HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation

Md. Tarikul Islam

VOICE COMMUNICATION IN MOBILE DELAY-TOLERANT NETWORKS

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

Espoo November 26, 2009

Thesis supervisor:

Prof. Jörg Ott

Thesis instructor:

Prof. Jörg Ott

Author: Md. Tarikul Islam

Title: Voice Communication in Mobile Delay-Tolerant Networks

Date: November 26, 2009 Language: English Number of pages: 9+83

Faculty: Faculty of Electronics, Communications and Automation

Professorship: Networking Technology Code: S-38

Supervisor: Prof. Jörg Ott

Instructor: Prof. Jörg Ott

Push-to-talk (PTT) is one class of voice communication system generally employed in cellular phone services. Today's PTT services mainly rely on infrastructure and require stable end-to-end path for successful communication. But users with PTT enabled mobile devices may travel in challenged environments where infrastructure is not available or end-to-end path is highly unreliable. In such cases those PTT services may exhibit poor performance or may even fail completely. Even though some existing PTT solutions allow users to communicate in an ad-hoc fashion, they need sufficient node density to establish end-to-end path and eventually fail to communicate in sparse mobile ad-hoc environments. Delay-Tolerant Networking (DTN) is an emerging research area that addresses the communication requirements specific to challenged networks.

In this thesis we develop a voice communication system (DT-Talkie) which enables both individual and group users to communicate over infrastructure-less and challenged networks in the walkie-talkie fashion. The DTN concept of asynchronous message forwarding is applied to the DT-Talkie in order to transmit voice messages reliably. We employ variable-length fragmentation mechanism in the application layer with the vision to speed-up session interactivity in stable scenarios. Some approaches to resolve codec interoperability issues are implied in this thesis.

To validate the concepts of the DT-Talkie, we implement an application for Maemo based Nokia Internet Tablets, leveraging the DTN reference implementation developed in the DTN Research Group. Moreover in this thesis we evaluate the performance of the DT-Talkie through conducting a set of simulations using several DTN routing protocols and using different mobility models.

Keywords: DTN, PTT, bundle protocol, asynchronous voice communication

Acknowledgments

This thesis has been accomplished in the Department of Communications and Networking of Helsinki University of Technology as a part of a CHIANTI project funded under Seventh Framework Programme of European Union.

I would like to express my heartiest gratitude to my supervisor, Professor Jörg Ott, for giving me opportunity to work on such an interesting topic. I am indebted for his constant assistance, encouragement, guidance and tremendous support throughout the thesis process.

I want to give special thanks to one of my colleague, Teemu Kärkkäinen, who mentored me in every stage of my thesis through solving various issues and giving valuable comments.

Thanks also goes to my other colleagues, especially Sathyanarayan Suryanarayanan and Shengye Lu, who provided constructive suggestions on my thesis and inspired me all the time.

Finally, I would like to express deep gratefulness to my beloved parents for their fruitful advice and mental support during my studies.

Otaniemi, November 26, 2009

Md. Tarikul Islam

Contents

List of Figures

List of Acronyms

API	Application Programming Interface
BP	Bundle Protocol
BPA	Bundle Protocol Agent
CLA	Convergence Layer Adapter
DoS	Denial-of-Service
DTN	Delay-Tolerant Network(ing)
DTNRG	Delay-Tolerant Networking Research Group
EID	Endpoint Identifier
GUI	Graphical User Interface
IP	Internet Protocol
MIME	Multipurpose Internet Mail Extension
OMA	Open Mobile Alliance
ONE	Opportunistic Network Environment
PCM	Pulse Code Modulation
PoC	Push-to-talk over Cellular
PRoPHET	Probabilistic Routing Protocol using History of Encounters and Transitivity
PTT	Push-to-talk
RTP	Real-time Transport Protocol
RWP	Random Waypoint
S/MIME	Secure/Multipurpose Internet Mail Extension
SDNV	Self Delimiting Numeric Values
SIP	Session Initiation Protocol
SnW	Spray-and-Wait
SSP	Scheme-Specific Part
TCP	Transmission Control Protocol
TCPCL	TCP Convergence Layer
TTL	Time-To-Live
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VAD	Voice Activity Detection
WDM	Working Day Movement
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

1 Introduction

Wireless communication between mobile users has been increasingly widespread, due to advancement of the wireless networking technologies (e.g. Wi-Fi, 3G/GPRS, WiMax) and proliferation of personal handheld devices (e.g. PDA, Smartphone) around the world. With the advent of wireless Internet, the mobile users are able to use popular Internet applications like web, email from almost anywhere. Internet telephony, also known as Voice over IP (VoIP), is one of the key Internet services which can be considered as a rapidly growing technology over public Internet. In addition to the globally-deployed wired Internet telephony services, integrating VoIP into wireless networks has also gained significant acceptance in the recent years. But Quality of Service (QoS) is still not guaranteed to the wireless VoIP application, because of peculiar and unpredictable behaviors of wireless networks.

