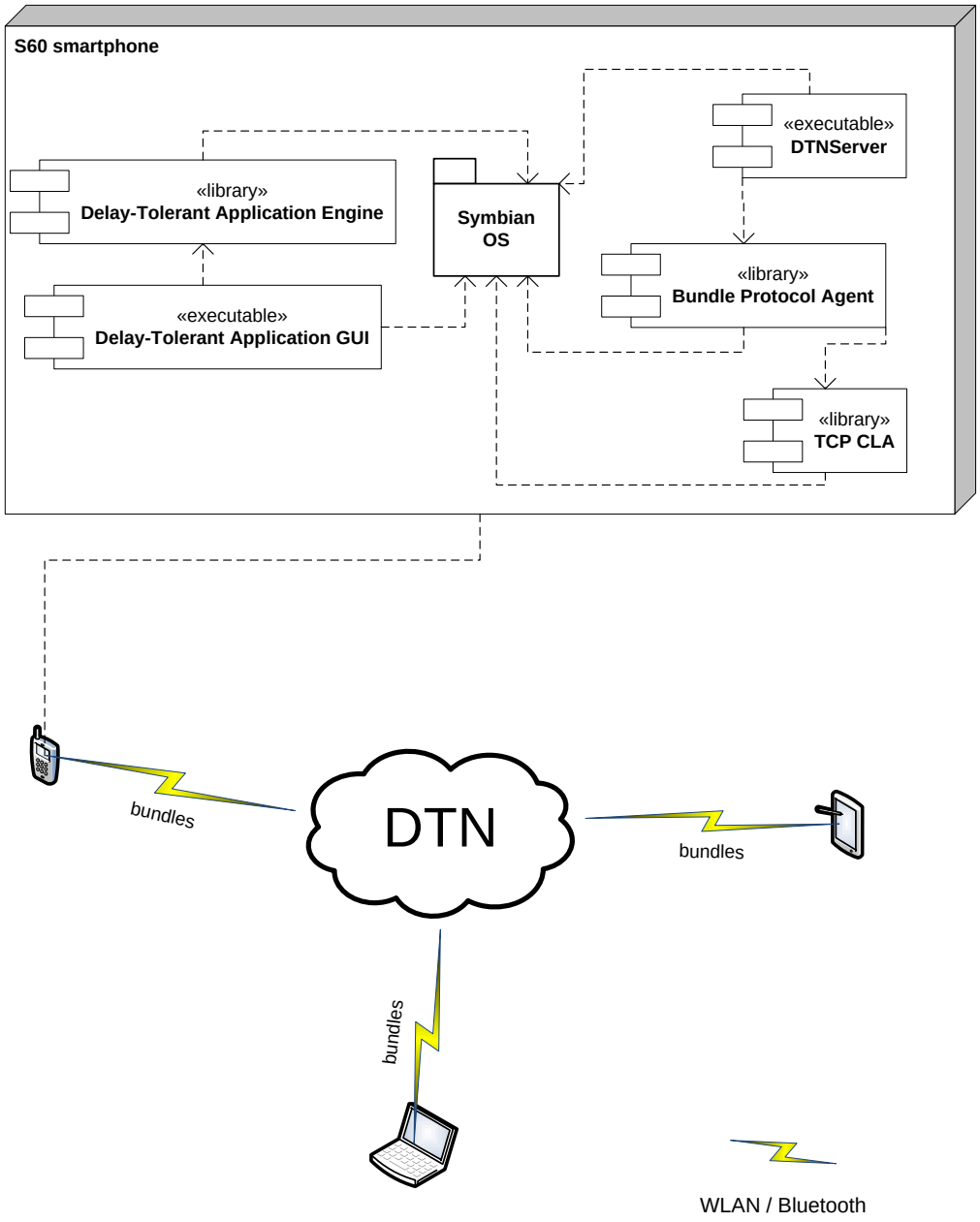

Figure 9: DTNS60

4.2.3 Functionality

The responsibility of the each component in Figure 9 is to perform a particular operations of the bundle node. All five components are loaded when starting a delay-tolerant application. The application engine and GUI components are described

more generically, because those can be any kind of delay-tolerant applications. A more detailed list of the features of the overall implementation has been provided in Appendix A.

DTNServer

The DTNServer is the server, or the service, part of the client-server framework supported application. It contains all the bundle protocol -specific logic in linked libraries. The dynamically linked libraries are used, because then the switching between different CLAs (or routing protocols in the future) is more flexible, and resources (mainly memory) are used more efficiently loading only the needed components at runtime. The communication between the client delay-tolerant applications and the service has been implemented using Symbian-specific inter-process communication mechanisms described earlier in the chapter 3.

The Bundle protocol agent is a DLL that makes it possible to be replaced or updated without affecting the rest of components. The DTNServer provides a common interface for the client applications. A delay-tolerant application always creates a session with the DTNServer daemon process for the lifetime of the application process. Also, the DTNServer is started and stopped depending on if any delay-tolerant application is using it's services. This way also the resource consumption is reduced as no idle processes run on the mobile phone. On the other hand, in a stressed environment, there should be some event indicating that new contacts are in reach, and waking up the daemon process. This is to be implemented in future releases of the software. In this release, one application needs to be running (e.g. lightweight DTNRecv) so the daemon is able to forward bundles even though the end-user of the phone is not interested in using the delay-tolerent applications.

Multiple delay-tolerant applications can use the singleton DTNServer executable component simultaneously on a mobile phone. This is accomplished by the multitasking capability of Symbian OS. The DTNServer is a dispatcher for the bundles, i.e., it dispatches the application data encapsulated in the bundles to the right applications.

In addition to the main responsibility of the DTNServer to dispatch requests originated from the delay-tolerant client applications to the bundle protocol agent and responses to those (e.g. deliver a bundle to the right application), the DTNServer also maintains sessions between the client and the server processes and provides a common and extensible application programming interface to the delay-tolerant applications.

The bottleneck of the server in the current release is the maximum size of the messages, that is 1MB, possible to deliver between the client and the server processes. Sending larger messages is undefined in this release. One solution for sending larger messages is to create delay-tolerant application layer level fragmentation and reassembly mechanisms.

Furthermore, the server currently contains legacy code which is originated from the legacy implementation when loading the libraries (BPA DLL and TCP CLA DLL). As mentioned, the new architecture allows to abandon and drop the inefficient and buggy parts of the legacy code and introduce a new more robust solution for the bundle protocol service components. However, the code should be investigated before dropping it since there are objects in the legacy implementation that can be reused efficiently also in the future releases.

Bundle Protocol Agent

The bundle protocol agent (BPA) library component contains all the logic that the bundle protocol agent uses, and is loaded by the DTNServer process at startup. The main services that BPA offers to the node are: registering a node to an endpoint, terminating a registration, switching a registration between active and passive states, transmitting a bundle to an identified bundle endpoint, canceling a transmission, polling a registration that is in the passive state and delivering a received bundle to the right delay-tolerant application. The bundle protocol agent uses a persistent storage to store the information of the pending bundles. It should be noted that the database enabling persistence is not supported in the 3rd edition feature pack 1 phones, that we use in the validating part; therefore, in this release persistence operations has been disabled.

Depending on the configurations, the correspondent convergence layer adapter is loaded. Also, the underlying routing information base and forwarding information base are configured and loaded at the same time as BPA (beginning the first application using the DTNServer). Other BPA-specific options such as delivery options can be also configured. Currently, the BPA uses static pre-configured files to load configurations. The current version of the BPA DLL does not fulfill all the requirements, and further development is needed to fulfill all the requirements for the bundle protocol agent[62]. The current version can be used, though, in the validation of the basic functionality, and in the application case study offering interoperability with the reference implementation. The BPA uses the services offered by the TCP CLA.

TCP Convergence Layer Adapter

The convergence layer adapter library is the interface between the bundle protocol and the transport layer. In DTNS60, TCP CL has been implemented also as a dynamically linked library, so that different adapters can be loaded as needed. For example, TCP CLA DLL or UDP CLA DLL can be used and switched on the fly when executing applications. However, the focus in this release of DTNS60 is on the TCP CLA. There are two main tasks for the TCP convergence layer adapter; it provides a service for sending bundles to all bundle nodes in the minimum reception group that is reachable via the TCP CL, and it provides a service for delivering received remote bundle to the bundle protocol agent. The TCP CLA DLL has been implemented against the specification but need to be hardened and tested more thoroughly.

Delay-Tolerant Application, GUI

A graphical user interface component is the visible part of the delay-tolerant client application to the user of the mobile device. It uses an application engine to render the screen, and for executing different functions. Symbian OS provides Avkon GUI framework for creating native Symbian GUI applications (described in the chapter 3), and it is used also in the example tools and in the application implementation that is described in the case study part of the thesis. The user interface of the delay-tolerant application should be designed to support asynchronous communication, i.e. it should provide some responsive elements to inform user that response for the request to the other bundle node may not come immediately or never. The GUI component uses the MVC design pattern. Also, it should be noted that console-based applications can be implemented on Symbian using the bundle protocol service.

Delay-Tolerant Application, Engine

A delay-tolerant application engine component, implemented as DLL, contains the logic of the GUI-based application. The engine provides also an application programming interface against user interface. Decoupling the non-UI parts from the underlying functionality makes the engine part almost entirely portable for different supported Symbian OS phones. The inter-process communication between the DTNServer is initiated from the engine component, when the application requests bundle protocol services to be used. The engine uses active objects for offering the concurrent functionality. For example, when the engine requests to receive messages from the bundle protocol service to the application end-point identifier, it does not jam the whole application, but allows the user to execute other functions in the

application at the same time.

4.2.4 DTN API

With a common and extensible delay-tolerant networking application programming interface (DTN API) the development of new delay-tolerant applications for the S60 smartphones, using the daemon process of DTNS60 for the bundle protocol services, has been enabled. Reusability of the bundle protocol service is the main design goal for introducing this API. However, this API is Symbian-specific since the Symbian platform uses the microkernel architecture, and the client-server framework for enabling the inter-process communication. Therefore, we loose cross-platform support for this API (for example, Linux-based devices uses commonly DBUS architecture [13] for providing IPC, that is, not supported in Symbian).

The current version of the API provides functions for sending and reading application-specific data from the bundle protocol service. This API is still evolving and will provide in the future, for example, means for sending metadata, registration information, and even functions that have not been specified in the bundle protocol service today. Next features for the API are to enable registering and unregistering a delay-tolerant applications for the bundle protocol agent, that further enables multitasking of the delay-tolerant applications, and dispatching of the bundles to the right recipient. This requires further development in the server component, for example, to implement a container of active and passive sessions (registrations) between the client and the server.