Push-to-talk (PTT) is a half-duplex voice application, which provides both individual and group communications in the walkie-talkie fashion over (mobile) wireless networks. Due to half-duplex nature of the communication, only one user can talk at a time and others can listen. PTT is commonly employed in the cellular phone services that use a single button to switch between voice transmission mode and voice reception mode. There is a version of PTT, called Push-to-talk over cellular (PoC), which is based on 2.5G or 3G packet-switched networks. The Open Mobile Alliance (OMA) has developed an open standard for PoC [1], which mainly relies on the Internet infrastructure.

The success of the traditional PTT services heavily relies on the performance of the Internet protocols. The traditional Internet protocols work on the basis of some basic principles. They assume that there is continuous end-to-end path between communication nodes, the delays and error probabilities in the network are relatively low. If the principles remain true, the protocols operate well. But in challenged environments, one or more of the above principles may be violated. Therefore the performance of the Internet protocols may deteriorate severely in such environments. On the other hand, PTT-like communication in the peer-to-peer fashion is also possible by forming an ad-hoc network. The classical ad-hoc routing protocols assume end-to-end path prior to forward the data. In such case node population is required to be sufficiently dense in order to establish end-to-end path. But if the ad-hoc network becomes increasingly sparse, then the ad-hoc routing protocols may

not work properly. So the design of new architecture and protocols is necessary to enable communication in challenged scenarios.

Delay-Tolerant Networking (DTN) is an emerging research area which is focused on addressing the communication requirements in the challenged networks, i.e., networks that may suffer from frequent disconnection, long or variable delay, high error rates, low or asymmetric data rates between source and destination. The Delay-Tolerant Networking Research Group (DTNRG) has devised DTN architecture and bundle protocol with the vision to improve network performance in the challenged environments. Delay-Tolerant Networks (DTNs) run as an overlay on top of heterogeneous type of networks, each of which may use different underlying networking technologies. The bundle protocol provides common means to interoperate among the diverse set of networks.

1.1 Related Works

The PoC service over either cellular networks or operator independent wireless networks, has received significant interest in the research arena. Wu et al. have proposed client architecture for the PoC service based on the OMA specification and implemented in the WLAN environment [2]. Akshai Parthasarathy in [3] presents a prototype implementation of Push-to-talk server in the Internet environment. In [4], Kim et al. have provided PoC solution for packet-switched networks accessed via GPRS/UMTS or WLAN technology, which is compliant with the OMA approach. IMS is used as a service infrastructure in their solution. In [5], Swapnil Kumar Raktale presents the architecture for PoC services in UMTS networks and evaluates the performance of the proposed architecture. Rui Santos Cruz et al. describes a PTT over IMS solution designed with a Talk Burst Control Protocol based on SIP messages for call session control [6]. They deploy and test their solution both with high bandwidth LAN and CDMA2000 wireless network. However the PoC enabled mobile devices may be used in environments where infrastructure is not available.

Valter Rnnholm presents an outline for a push-to-talk system over Bluetooth, which is independent of cellular networks. His solution is not based on client-server architecture unlike the OMA specified PoC [7]. Lin et al. have proposed peer-to-peer Push-to-talk (PTT) service for Voice over IP with the aim to provide PTT service over distributed and operator independent network environments [8]. Their solu-

tion is based on standard SIP and RTP/RTCP, and does not rely on functionalities provided by the underlying mobile networks. In [9], Chai-Hien Gan et al. propose a distributed PTT system for the Intelligent Transportation Systems environment. Under their system, group communication is performed through distributed learning interaction unlike the OMA central arbitrator approach. In [10], L.-H. Chang et al. have designed and implemented the PTT mechanism in ad hoc VoIP network. Under their implementation, the PTT server and user agent combined with the pseudo SIP server provide the PTT service without the support of standalone SIP server.

1.2 Motivation

We present the motivation of this thesis through discussing the limitation of existing PoC solutions. The traditional PoC services which are implemented in [2], [3], [4], [5] and [6], rely on infrastructure. But the mobile users may roam in an environment, where infrastructure is not available. In such case the PoC services may fail to communicate successfully.

The PoC solutions provided in [7], [8], [9] and [10], work in ad-hoc environments in the peer-to-peer fashion. But the traditional ad-hoc networking requires higher density of nodes to establish a network layer end-to-end path between communication peers. So the mobile users may not be able to communicate in the sparse ad-hoc networks using those PoC services.

All of the PoC solutions discussed above, either in infrastructure-based environments or in ad-hoc environments, more or less rely on the traditional Internet protocols. But the traditional Internet protocols require stable end-to-end path between source and destination for successful protocol operations. The protocols may not work in the scenarios characterized by intermittent connectivity and frequent partitions. Moreover the phenomena of attenuation and interference in wireless networks may lead to packet loss and this packet loss characteristics may degrade voice quality.

In all of the extreme scenarios, the performance of underlying networks may be reflected badly in the application performance, and eventually the overall user experience. So a unique solution is demanded to combat with the above problems.

1.3 Objective

The objective of this thesis is threefold. The first objective is to develop a voice communication system which enables both individual and group users to communicate over infrastructure-less and challenged networks in the walkie-talkie fashion. To achieve this we choose DTN technology as a basis, which allows communication in the sparse mobile ad-hoc and other challenged networking environments. We call our system DT-Talkie.

A preconfigured codec is used in the DT-Talkie for encoding and decoding. We employ an application layer framing mechanism to aggregate voice and optionally other contents in a structured way, rather than aggregating them simply one after another. We also focus on splitting the entire voice message into several pieces, which is basically the idea of application layer fragmentation. This is due to the fact that if the latency in the network is assumed to be low, sending fragments of each large voice message can boost up the interactivity of the DT-Talkie communication. Now the question is that how the voice message will be fragmented? Dividing a 10 seconds voice message into fixed-length fragments of 2 seconds is trivial and each fragment may contain fraction of a talk-spurt[1]. We suggest splitting up the voice message into meaningful fragments using silence-suppression mechanism, where each fragment holds single random-length talk-spurt of the voice message.

The second objective of this thesis is to implement the DT-Talkie for Linux based Nokia Internet Tablet, leveraging the DTN reference implementation[2] developed in the DTN Research Group. This serves as a validation of the concepts presented in the DT-Talkie. We use Maemo as a development platform and other open source technologies for the DT-Talkie implementation, with the aim to produce a useful piece of software.

The final objective is to carry out a set of simulations using the ONE simulator to evaluate the performance of the DT-Talkie using several DTN routing protocols in different mobility scenarios.

[1] A complete uttered sentence followed by silence period in a voice message.
[2] We will discuss about DTN Reference Implementation in Chapter 4.

1.4 Thesis Outline

In Chapter 2, we discuss about DTN architecture and bundle protocol in detail. A wide range of routing approaches in DTNs is introduced. We also give a brief overview about the OMA specified PoC architecture in this chapter.

We present an elaborate description of higher level components of the DT-Talkie system architecture in Chapter 3. This includes the discussion of application layer framing mechanism for transmitting optionally other contents along with voice messages. We explain how a fragmentation technique is applied in the DT-Talkie. Some codec negotiation approaches are also implied in this chapter.

Chapter 4 introduces implementation details of the DT-Talkie. We present a short discussion about high-level application architecture and then individual functional components of the DT-Talkie application are broadly described in this chapter.

In Chapter 5, we talk about mobility scenarios and simulation environment where a series of simulation is conducted. In this chapter we assess the performance of the DT-Talkie through discussing the simulation results.

Finally, the last chapter concludes the thesis with review of everything done before and with possible future works.

2 Technology Background

DTN is an approach which enables communication in the environments characterized by intermittent connectivity, long or variable delay, high error rates, low data rates and frequent network partitions. The traditional Internet protocols may fail to operate in those stressed environments. A delay-tolerant network operates as an overlay on top of diverse regional networks, including Internet. The bundle protocol, a primary protocol of DTN, provides key services to interoperate among various internets, regardless of underlying network characteristics.