We illustrate the functionality of the DTN API at the level of methods calls using sequence diagrams. The DTNServer is an asynchronous service provider, and therefore, only asynchronous requests have been implemented in the current release (except when canceling an outstanding request that finishes always immediately). Synchronous requests can be implemented in the future but, keeping in mind that those are blocking other functionality of the application, if the response does not come immediately - this may be of only liked one in DTNs.

We look at two scenarios how the API can be used, and these can be thought of as a generic way of doing requests, or canceling requests that has been made earlier from the client to the DTNServer's bundle protocol service. Every request to the server has a same kind of sequence flow. The particular requests that we look at are: how to receive application data from the bundle protocol agent, and how to cancel the

previous read-data request from the server in case of, for example, shutting down an application.

It should be noted in these scenarios that, before requesting to read the received application data from the BPA, we have made beforehand a request to receive a notification from the DTNServer that there are data to be read by this particular application and got a response for that request. The response for this notification request initiates the introduced read application data request, and is called again after the data has been received to receive a next notification, that the next unit of the application data can be read.

Making DTNServer request

In Figure 10, RequestReadData is called first when the notification for the received bundle has been received in the client-side active object. This method calls CDtnRequestHandler::RequestReadData method in the active object. Then, RDtnSession::RequestReadData is called from the active object, and the active object's iStatus property is passed to it. Also, a reference for a descriptor (Symbian-specific data structure to store data and strings), where the received data is to be saved, is passed to the session. The TIpcArgs object is passed to the SendReceive method that sends a request to the server including the reference to the descriptor. It is an asynchronous method that returns immediately. Then, the SetActive method is called, to indicate that the active object has issued a request that is now pending. CDtnRequestHandler::RunL method is called by the framework after the server has completed the request.

In Figure 11, the ServiceL method is called by the client-server framework after the client has called SendReceive method. Then, RMessage2::Function is called to determine the request type and the request (RMessage2) is saved. If the request is for reading the received application data, server's GetBpaEngine method returns the bundle protocol agent, and the application data is read in to the descriptor. This descriptor is then written to the address, that was received in the request object, in WriteL method. After this, the server side session calls Complete method for the request to indicate that the request has been fulfilled.

In Figure 12, the active scheduler (in the active object framework) calls the RunL method in response to the server completing the request. The object that implements the observer interface is notified in the delay-tolerant application and the screen of the device may be updated to notify the user that the application data unit has been received. The observer can implement also other operations after this event

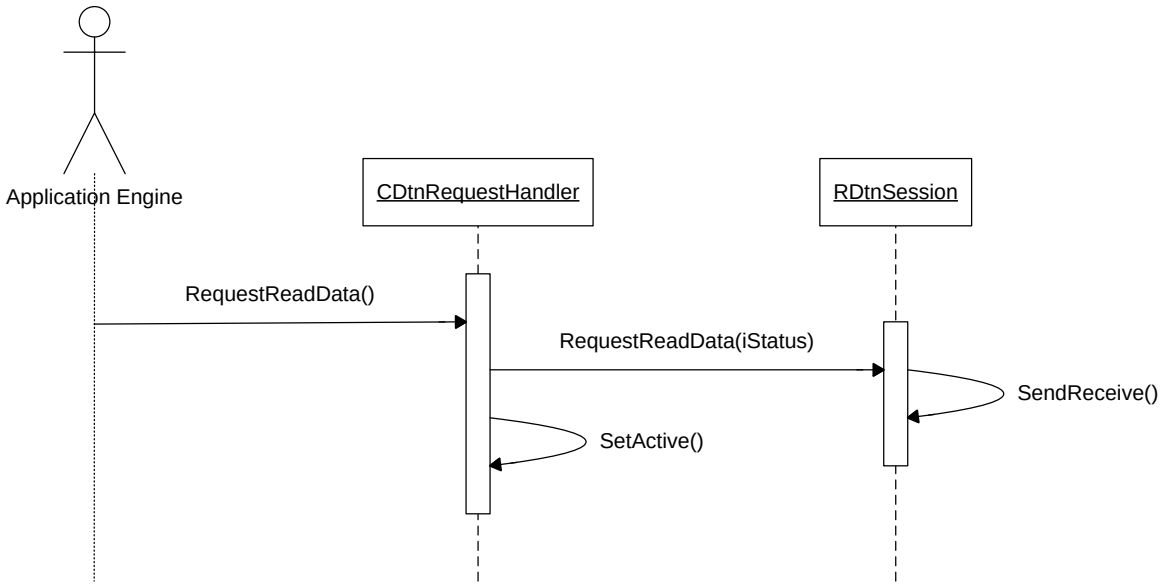

Figure 10: The sequence involved in making a request to the server

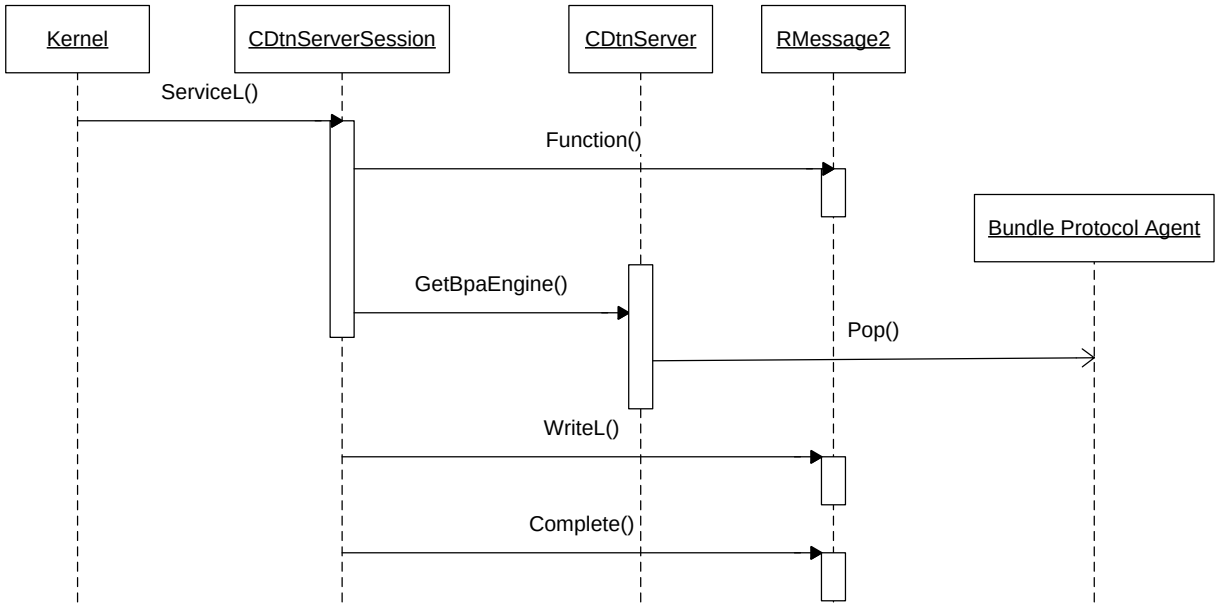

Figure 11: Server-side processing in making a request

signal is initiated by implementing an event chain. Then, the sequence flow has been completed and the client calls RequestReadDataNotification again, and the sequence starts from the beginning.

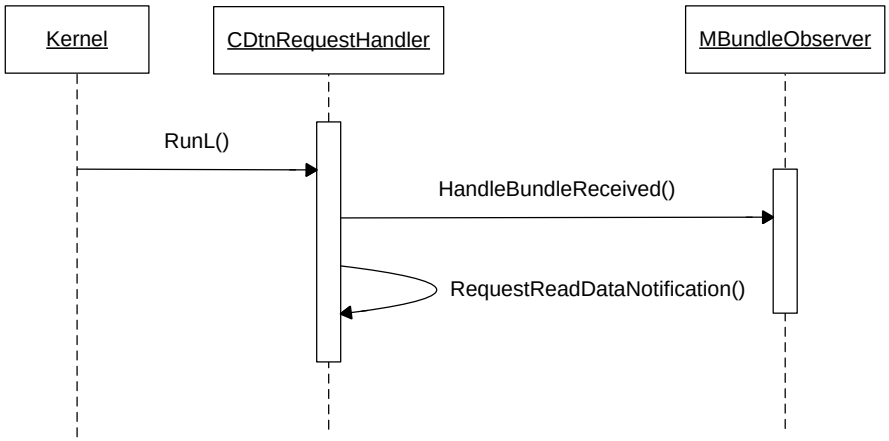

Figure 12: After the response has been received at the client

Canceling DTNServer request

In Figure 13, CancelRequestReadData is called in response to the user initiating to stop receiving application data, for example, when quitting the application. The active object then calls CActive::Cancel, that calls (from the framework) the implemented DoCancel method. This method instructs the session to cancel the pending request by calling RDtnSession::CancelReadDataRequest method. The synchronous request is now sent to the server, instructing it to cancel the pending request by calling SendReceive.

In Figure 14, the ServiceL method is called by the framework after the client has made the SendReceive call. The request is identified again by calling the RMessage2::Function method. In this case, it indicated the the client wants to cancel an outstanding request. The client waits for the completion (because of the synchronous method call) so the request should complete in as short time as possible. The server side session calls Complete method for the pending request (RMessage2), that was stored to the server when making the original request. The server calls also Complete method for the current cancel request. The cancellation is now complete and the client can continue processing, e.g., exit the application cleanly. It should be noted here that the bundle protocol agent still holds a received bundle, and we rely on the timeout mechanisms to cancel it in the bundle protocol agent component, if we don't want to receive it anymore. Another way to implement this is to continue cancellation chain further to the bundle protocol agent component.

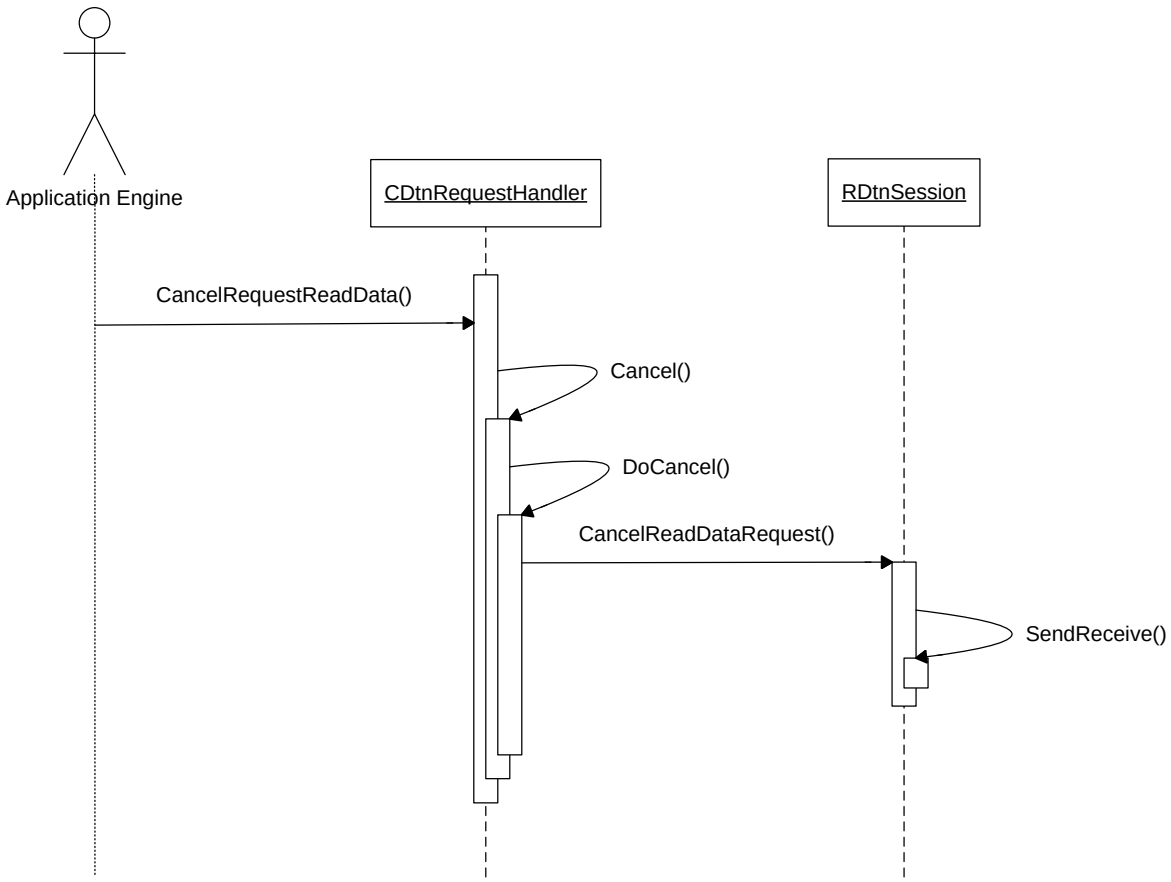

Figure 13: The sequence involved in making a cancel request to the server

4.2.5 Tools for testing

In addition to DT-Talkie, a delay-tolerant application described in the case study later, we have developed three more basic delay-tolerant tools for the DTNS60 bundle node implementation, that use the previously described DTN API. The motivation for these tools is to prove the interoperability functionality on a real device. The Linux-specific implementations of these can are included in the reference implementation [18].

These three has been implemented using only the GUI-based application component, and the architecture is similar for all three applications. These applications do not provide more advanced functionality as DT-Talkie at the application level, and therefore the main functions beneath the UI layer are not separated to the different library components. If these tools are ported to differing screen devices, or to use the Qt layer, then the engine objects of the tools can be still reused. These tools

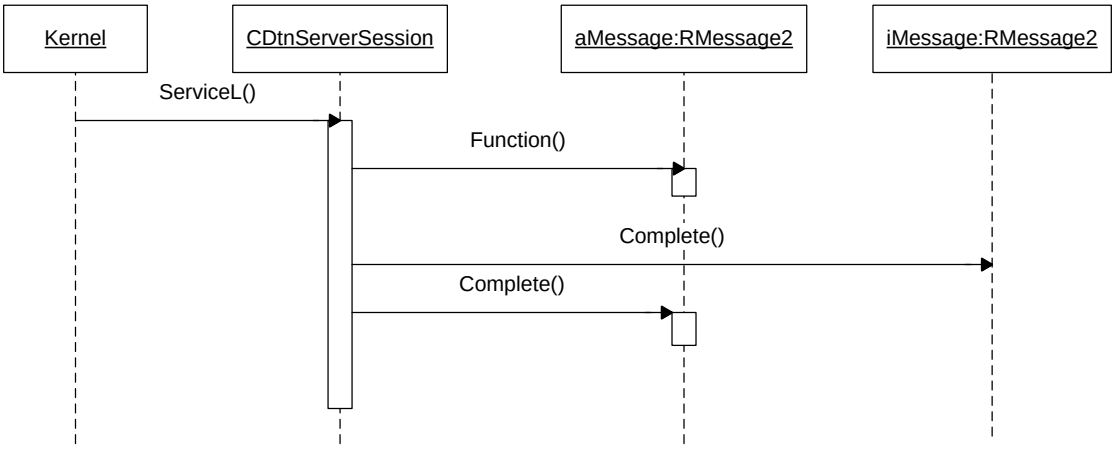

Figure 14: Server-side processing in cancelling a request

are used later in the testing scenarios for sending, receiving and forwarding bundles, and pinging other bundle nodes.

DTNReceive

DTNReceive is an application that receives bundle payload data for a particular end-point identifier and stores it to the file system of the mobile phone. The files to be stored are only dumped to the file, so to identify the filetype one needs to investigate the file more thoroughly outside the bundle node implementation.

DTNSend

DTNSend is an application that sends a file from the Symbian phone to the other remote bundle node identified by a particular end-point identifier using the bundle protocol. The file to be sent can be selected freely from the file system of the mobile phone.

DTNPing

DTNPing sends ping messages in bundles, that basically are echoed back from the DTN2 implementation's server process component that we use in the interoperability tests. In addition to the destination EID, the values of ping bundle count, interval sending ping bundles and expiration of a ping bundle are configurable values.

4.3 Summary

In this chapter we introduced the bundle node implementation for the S60 smartphones and the legacy for this implementation. We looked into the design, the architecture and the functionality of the re-engineered solution. The common API for the bundle protocol service was introduced using the sequence diagrams to show the basic usage. Three tools, that offer the basic interoperability functions for DTNS60, were also introduced. Next, the implementation is validated, evaluated and discussed about the future development ideas for the software. Then, we describe a more advanced DT-Talkie application that has been developed to use the bundle protocol service offered by DTNS60.

5 Validation and Evaluation

In this chapter, we first verify the implementation against the requirements specifications. Then, we stress the DTNS60 bundle node implementation for various kind of measurements and tests. For measuring the resource consumption, we used Nokia Energy Profiler introduced in the chapter 3. We measure also the throughput capabilities of DTNS60. Then, the interoperability of the implementation is evaluated with the other implementation on an another device being the DTN2 reference implementation on a laptop computer. In these interoperability tests and throughput measurements we use DTNPing, DTNSend and DTNRecv tools that can be found on both implementations (correspondingly, these tools are named dtnsend, dtnrecv and dtnping in the reference implementation). The future development ideas for the software suite are discussed in the end of the chapter.

5.1 Verification

We verify the implementation against the requirements specifications, and describe missing features in the current release, DTNS60 0.4.1. The current release is an alpha release and is by no means a comprehensive implementation for the real world deployments. Only "just enough requirements" are implemented to stress the implementation to the interoperability tests and study a minimal bundle forwarder characteristics. Our goal is also to implement extensions for the experimental bundle protocol. A detailed list of the features against the requirement specifications [62, 15] is depicted in the Appendix A. For curiosity, we depict also sloccount[75] statistics in the Appendix B (the test framework and the duplicated code in the legacy application agent have been excluded from stats of the current release; and the code of the tools and DT-Talkie is included).

5.2 Throughput capabilities

In this section we stress the implementation and study the throughput capabilities of DTNS60. We send three variable-sized messages from the reference implementation to the Symbian implementation, and measure the received throughput at the Symbian-side. In the sender-side, the sending rate is increased using dtnsend application of the reference implementation. In the receiver-side the DTNRecv application is modified slightly to log the receiving times of the application data, that

is, a time when the application agent passes the message to the delay-tolerant application. The clocks have been synced between the two devices manually. WLAN in the infrastructure mode is used, and the sizes of the messages are 1KB, 64KB and 1MB (maximum message size supported in this release). The results are plotted in Figure 15.

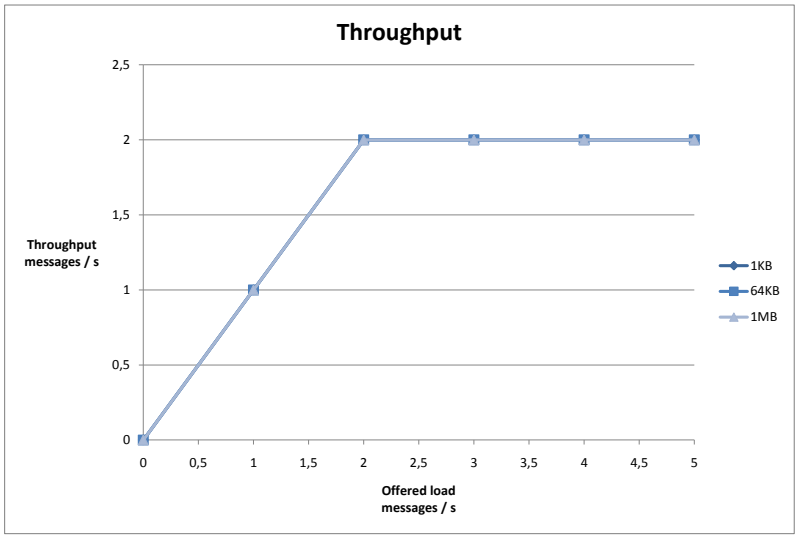

Figure 15: Throughput at the receiver DTNS60 bundle node

It can be seen from Figure 15 that the message size does not matter; instead the current release of the implementation has a characteristic in the main loop, that it can handle only two messages per second stably. However, we observed that this release becomes somewhat unstable when sending 1MB messages due to the memory leak defect in this release. The rate of 2 messages/second is the bottleneck of the current release of DTNS60 that may be modified using the different polling period in the implementation's main loop. On the other hand, changing the polling period smaller, drains the battery more quickly since it prevents the system from going to the low-power mode, so there is a trade-off between these two characteristics. The polling period should be chosen in the future releases depending on the target deployment scenarios.

In practice, two messages per second is a reasonable throughput rate especially in the environments where the contact opportunities are scarce. For example, when

sending a large MP3-based music file sized 5-10MB (usually), it would take only a few seconds to be transmitted. Drawback for this release is, that this larger music file need to be split into the smaller messages in the sender side and reconstruct on the receiver side. Currently, this fragmentation and reassembly mechanism must be implemented at the delay-tolerant application level. Moreover, if the phone communicates with multiple devices simultaneously, the throughput rate becomes a more severe bottleneck (for example, in the urban areas where contact opportunities are frequent). The current release of the implementation operates more stably if transmitting small files sized only a few kilobytes, for example, messages including only text.