In this chapter, we motivate the reasons for DTN. We give an overview of the major parts adopted in the DTN architecture. A detailed description of the bundle protocol is presented in this chapter. We introduce several classes of routing protocols proposed for DTNs and finally present the OMA specified PoC briefly.

2.1 Why DTN?

The existing Internet protocols require stable end-to-end path between source and destination prior to data transfer. The motivation of DTN stems from the observation that there are some environments where the Internet protocols may fail to establish end-to-end path. The overall performance characteristics of the existing Internet protocols rely on some fundamental assumptions of underlying networks. The assumptions include continuous end-to-end connectivity, low end-to-end delay, symmetric data rates and low error rates. The protocols perform well when the above assumptions are met. The challenged networks do not conform to one or more of the Internets underlying assumptions and the Internet protocols may work poorly or may even fail completely in those networks.

One class of challenged networks is the Interplanetary Internet, which focuses on the issues of deep-space communications in high-delay environments. The Interplanetary Internet is considered as the basis for DTN architecture [11]. Typically, the round-trip delays are just fractions of a second in the existing terrestrial Internet which spans the globe. On the other hand, the delays may be several minutes or hours in the deep-space communications. For example the round-trip transmission delay between Earth and Mars lies between 4 minutes to 20 minutes [12]. Using Internet protocols in such an environment would be highly impractical.

DTN strives to overcome the problems associated with intermittent connectivity, long or variable delay, high error rates, low data rates and frequent partitions, using store-and-forward message switching mechanism. DTN also defines a common overlay layer called bundle layer which lies on top of dissimilar regional networks and offers generic services to communicate in the non-compatible network environments.

2.2 DTN Architectural Overview

The DTN architecture is aimed to provide interoperable communication among a wide range of networks that may suffer from frequent partitions and that may be used more than one divergent set of lower layer protocols or protocol families. The architecture is based on message switching abstraction. The DTN architecture is also envisioned to enable communication in the extreme challenged and disruptive environments where the protocols of todays Internet may operate poorly, or may even fail completely. The DTNRG [13] has defined the design principles in [11] to interconnect challenged and frequently-disconnected networks. Major features of the DTN architecture guided by those design principles are briefly described below.

2.2.1 Store-and-Forward Message Switching

In the DTN architecture, application data sent by DTN-enabled application is carried in variable-length messages, called bundles. The idea of bundles stems from the consideration that attempts to reduce the number of round-trip message exchanges by bundling all information together. This makes sense in the scenarios where round-trip time is hours, days, or even weeks [14]. Bundles are sent towards the destination using store-and-forward model like the e-mail system. Intermediate nodes along the path from the source to the destination hold the bundles in storage for a while until the next node becomes available. This means there is no need of end-to-end connectivity between source and destination at any point of time unlike traditional Internet. The bundles are typically stored in persistent storage to increase reliability and to cope with hardware failures.

Even though IP networks are also based on store-and-forward operation, there is an assumption that the packets will be stored in the Internet routers queue for shorter duration, whereas in the DTN architecture, it is not expected that network links

are always available. In the typical DTN scenario, bundles are stored in persistent storage (perhaps for longer period of time) and forwarded as soon as the opportunity occurs.

2.2.2 Endpoint Identifier (EID) and Registration

A DTN node is an entity that can send, receive or forward bundles. A DTN endpoint is therefore a set of DTN nodes. In the DTN architecture, All nodes are identified by a unique *endpoint identifier* (EID), which conforms to the Uniform Resource Identifier (URI) [15] syntax. An EID may refer to one or more DTN nodes and an individual node may have more than one EID. EIDs may be of unicast, anycast, multicast or broadcast types. Every node, however, must have at least one EID that identifies it.

A DTN EID is composed of an EID scheme followed by a scheme-specific part (SSP): <scheme name>:<scheme-specific part>. "dtn:" is the one default scheme in the bundle protocol that takes arbitrary string as SSP. An example of an EID is as follows:

```
dtn://host.dtn/path
```

The first part of the EID before the colon is the scheme name and the subsequent part after the colon is the SSP. DTN EIDs are restricted to being not more than 1023 bytes [16]. There is a notion of null EID in DTN where no addressing is included, and this is represented by "dtn:none". Wildcarding some portion of EID may be useful for routing and diagnostic purposes. The adoption of URI-like general naming syntax allows multiple naming schemes to be used in conjunction with the basic DTN protocols. Currently there is an Internet draft which proposes the use of other EID schemes [17].