5.3 Resource consumption

Resources are scarce in a mobile device compared to the desktop devices. Therefore, the resource consumption should be minimized on a mobile phone. For measuring the resource consumption Nokia Energy Profiler is used.

We provide an example scenario where the CPU load and the energy usage of the device over a measurement period are measured. DTNS60 is utilized in this scenario for the bundle forwarding purposes. The results are phone-dependent (it is not in the scope of the thesis to test with a large set of the Symbian phones), and can not be generalized, but the results give some direction how the current release behaves in a somewhat modern Symbian smartphone.

The measurements have been created using Nokia N95 8GB mobile phone as a bundle forwarder. 1KB messages encapsulated as bundles are sent between two DTN2 processes through the DTNS60 forwarder. The message sending rate is 1 bundle per second for a duration of 68 seconds. The battery capacity of the phone is 1267mAh, and the firmware version is 31.0.015. WLAN in the infrastructure mode is utilized. DTNRecv application is the only application process running on a phone in addition to NEP and default operating system services.

Figure 16 depicts the CPU load. We can see that the load is over 80% when the bundle forwarding is ongoing. In Figure 17 the cumulative energy usage is shown over the same measurement period. It can be seen that the CPU cycles of the implementation should be minimized to not drain the battery eagerly. If the forwarding operation would be continuous, the battery would be drained in 5-6 hours using the characteristics of the scenario. Indeed, the current release is not capable for the real

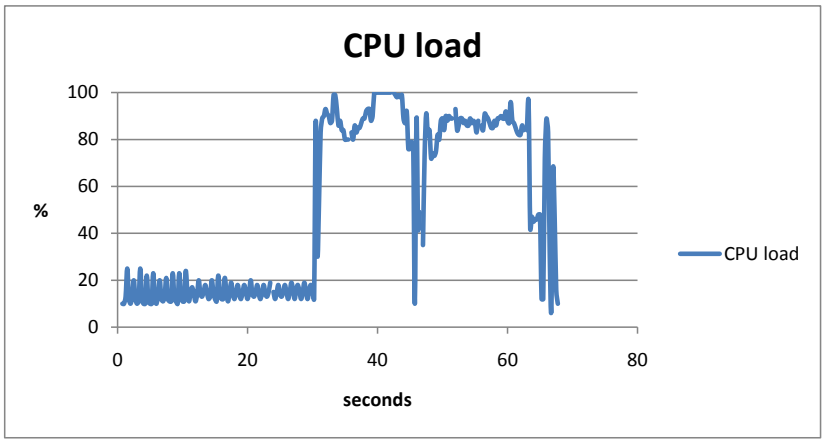

Figure 16: The CPU load of DTNS60

world deployments if the bundles are forwarded with this rate, or even more loosely. This also introduces the security question, that is, unwanted traffic should not be accepted to be forwarded in the resource-constrained devices because there is cost with the resource consumption.

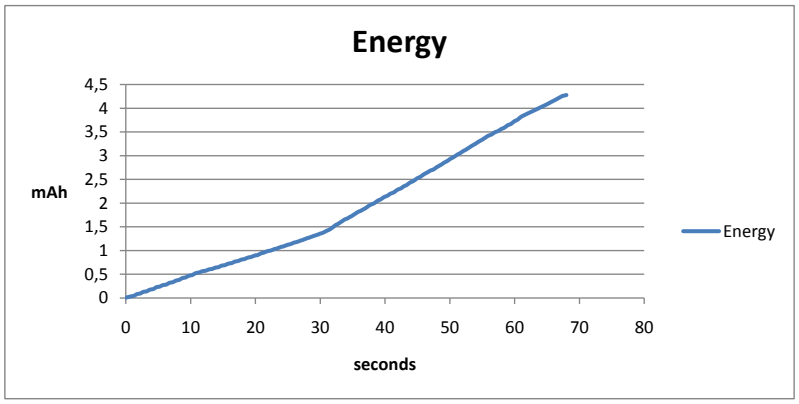

Figure 17: The energy consumption of DTNS60

5.4 Interoperability

In the interoperability tests we stress the DTNS60 implementation with bundle sending, receiving and forwarding tests. The other implementation, that is used in the tests, is the DTN2 reference implementation. We use the static pre-configured bundle forwarding between the bundle nodes when the multihop testing is utilized, and also static direct routes are configured in the single hop testing. In addition, the interoperability between different devices running only DTNS60 has been also tested. The successful testing scenarios are depicted in Figure 18. These scenarios show the simplest interoperation scenarios, that can be extended to support more than two hops (using static configurations), and also taking in use other devices that are able to run DTN2, such as, internet tablets. In addition, we are confident, that the bundle protocol implementation for Symbian S60 devices is also able to interoperate with a wider set of implementations, because it interoperates with the reference implementation (using TCP CL). DTNS60 was utilized also in the DTNRG interoperability testing event [16].

5.5 Further development

In addition to get all the requirements implemented from the specifications, there are two main paths to continue the development of the delay-tolerant software for the Symbian platform. First one is to merge DTNS60 with the Qt application and GUI framework, when the same source code can be used across embedded Linux, Mac OS X, Windows, Linux/X11, Windows CE, S60 and Maemo platforms. This ”code once, deploy everywhere” vision [53] offers a cross-platform support, eases the maintainability and shortens the development time of the delay-tolerant software tremendously. The whole Symbian platform is going to be merged with Qt in the future [65], so merging makes sense. We suggest to continue this path. This path enables the evolution of the delay-tolerant software for Symbian mobile phones also for the upcoming mobile phones. More features of Qt are introduced in a while.

The other main path is to port the DTN2 reference implementation for Symbian smartphones using Open C/C++ libraries (on top of C/C++ compatibility layer). This is, in theory, feasible because of the ever-increasing processing and memory power that Symbian smartphones offer for the end-users. The porting process may require implementing some parts from scratch, and requires further research. Furthermore, DTN2 has suffered from ignoring the “keep it simple” -principle, and may

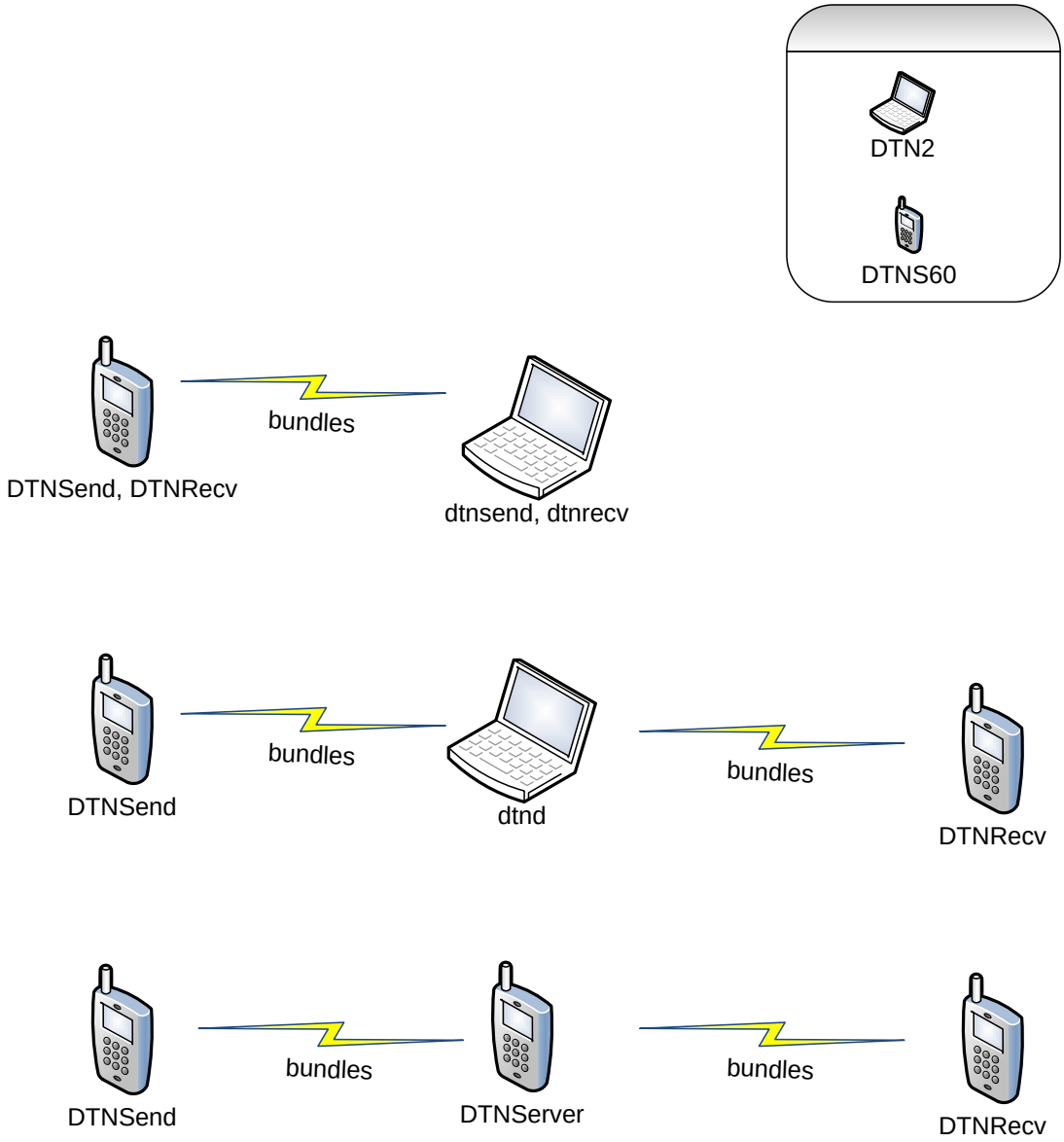

Figure 18: Interoperability between DTNS60 and DTN2

not be the only implementation for the mobile device environment for which everyone relies anymore.

Other alternatives for the implementation techniques are, for example, using PyS60, .NET for S60, Java ME or other run-time plug-ins that are offered by the platform [65]. These runtimes enable the possibility to implement a bundle protocol service implementation for the Symbian smartphones. Drawback for using these runtimes, is that the access to the lower level service APIs would be restricted, and therefore,

access to those services would provided by the native Symbian APIs anyway. There are many alternative techniques to choose from, and the goal of the deployment scenarios, and how robust and efficient the implementation should be, would act as a requirement for choosing the right technique. For example, if one wants to create a rapid prototype for research purposes, then PyS60 would be a convenient choice since it offers means to implement prototypes rapidly, but a more robust solution would be provided by the native Symbian. Implementing a novel bundle node implementation(s) from scratch (when the legacy codebase is not a burden as in DTNS60) using, for example, Qt, and native APIs when necessary, and to compare, verify and test different implementation architectures is an interesting research area to look for.