A *registration* is a process that associates an EID to an application that is intended to receive ADUs to that particular EID, and is maintained persistently by a DTN node. Any bundles received by a DTN node at a registered EID, are transferred to the application with the associated EID. A DTN node may have several registrations concurrently. A single registration at any point of time can be one of two states:

Active or Passive.

2.2.3 Late Binding and Class of Service

In any communication network, name resolution techniques may be required to actually locate the destination. The DTN architecture calls the idea of *late binding*, where the binding of a destination EID to a region-specific lower layer address does not necessarily happen at the bundle source node. The binding may take place at the source node, at the intermediate nodes during transit, or possibly at the destination. This is completely opposite to the DNS and ARP name resolution techniques of traditional Internet, in which the mapping occurs at the source node before transmitting the data. The late binding principle is beneficial in the occasionally-connected networks because the transit time of a message may exceed the validity time of a mapping. So the mapping at the source node in this case would be impractical.

The DTN architecture defines three different classes of service for bundling priorities which include bulk, normal and expedited. A bulk bundle has the lowest priority and it is not delivered before other classes of bundles. Normal-class bundles are shipped before bulk-class bundles but not prior to expedited-class bundles. An expedited bundle gets the highest priority over other classes of bundles. The priority class of bundle is only necessary to differentiate bundles from the same source. Based on a particular DTN node's forwarding policy, priority may or may not be applied on different sources [11].

2.2.4 Fragmentation and Reassembly

As we have discussed earlier that application data is conveyed in variable-length bundles, but there may be some situations where contacts between DTN nodes are only of such a short duration that the entire bundle cannot be sent in one piece. DTN incorporates fragmentation and reassembly to enhance the efficiency of bundle transfer through ensuring maximum utilization of link capacity and through avoiding retransmission of partially-forwarded bundles.

Two types of fragmentation exist in the DTN architecture: *proactive* and *reactive*. In proactive fragmentation, a DTN node divides the entire large bundle into multiple fragments prior to a transmission attempt and transmits each fragment as an

independent bundle over the DTN infrastructure. This approach is used in the scenario where contact volumes are predicted or known ahead of time to the DTN node [11].

In reactive fragmentation, a bundle may be fragmented cooperatively when the bundle is only partially transferred. In this case the previous-hop DTN node may learn via lower-layer protocols that only a portion of the entire bundle was transmitted to the next hop and send the remaining portion(s) when subsequent contacts become available (likely to different next-hops if routing changes).

In case of both fragmentation types, the fragments are only reassembled at the final destination node. Fragments may be further fragmented, either proactively or reactively. To verify the integrity of a digitally signed bundle, it can be set "do not fragment" flag in the bundle to avoid fragmentation [12].

2.2.5 Contacts

A contact is defined as a time period during which two nodes have the opportunity to communicate. In the highly disrupted and frequently-disconnected networks, all the nodes may not contactable at any point of time. This is in contrast to the principle of the regular Internet where nodes are strictly considered to be online all the time [12]. The following major types of contacts have been defined in [11].

- Persistent - links are always available and no action is required to instantiate.
- On-Demand - like persistent contact but needs some action in order to instantiate.
- Scheduled - An intermittent link established for a particular duration which may or may not be periodic.
- Opportunistic - An intermittent link formed unexpectedly without any prior knowledge.
- Predicted - An intermittent link is setup based on previously observed contacts or by other means.

2.2.6 Time

The DTN architecture requires some degree of time synchronization to identify bundle and to compute bundle expiration time. Bundle identification and expiration are maintained by setting the creation time and lifetime in each bundle. If a DTN node in transit receives a bundle and the bundle is expired, then the bundle will be dropped and no longer forwarded. Time information is also needed to define application registration expiration. When an application registers to a particular DTN node, this registration is maintained only for finite period of time specified by the application [11].

2.2.7 Custody Transfer

Since communication in DTNs does not rely on end-to-end connectivity, ensuring end-to-end reliability in this kind of communication is a challenging problem to cope with. The DTN architecture supports the notion of custody transfer to improve delivery reliability of a message, effectively creating hop-by-hop reliability. This is achieved through transferring the responsibility of reliable delivery in the intermediate DTN nodes along the path from source to destination.