In the current release of DTNS60, the integration of the Symbian OS Unit framework has been done, but the coverage of the tests is still very minimal. Figure 19. depicts the coverage of the tests that is our goal for the delay-tolerant software in the bundle node. An another integration suggestion to DTNS60 is Nokia Energy Profiler, that can be used to provide more efficient and robust operations for the resources management in the mobile device at runtime. Using NEP APIs the total operation time of the bundle node can be increased if the battery-charging possibilities are limited (as in the challenged and scarce-resourced environments). Moreover, because the DTN operations are basing on the custodians, the bundle nodes do a lot of additive processing and data persistence in their logic that need to be minimized (using NEP for real-time measurements).

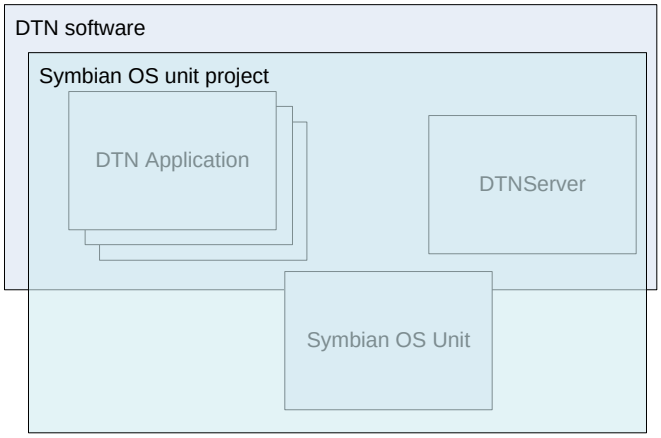

Figure 19: Test framework for DTNS60

To continue the evolution of DTNS60 in the future, we introduce features of the

Qt for Symbian application and GUI framework little bit more detail. It provides a technical framework for the Symbian-based mobile phones to take in use. We believe that merging the current version of DTNS60 with Qt is the most sensible path to continue development.

As said, using the Qt framework, support for the cross-platform deployment of the applications for different platforms using the same source code has, been enabled. People with desktop application coding skills can seamlessly move into the mobile and delay-tolerant software development. This would also reduce the development time drastically when multiple platforms are targeted, and would be especially convenient for the research purposes, when evaluating the bundle protocol functionality in the heterogeneous devices and networks. After all, the deployment goal for the full-fledged bundle protocol is cross-platform.

Also, there are minimal reasons to develop a novel (delay-tolerant) software for the mobile phones that are going to be replaced in the very near future. On the other hand this is also a problem, because the old mobile phones are not going be replaced all of a sudden, so a lot of legacy software and mobile phones are going to be used still an unknown time, and requires both legacy and modern implementations.

The Qt GUI module [54] is going to replace somewhat complex way of doing user interfaces with the Avkon GUI framework. Consequently, current example delay-tolerant applications in DTNS60 with graphical user interfaces will not work anymore in the upcoming phones, but on contrary, the application logic can be ported straightforwardly. In other words, there will be a binary break when Qt replaces Avkon, and the old binaries need to be ported to the new devices at least in the UI layer.

Qt provides a lot of APIs (in addition to GUI APIs) [54], e.g. for networking, multimedia and persistence capabilities that would be enough for implementing a bundle node. A general overview of the API usage of the delay-tolerant applications for Symbian using Qt is depicted in Figure 20.

Qt labs [55] has also an interesting research project called Qt mobility [56]. This project researches how the cross-platform mobility capabilities can be developed at once with Qt. For example, the bearer management API in the Qt mobility APIs provides means for choosing the most convenient network interface optimizing speed and cost of the underlying network connection in a cross-platform fashion. Qt mobility provides also interesting APIs today to be used for operations, such as, location management, contact management and multimedia operations, that can

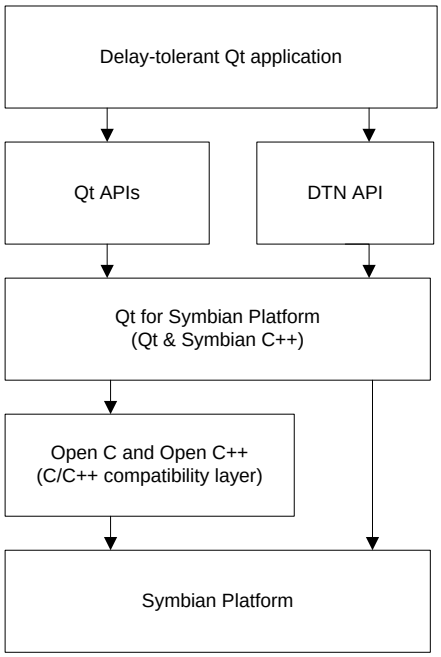

Figure 20: Merging to Qt

leverage the delay-tolerant application development.

The development of Symbian software is well-known to be complex compared, for example, to Java. The learning curve of Symbian programming is steep. Qt eases the development effort tremendously compared with the native Symbian development since Qt development is known to be intuitive and developer-friendly. Merging to Qt would also lower the barrier to grasp into the DTNS60 project, and get more contribution from the open source community. There is no need to learn yet another programming language if one can already practice Qt development in the desktop environment. The transformation from the desktop software development to the mobile and delay-tolerant software development would be seamless.

If the delay-tolerant applications are ever to be commercialized, then the applications' user interface should also be attractive, intuitive, and give a modern look and feel for the end users. This would also favor Qt for Symbian, that provides appealing user interface components for the delay-tolerant application development.

Also, if everything can not be implemented using Qt, it can always be skipped and develop parts, or components, with native Symbian, and integrate these low-level components with the main implementation that supposedly uses Qt. Symbian and

Qt can work seamlessly in conjunction (Qt being an overlay layer for the Symbian APIs).

Lastly, we suggest a roadmap for DTNS60 that is depicted in Figure 21. The roadmap gives some direction and is by no means strict. The development should be steered to the direction that serves the overall goals of the mobile DTN testbed and research purposes.

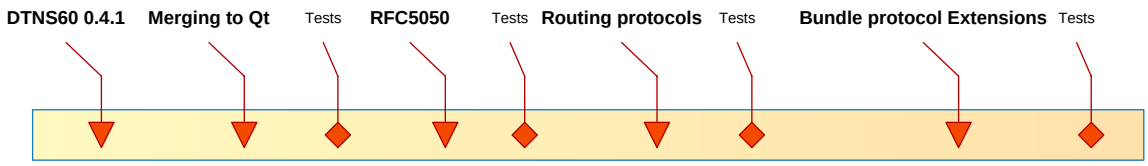

Figure 21: Roadmap

5.6 Summary

In this chapter, we measured and tested DTNS60 implementation. Different parameters of the resource consumption were measured and looked how the energy consumption of DTNS60 affects to the battery life. In the interoperability testing part, we tested interoperability with the simplest case scenarios that can be extended to support larger scenarios. We utilized the DTNS60 bundle node to measure the receiver throughput to receive the small and large messages, and identified the bottleneck of the current release. Lastly, we looked the future development ideas, and introduced briefly the features of Qt that can be merged to DTNS60, and furthermore, continue the evolution of the delay-tolerant software on Symbian. In the next chapter, we study DTNS60 as means of providing a more advanced end-user application for the voice communication in the challenged mobile networks.

6 Case study: Cross-platform delay-tolerant voice communication

In this chapter, we study DTNS60 as means of providing practical voice communication between humans parties in the challenged networks [32] using the devices of the two widely-spread mobile software platforms. The application used is DT-Talkie, that already has an implementation instance [31] on the Maemo platform (using Nokia N810 Internet tablets). DT-Talkie is a push-to-talk application that sends and receives voice messages encapsulated as bundles. DT-Talkie offers means for asynchronous voice messaging [27], and it is a delay-tolerant version of the traditional walkie-talkie application [47]. Moreover, the messages are encapsulated as MIME messages at the application layer to add attachments for the voice messages, such as, pictures of the users.

DT-Talkie can be utilized, for example, in the rural areas where the basic network infrastructure is lacking, or in the military battlefields among the infantry that communicate in an ad-hoc fashion. This kind of walkie-talkie style asynchronous voice communication, offered by DT-Talkie, is embraced by the DTN architecture.

We introduce a novel DT-Talkie for S60 implementation. Moreover, we describe the porting process of this delay-tolerant application to the Symbian platform, and scenarios where DT-Talkie can be utilized.

The goals of the case study are:

- to implement a delay-tolerant voice application that uses the bundle protocol service offered by DTNS60;
- to deliver voice messages, encapsulated by the bundle protocol and the TCP CL protocol, between two platform-specific instances of the delay-tolerant voice application;
- to gain experience about porting an application from the Linux platform to the Symbian platform;
- to discuss the voice capabilities of the Symbian phones; and
- to provide a larger scenario for a delay-tolerant application for terrestrial mobile delay-tolerant networks.

6.1 DT-Talkie for S60

In this section we introduce the design, the architecture and the functionality of the novel Symbian-specific DT-Talkie implementation. Also, the porting process is described, and further development ideas are discussed.

6.1.1 Design considerations

The most important design patterns are the MVC, the event mixin and the active objects. The MVC is used for the application structure conforming the Avkon GUI framework. The event mixin is used for handling application data unit receiving and sending functions asynchronously to the lower layers to the bundle protocol service. We use the DTN API for sending and receiving voice messages to and from the other bundle node. Also, the active objects are used in conjunction with event-driven programming to enable concurrent processing, i.e., creating the application more responsive.

The user interface is minimalistic, and configurations are loaded at start time from the configuration files. We want to create a proof-of-concept prototype, but at the same time, enable further development.

6.1.2 Architecture

We have used as a good practice to separate the GUI and the engine part of the application to enable different phone UIs to use same the engine, as in Figure 22. The engine component uses the bundle protocol service. When the application is ported in the future, for example, for the touchscreen phones or using Qt framework in the UI layer, only the UI component need to be re-implemented. Furthermore, the development of the engine and the GUI components can be done separately since the coupling is loosed. In this version of DT-Talkie, we have used the native Avkon GUI framework for Symbian mobile phones.

6.1.3 Porting process

There are few delay-tolerant applications written in ANSI C (e.g. contained in DTN2) that would be also good example applications to function using the bundle protocol services offered by DTNS60. Therefore, we introduce the porting process of DT-Talkie from the Linux-based platform to the Symbian platform. The process can

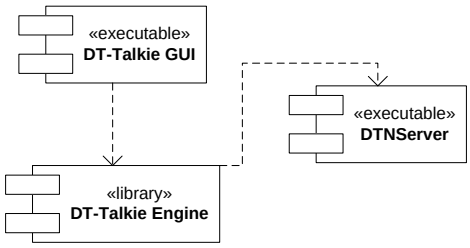

Figure 22: The component model of DT-Talkie for S60

be thought also more generically for porting traditional applications to the Symbian platform.

DT-Talkie was originally developed for the Maemo environment using gmime[23] and gstreamer[24] 3rd party libraries in conjunction with the GTK+ framework [72] that is comparable to the Qt framework. We describe briefly, how the porting process from the Maemo platform to the Symbian platform should work, and how it in practice turned out. As of our knowledge, this is a hard problem in the mobile software development community in general.

The original plan was to port DT-Talkie rapidly for the Symbian platform. The release of P.I.P.S.[74] and Open C/C++ libraries [46] has helped the migration of desktop and Linux-based applications to Symbian OS. These provide many standard C libraries that are commonly used in the development of the Linux applications. Therefore, already developed programs for Linux are, in theory, portable to Symbian OS, and even the porting without changes may be possible. However, P.I.P.S. and Open C are not fully POSIX compliant [74], but the main functions are supported.

We thought that using a well-defined process [76] for porting the application would ease the task. The porting process of DT-Talkie was applied using the following non-linear steps, and related actions described (if the step is not self-evident):

Choose a project to port The chosen project was DT-Talkie.

Analyze the code The user interface was certainly needed to be re-written from scratch due to differing UI frameworks and the difference between the Maemo touchscreen and the Symbian non-touchscreen devices. Also, the original DT-Talkie had 3rd party libraries (gmime and gstreamer).

Re-architect The user interface and the application logic were separated to different components, that are, GUI executable component and dynamically linked library containing the application logic

Set up the development environment Moving from the Linux-based Scratchbox[63] environment to the Windows-based Carbide C++ environment [9].

Integrate project with build system Using Open C/C++ and basic glib libraries.

Compile

Fix problems

Run and test

Debug

Re-integrate with the original code, if desirable

After many iterations, the porting of the DT-talkie application from the Internet Maemo platform to the Symbian platform turned out to be, in the scope of the thesis, impossible and the development needed to be done from scratch to the native Symbian environment. The biggest problem was that there were no support for the 3rd party libraries, i.e., the libraries would have needed to be ported also for Symbian. Even though there exists glib libraries for Symbian, the other two libraries were the bottleneck. Furthermore, the differing audio codec capabilities between Maemo devices and Symbian devices raised problems, and we needed to add the basic PCM audio codec to the Maemo implementation to make these two platform-specific implementations interoperate in a simple fashion providing proof-of-concept operations.

6.1.4 Functionality

Figure 23: The user interface of DT-Talkie for S60

DT-Talkie application consists of the three separate navigable views. The user interface of the DT-Talkie for S60 is depicted in Figure 23. We have used the Symbian-specific view-switching architecture for implementing this multi-view application. The design decision to split different UI areas to different views, that differs from original Maemo version, is because of the mobile phone's screen is smaller than in the Internet tablet, and our target phones does not have touchscreen that would enable different selectable UI components to be more convenient as in the Maemo environment.

The settings for the DT-Talkie application can be configured in the configuration files where one can configure application-specific and bundle protocol agent -specific configurations.

The main view shows current information of the state of the DT-Talkie. The number of incoming and the number of outgoing voice messages are shown at the top of the view. Also, the active user contact and the local user and their end-point identifiers and IP addresses are shown in the mainview. The active contact is an endpoint identifier with whom we are interested to communicate with. There is a logging component that shows useful information for the local user about the state of DT-Talkie. This log is deleted after the process dies, but a more comprehensive log can be found in the phone's file system that provides more low-level information for the developers persistently.

The pending list shows incoming and outgoing voice messages in a list. The incoming voice messages are shown at the top of the list, and after those the outgoing voice messages, that has not been sent yet to the delay-tolerant network, are shown. The different icon type separates these two types of voice messages. Each voice message contains information about the destination IP address and the end-point identifier. The name of the message is a timestamp from the mobile phone's system information. The user can at anytime listen or delete these application-local voice messages.

The contact list shows a list of contacts that are in the DT-Talkie's persistent memory. We can use ad-hoc peer discovery option in the menu to find more contacts from the currently connected network in the future releases. Also, if we get a new voice message from the sender that is not yet a contact in our local store, a new contact is added automatically, and also changed as an active user at the same time. The local user can also add manually contacts to the persistent memory by adding their end-point identifier and IP address with port number manually. Every contact has

been associated with a picture, that may have been received with a voice message. The picture is shown as an icon beside the contact 's end-point identifier.

The main steps for a sender and a receiver are listed below.

Sender:

1. Digitizing analog speech (record)
2. Encapsulate digitized speech as a MIME message
3. Pass the MIME message to the DTNServer
4. Encapsulate the MIME message as a bundle
5. Pass the bundle from the BPA to the TCP CLA
6. Send the bundle to the delay-tolerant network

Receiver:

1. Receive the bundle from the delay-tolerant network
2. Pass the bundle from TCP CLA to the BPA
3. Decapsulate the MIME message from the bundle at DTNServer
4. Pass the MIME message from the DTNServer to DT-Talkie
5. Decapsulate the MIME message
6. Convert digital speech to an analog signal (playback)

6.1.5 Further development

In the next release of DT-Talkie, we suggest that at least the UI layer is ported to use the Qt framework. Also, the application logic is portable and requires further investigation, if it is ported to use the Qt layer (enabling cross-platform support), for example, by using an adapter between the Qt and native Symbian implementations. We could also only link the Symbian-specific application logic to the Qt-based application GUI process.

Further investigation is needed to support more audio codecs: to use those included in the Qt mobility project, or to use the lower level codecs offered by the Symbian

multimedia framework. A set of heterogeneous mobile devices have varying audio codec capabilities, and therefore, also the mechanism to negotiate the used audio codec between the two devices is a convenient feature to implement for the next release. This is especially required when the peer discovery has been taken in use, when we don't know beforehand what devices we are going to discover.

We want also DT-Talkie to be integrated to the native contact management of the mobile device, so the end-user does not need to manage duplicate contacts, for example, between telephony services and the delay-tolerant voice communication services.

User interface design need to be enhanced since the current version only provides the simplistic user interface that is not very usable outside the research purposes. The user interface should be more intuitive.

6.2 Point-to-point voice communication

In the simplest scenario, we utilize two devices, where other one is running DT-Talkie for S60 using the bundle protocol service (DTN API) offered by DTNS60. The other one is running the DTN2 reference implementation, and Linux-based DT-Talkie implementation. Figure 24 depicts this scenario. Two persons can now communicate in an asynchronous manner with intermittent connectivity. For example, when there are disruptions of the communication links, the message gets delivered anyhow, compared to the traditional approach when the disruption hangups the current communication session. It should be noted that, if the delays grow too large, the voice communication is not anymore conversational, instead the communication approaches similar behavior as in the voicemail applications.

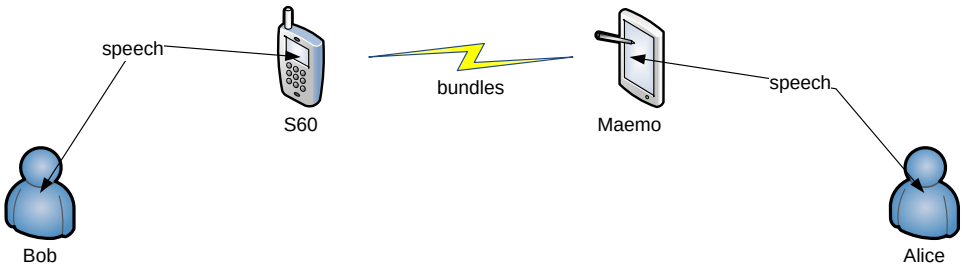

Figure 24: Scenario with point-to-point communication

6.3 Multihop voice communication

The multihop use case scenario utilizes more than one hops in between two humans, and can potentially be extended to a N-hop delay-tolerant network. The scenario is depicted in Figure 25. There are one or more bundle forwarders in between a sender and a recipient. The bundle forwarders function with the store-carry-and-forward manner. We use also a linux bundle forwarder that can operate as a voice message throwbox (running DTN2). Users can leave, for example, voice messages in the rural areas to provide information about the surrounding environment for other users that discover later the same place where this throwbox lies. An analogy to the postal service serves well when the throwbox is in use - the traditional letters are replaced by the digitized voice messages, and the throwbox is conceptually same as a mailbox.

The lifetime of the bundles and the time synchronization between the devices is a research problem to be solved in this kind of scenarios. The deployment scenario will serve as an requirement here. For example, in the military applications lifetime of the bundles is significantly smaller than in the rural areas where data mules, such as helicopters or buses, bring messages, like voicemails or news as podcasts, to the village out of reach of the Internet and other networks. Moreover, if the devices' clocks are out-of-sync, the bundles may be dropped in an erroneous fashion.

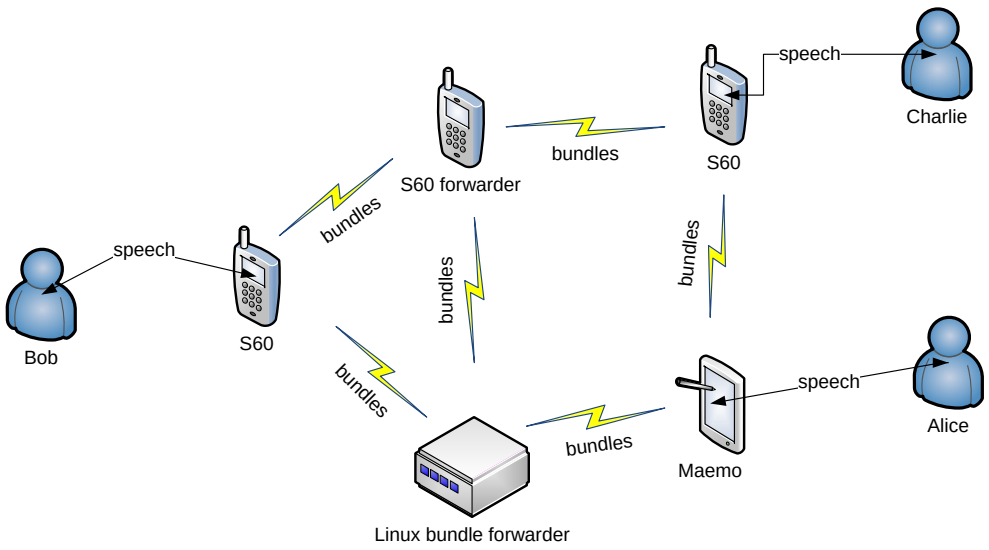

Figure 25: Scenario with bundle forwarders

6.4 Conclusions

In this chapter, we introduced the DT-Talkie application using the bundle protocol service offered by the DTNS60 bundle node implementation. The design, the architecture and the functionality of DT-Talkie for S60 were described. We discussed also about the porting process, and reasoned why it needed to be developed from scratch due to the incompatibilities of the different 3rd party libraries. In the end, the real world deployment scenarios were provided.

The user interface design for the delay-tolerant applications is more complex than for the traditional applications. On the other hand, the client-server based communication between DT-Talkie and the bundle protocol service is a convenient design decision, when implementing delay-tolerant applications for the Symbian mobile phones. The client-server framework enables also multitasking of the different delay-tolerant applications that can be run at the same time. We should use more native features of the Symbian phones, such as, contact management API. Also, an audio codec negotiation protocol is a feature to be investigated in more detail in the future that enables delay-tolerant voice communication between heterogeneous mobile devices.

7 Lessons learned and conclusions

In this chapter, we describe the lessons learned and make conclusions about implementing a bundle node for the mobile devices using the Symbian mobile software platform. We verify the thesis against the goals described in the introduction chapter. We discuss also about the robustness of the current release, implementation experiences, and future development paths of the proposed implementation. Finally, we look future research ideas where delay-tolerant networking for the Symbian platform can be applied.

7.1 Conclusions

We started with conducting a literature study for the requirements for the bundle node, and described the technologies for the Symbian mobile software platform that are required to implement the bundle node. Then we proposed the novel DTNS60 bundle node implementation for Symbian S60 mobile phones, and evaluated it's functionality, interoperability, throughput and resource consumption capabilities. Finally, DTNS60 was applied in the form of application case study where the implementation was used in the more advanced multi-platform environment, and developed a delay-tolerant voice application using the DTN API provided by the bundle protocol service of DTNS60.

Implementing a bundle node for Symbian is a complex software development project. In the scope of the thesis, only a prototype can be developed and basic proof-of-concept scenarios demonstrated in the laboratory environment. To implement a more robust solution, more resources (time, developers and testers) are needed, for example, from the open source community. The current release of the DTNS60 implementation is not ready for real-world deployments being unstable and not using resources of the device efficiently. Therefore, we have released the implementation for a shared and public repository [17] in hope of attaining contribution to the development. The current version of DTNS60 is 0.4.1 at the time of writing the thesis.

We chose to continue the development of the legacy implementation instead of creating one from scratch. This is not always a good approach for creating a new design and architecture for the solution. Firstly, almost everything in the legacy codebase needed to be debugged anyway, which was a very time-consuming process. Secondly, implementing a new solution from scratch may even ease the task, and

provide a more efficient solution when the design decisions, such as using the client-server framework, can be applied right from the beginning. Indeed, when including more features, such as the bundle protocol security, into the development in the beginning of the implementation phase (if creating the solution from scratch), it is more convenient to integrate the features into the solution than adding those to the already-grown solution later.

On the other hand, the current and further enhanced implementation of the legacy implementation provides now a good groundwork for the further development. It has been developed so that every legacy part can be replaced or updated easily, if necessasary. There are parts in the legacy implementation that should be certainly reused, such as, stream-based convergence layers and bundle related classes for message formats. The investigation before replacing and updating the legacy code should be made to reduce the development effort (no need to introduce duplicate code).

Robust on-device debugging, such as using FineToothCamb[4], requires specific software and hardware that is usually available only for the device manufacturers. Instead, we chose to use a simple file-based debugging method on the device, that is slow to analyze and not feasible anymore when the solution grows (even though it can be kept beside the on-device debugging). More advanced on-device debugging methods are reasonable to take in use in next releases.

Porting the DT-Talkie application from the Linux platform to the Symbian platform introduced problems due to the incompatible 3rd party libraries. As of our knowledge, this porting problem is quite a common problem generally in the mobile software development community. The porting should be feasible in theory, but in reality eventually every part had to be made from scratch using the native Symbian platform and APIs provided by the S60 application platform. The porting required a lot of iterative development before finding out that it was not feasible (i.e. there was a need to develop a new Symbian-specific implementation from scratch). Moreover, the audio codecs used in the original DT-Talkie were not supported in Symbian without partnership access to the lower level hardware-accelerated audio services, so we needed to add more codecs also for the Maemo version to achieve interoperability between these two platform-specific implementations. The audio codec negotiation protocol should also be implemented, when the device that an other party is using, is unknown.

Delay-tolerant networking embraces asynchronous communication between applica-

tions. This kind of communication paradigm can not be applied to satisfy hard, or even soft, real-time requirements. Therefore, critical real-time applications are not sensible to be taken use in the challenged network environments. This limits the scale of applications, that can be developed on top of the bundle protocol, to applications where application messages eventually may or may not be delivered between two bundle nodes. This also affects for the voice communication when using the DT-Talkie implementations. The communication can not be conversational with the asynchronous communication paradigm, if the delays to deliver voice messages between two instances grow too large.

The current release of DTNS60 is not using resources efficiently. The resource usage can be modified by changing the polling period, which on the other hand may introduce non-responsive software, or drain the battery quickly. In a scarce environment where maximizing the battery-life is an especially important goal, the implementation should be as optimized as possible. Integrating Nokia Energy Profiler for the real-time resource consumption measurements, and modifying the operations of the implementation to work in a more resource-saving manner, is suggested.

DTNS60 is limited to receive, send or forward messages at maximum two messages per second in this release. This is due to polling period used in the main loop of the application that the whole bundle protocol service is dependent from. The maximum message size supported in this release is limited to 1MB.

On the other hand, even though these resource consumption and throughput limitations, one goal of the thesis was to made proof-of-concept tests in a laboratory environment, that were successful.

The interoperability tests with the other DTN implementations turned out to be successful way for finding defects and testing simple interoperability between the other implementations. Also, the tests provided means for debugging the software in a ways that are reasonable for intercommunication. After all, intercommunication between different bundle nodes (that use also different implementations) is the main goal in the end of the day.

Looking from the more general mobile software development point-of-view, we noticed a bunch of problems when developing first for the emulator in a desktop computer, and then afterwards deploying a build for a mobile phone. Generally, we noticed that if the implementation works on a emulator, it is not guaranteed to work on a mobile phone. For example, emulator environment may give wrong impression about the real memory available in the real device, or give a wrong set of

libraries in use that are not available on the device (e.g. SQLite libraries [59] are not available for the 3rd edition feature pack 1 devices).

It is important to choose the target mobile phones beforehand when practicing mobile software development. In theory, Symbian-based code should work in a many different models without need to change the source code. In practice the implementation must be tested with every target mobile phone, that is, not feasible within scope of the thesis. Usually, devices have, for example, different screen sizes, different supported audio codecs and different supported libraries, even though those are under the umbrella of the Symbian S60 devices.

We conclude that implementing the DTN functionality, and the bundle layer for the terrestrial mobile networks that cannot maintain end-to-end connectivity, is feasible using the traditional heteregeneous (mobile) networks and heterogeneous mobile devices. However, the thesis-related work is not fully conclusive, but offers groundwork implementation and tools for the future research.

7.2 Future research

DTNS60 is not yet fully compliant with the specifications of RFC5050; the available features are interoperable, but not all features are implemented. Therefore, the next steps are to implement the missing features, and test those thoroughly using the test framework. More advanced and appealing features to implement in the subsequent releases are routing protocols, security and other extensions for the bundle protocol. After the implementation has been tested and is in a stable state, it can be deployed into the more advanced real world scenarios, for example, in the rural areas where the basic network infrastructure is lacking and disconnections of the communications links are apparent due to mobility characteristics.

Comparing the DTNS60 bundle protocol implementation to the other implementations on the mobile software platforms should be done to choose the reasonable mobile software platform(s) to take in use in the real world scenarios. The resource consumption should be minized and throughput capabilities good enough to not introduce bottlenecks to the basic delay-tolerant communication between the delay-applications.

The deployment of DTNS60 into the real world requires compelling applications that people are interested to take in everyday use. Such applications may be, for example, delay-tolerant social networking applications [6] that can enable commu-

nication among the local communities in an ad-hoc manner. Deploying the bundle protocol to the real world scenarios can offer also new insights to the applications that are not yet known.

Hybrid solutions offer an interesting research area to implement on a real mobile phones. Those support both the traditional synchronous communication, and the DTN-based asynchronous communications when challenged environments are apparent, and choosing between these two modes has been made seamless for the end-user.

Designing delay-tolerant applications is a more complex task than designing conventional (TCP/IP-based) applications, because the response from the other end may not come immediately after the initial request has been made. Delay-tolerant user interface design offers an research area to look for. The user interface should be more intuitive for asynchronous communication. It is not an error condition if there is no connectivity available at the moment.

Also, choosing the right network interface that favors current conditions of the environment at particular time is something that is not clear to us. This service should take in account cost criterias, that may be for example, monetary cost of using the network (e.g. free WLAN vs. operator-invoiced 3g).

Implementing a peer discovery mechanisms for the bundle node excludes the problem of configuring manually the contacts in reach beforehand. For example, Bonjour[5] mechanism, that is implemented in DTN2, may be a good algorithm to use for peer discovery, but is capable to work only in the IP networks.

Time synchronization between mobile devices is a research problem that has not been solved yet for the different deployment scenarios where different accuracy in the synchronization is needed, and where the infrastructure network is not available to offer time synchronization.

The world is still waiting a delay-tolerant killer application [39] that could leverage delay-tolerant networking concept to transform in to the everyday use among the end-users using mobile devices that face disconnections and disruptions on the edges of the Internet.

References

[1] Android. `http://www.android.com/`, January 2010.

[2] L. Arantes, A. Goldman, and M. Vinicious dos Santos. Using evolving graphs to evaluate dtn routing protocols. In *Proceedings of the ExtremeCom Workshop*, August 2009.

[3] Steve Babin. *Developing Software for Symbian OS: A Beginner's Guide to Creating Symbian OS v9 Smartphone Applications in C++*. Wiley Publishing, 2008.

[4] A. Baldwin. Exploring qt performance on arm using finetoothcomb. http://labs.trolltech.com/blogs/2009/09/29/exploring-qt-performance-on-arm-using-finetoothcomb/, September 2009.

[5] Bonjour. `http://developer.apple.com/networking/bonjour/index.html`, January 2010.

[6] S. Buchegger. Delay-tolerant social networking. In *Proceedings of the ExtremeCom Workshop*, August 2009.

[7] Bytewalla. `http://www.tslab.ssvl.kth.se/csd/projects/092106/`, January 2010.

[8] Iain Campbell. *Symbian OS Communications Programming*. Wiley Publishing, 2007.

[9] Carbide Integrated Development Environment. http://forum.nokia.com, January 2010.

[10] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838 (Informational), April 2007.

[11] CHIANTI. `http://www.chianti-ict.org/`, January 2010.

[12] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3561, October 2003.

[13] D-Bus. http://www.freedesktop.org/wiki/Software/dbus, January 2010.

[14] Delay-Tolerant Networking Research Group. `http://www.dtnrg.org`, January 2010.

[15] M. Demmer and J. Ott. Delay-Tolerant Networking TCP Convergence Layer Protocol draft-irtf-dtnrg-tcp-clayer-02.txt. Internet draft, November 2008.

[16] Disconnectathon. `http://www.dtnrg.org/wiki/DtnBone/Disconnectathon`, July 2009.

[17] DTN for S60. `http://sourceforge.net/projects/dtns60/`, January 2010.

[18] DTN2 reference implementation. `http://www.dtnrg.org/wiki/Code`, January 2010.

[19] W. Eddy. Using Self-Delimiting Numeric Values in Protocols. Internet Draft, January 2010.

[20] Leigh Edwards, Richard Barker, and Staff. *Developing Series 60 Applications: A Guide for Symbian OS C++ Developers (Nokia Mobile Developer Series).* Addison-Wesley Professional, March 2004.

[21] K. Fall. A delay tolerant network architecture for challenged internets, 2003.

[22] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, illustrated edition edition, November 1994.

[23] Gmime. http://spruce.sourceforge.net/gmime/, January 2010.

[24] GStreamer. http://www.gstreamer.net/, January 2010.

[25] S. Guo, M. H. Falaki, E. A. Oliver, Ur S. Rahman, A. Seth, M. A. Zaharia, and S. Keshav. Very low-cost internet access using kiosknet. *SIGCOMM Comput. Commun. Rev.*, 37(5):95–100, October 2007.

[26] Haggle. `http://www.haggleproject.org/`, January 2010.

[27] R.J. Honicky, O. Bakr, M. Demmer, and E. Brewer. A message oriented phone system for low cost connectivity. HotNets workshop, 2007.

[28] Internet Research Task Force. `http://www.irtf.org`, January 2010.

[29] Interplanetary Internet Special Interest Group. `http://www.ipnsig.org`, January 2010.

[30] iPhone. http://www.apple.com/iphone/, January 2010.

[31] Md. T. Islam. Dt-talkie: Push-to-talk in challenged networks. ACM MobiCom, 2008.

[32] Md. T. Islam, A. Turkulainen, T. Kärkkäinen, M. Pitkänen, and J. Ott. Practical voice communication in challenged networks. In *Proceedings of the ExtremeCom Workshop*, August 2009.

[33] Adrian J. Issott. *Common Design Patterns for Symbian OS: The Foundations of Smartphone Software*. Wiley Publishing, 2008.

[34] Java ME. http://java.sun.com/javame/, January 2010.

[35] Ho-Won Jung, Seung-Gweon Kim, and Chang-Shin Chung. Measuring software product quality: a survey of iso/iec 9126. *Software, IEEE*, 21(5):88–92, 2004.

[36] H. Kruse and S. Ostermann. UDP Convergence Layers for the DTN Bundle and LTP Protocols draft-irtf-dtnrg-udp-clayer-00. Internet draft, November 2008.

[37] LiMo Foundation. http://www.limofoundation.org/, January 2010.

[38] A. Lindgren, A. Doria, E. Davies, and S. Grasic. Probabilistic Routing Protocol for Intermittently Connected Networks draft-irtf-dtnrg-prophet-02.txt. Internet draft, March 2009.

[39] Anders Lindgren and Pan Hui. The quest for a killer app for opportunistic and delay tolerant networks: (invited paper). In *CHANTS '09: Proceedings of the 4th ACM workshop on Challenged networks*, pages 59–66, New York, NY, USA, 2009. ACM.

[40] Maemo. http://maemo.org/, January 2010.

[41] O. Mukhtar. Design and Implementation of Bundle Protocol Stack for Delay-Tolerant Networking. Master's thesis, Helsinki University of Technology, 2006.

[42] Omar Mukhtar and Jörg Ott. Backup and bypass: introducing dtn-based ad-hoc networking to mobile phones. In *REALMAN '06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 107–109, New York, NY, USA, 2006. ACM.

[43] NASA - Delay-Tolerant Networking (DTN). http://www.nasa.gov/mission_pages/station/science/experiments/DTN.html, January 2010.

[44] Nokia Energy Profiler. `http://forum.nokia.com`, January 2010.

[45] Earl A. Oliver and Srinivasan Keshav. Design principles for opportunistic communication in constrained computing environments. In *WiNS-DR '08: Proceedings of the 2008 ACM workshop on Wireless networks and systems for developing regions*, pages 31–36, New York, NY, USA, 2008. ACM.

[46] Open C/C++ libraries. `http://forum.nokia.com`, January 2010.

[47] Open Mobile Alliance. Push-to-Talk over Cellular (PoC). `http://www.openmobilealliance.org/`.

[48] J. Ott and D. Kutscher. A disconnection-tolerant transport for drive-thru internet environments. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1849–1862 vol. 3, 2005.

[49] S. Perkins, E. Belding-Royer, and S. Das. Ad Hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, July 2003.

[50] A-K Pietiläinen and C. Diot. Experimenting with opportunistic networking. In *MobiArch'09: Proceedings of International Workshop on Mobility in the Evolving Internet Architecture*, June 2009.

[51] Anna K. Pietiläinen, Earl Oliver, Jason Lebrun, George Varghese, and Christophe Diot. MobiClique: middleware for mobile social networking. In *WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks*, pages 49–54, New York, NY, USA, August 2009. ACM.

[52] Roger Pressman and Roger Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Science/Engineering/Math, 6 edition, April 2004.

[53] Qt application and GUI framework. `http://qt.nokia.com/`, January 2010.

[54] Qt application and GUI framework, reference documentation. http://doc.trolltech.com/, January 2010.

[55] Qt Labs. http://labs.trolltech.com/, January 2010.

[56] Qt Mobility. http://labs.trolltech.com/page/Projects/QtMobility, January 2010.

[57] M. Ramadas, S. Burleigh, and S. Farrell. Licklider Transmission Protocol - Specification. RFC 5326, September 2008.

[58] Research in Motion. `http://www.rim.com/`, January 2010.

[59] S60 Platform SDKs for Symbian OS. `http://forum.nokia.com`, January 2010.

[60] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984.

[61] Sami Network Connectivity. `http://www.snc.sapmi.net/`, January 2010.

[62] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050, November 2007.

[63] Scratchbox. http://www.scratchbox.org/, January 2010.

[64] T. Socolofsky and C. Kale. A TCP/IP Tutorial. RFC 1180, January 1991.

[65] Symbian Foundation. `http://www.symbian.org/`, January 2010.

[66] Symbian OS Unit test framework. `http://www.symbianosunit.co.uk/`, January 2010.

[67] S. Symington. Delay-Tolerant Networking Previous Hop Insertion Block draft-irtf-dtnrg-bundle-previous-hop-block-09. Internet draft, November 2009.

[68] S. Symington. Delay-Tolerant Networking Retransmission Block draft-irtf-dtnrg-bundle-retrans-block-06. Internet draft, October 2009.

[69] S. Symington. Delay-Tolerant Networking Metadata Extension Block draft-irtf-dtnrg-bundle-metadata-block-06. Internet draft, November 2009.

[70] S. Symington, S. Farrell, H. Weiss, and P. Lovell. Bundle Security Protocol Specification Bundle Security Protocol Specification. Internet draft, October 2010.

[71] Andrew S. Tanenbaum. *Modern Operating Systems (2nd Edition) (GOAL Series)*. Prentice Hall, 2 edition, March 2001.

[72] The GTK+ Project. http://www.gtk.org/, January 2010.

[73] The Networking for Communications Challenged Communities (N4C) project. `http://www.n4c.eu/`, January 2010.

[74] Vinod Vijayarajan. A Guide to P.I.P.S. Symbian Developer Network, 2008.

[75] D. Wheeler. SLOCCount. http://www.dwheeler.com/sloccount/, January 2010.

[76] Mark Wilcox. Porting from Linux to Symbian OS. Symbian Developer Network, 2009.

[77] Windows Mobile. http://www.microsoft.com/windowsmobile/, January 2010.

[78] Windows Presentation Foundation. http://windowsclient.net/wpf/, January 2010.

Appendix A

BUNDLE PROTOCOL REQUIREMENTS		
	DASM 0.3.0	DTNS60 0.4.1
RFC 5050		
commencing a registration (registering a node in an endpoint)	not started	started
terminating a registration	not started	started
switching a registration between Active and Passive states	not started	started
transmitting a bundle to an identified bundle endpoint	done	done
canceling a transmission	not done	done
polling a registration that is in the passive state	not started	started
delivering a received bundle	done	done
bundle formats	done	done
bundle processing	started	started
administrative record processing	started	started
draft-irtf-dtnrg-tcp-clayer-02		
Encapsulation/decapsulation of bundles	done	done
Procedures for connection setup and teardown	done	done
Transmitting a bundle to the specified EID	done	done
Other features		
DTN API	not started	started
client-server architecture	not started	done
Routing	not started	not started
Security	not started	not started
Metadata	not started	not started
UPD CLA	not started	not started
Bluetooth	started	started
Persistence	not started	started
Qt layer	not started	not started

Appendix B

SLOCCOUNT Statistics

	DASM 0.3.0	DTNS60 0.4.1
Total Physical Source Lines of Code (SLOC)	10297	17325
Development Effort Estimate, Person-Years (Person-Months) (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))	2,31 (27,77)	4,00 (47,95)
Schedule Estimate, Years (Months) (Basic COCOMO model, Months = 2.5 * (person-months**0.38))	0,74 (8,84)	0,91 (10,88)
Estimated Average Number of Developers (Effort/Schedule)	3,14	4,41
Total Estimated Cost to Develop (average salary = $56,286/year, overhead = 2.40).	$ 312,599	$ 539,819