Custody transfer operation is initiated by the source application. The source node sends a bundle with custody transfer request to the next hop and starts a time-to-acknowledge retransmission timer. If the next node accepts custody, an acknowledgement is returned to the sending node through return receipt. The node which currently has a bundle with custody request is called the custodian of the bundle. However the source node retransmits the bundle if no acknowledgement is received before expiration of the retransmission timer. A custodian must store the bundle until another node accepts custody or the bundles time-to-live expires [18].

In DTN architecture, every node is not necessarily provided custodian transfer service. Some nodes may refuse to accept custody transfer for message due to shortage of free storage space or power limitation. The custody transfer mechanism is advantageous to allow an endpoint to free storage resources as soon as a custody transfer acknowledgment arrives [19].

2.2.8 Security

As discussed before, the DTN overlays on top of heterogeneous regional networks. These underlying networks might suffer from scarce resources such as limited bandwidth, limited connectivity, constrained storage in the intermediaries, etc. Security is a critical issue in the DTN architecture, where the communicating nodes running in the extreme environments, may be threatened by malevolent security attacks. The DTN Security Overview document [20] highlights some possible security threats that pose unique challenges to secure DTN communication. These include unauthorized resource consumption and denial-of-service (DoS) attacks. Without integrity and confidentiality, bundle data might be corrupted or read by malicious users while in transit. So security services are required in some circumstances in the delay-tolerant networks. The DTN architecture security requirements differ from traditional network security model in the sense that security services need to be incorporated in the intermediate DTN nodes in addition to the source and destination node [21].

DTN security is concerned with the authenticity, integrity and confidentiality of bundles conveyed among bundle nodes. These features are realized via the use of three independent security specific bundle blocks, which may be used together to provide multiple bundle security services or independently of one another, depending on perceived security threats, mandated security requirements, and security policies that must be enforced [16].

DTN security allows for intermediate DTN nodes to apply or check the validity of the cryptographic credentials. The nodes are called security-source and security-destination, which may or may not be the original bundle source and destination nodes. Authenticity and integrity can be provided by means of the Bundle Authentication Header (BAH) block along a single hop from sender to receiver and the Payload Security Header (PSH) block from PSH security-source to PSH security-destination. Secrecy can be assured by using the Confidentiality Header (CH) block between CH security-source and CH security-destination offered by BSP [22].

2.3 Bundle Protocol

As mentioned earlier, DTN provides communication in the extremely challenged environments like those with intermittent connectivity, long or variable delays, high

error rates and low bit rates. DTN also defines an overlay network which operates on top of heterogeneous internets running diverse family of protocols. A bundle is the variable-length DTN protocol data unit that contains application data as well as signaling information needed to traverse the overlay network. In DTN, a bundle node is an entity that can send, receive, or forward bundles - typically, a process running on a general-purpose computer but might be a thread, an object, or a special-purpose hardware device [16]. The key protocol of DTN used by bundle nodes is called bundle protocol and the layer in which the protocol works is termed as bundle layer.

DTN protocol hierarchy is depicted in Figure 2.1. The bundle protocol sits at the application layer or at least above the transport layer. A bundle node can be acted as a *host*, *router*, or *gateway*. A host can be a source or destination in the DTN communication which sends and receives bundles, but does not forward them. A router is an intermediary which forwards bundles within a single DTN region (e.g. from Internet to Internet). A gateway is also an intermediary which forward bundles among two or more dissimilar regions (e.g. from Internet to non-Internet). Both router and gateway may optionally be a host [18]. In the left-hand side of the figure, the bundle protocol runs on top of the Internet. The right-hand side shows a network under the bundle layer which uses different suite of protocol other than TCP/IP. The bundle protocol communicates with arbitrary transport protocols to provide interoperable communication. The application layer does not aware of the underlying transport mechanisms.

In DTN, each bundle node has three conceptual components (Figure 2.2): a *bundle protocol agent*, zero or more *convergence layer adapters* and an *application agent* [16]. The bundle protocol agent (BPA) of a node is the component that implements bundle protocol functionalities. It has an interface with the application agent (AA) to provide bundle protocol services. A convergence layer adapter (CLA) is the bundle node component that implements convergence layer protocol. It sends and receives bundles on behalf of the BPA, using native internetwork protocol services. A CLA interfaces between the BPA and the specific internetwork protocol suite. A bundle node may have several CLAs, which enables the BPA to adapt in heterogeneous networking environment. The application agent (AA) is the component that utilizes bundle protocol services for effective communication and could provide interfaces to upper layers.

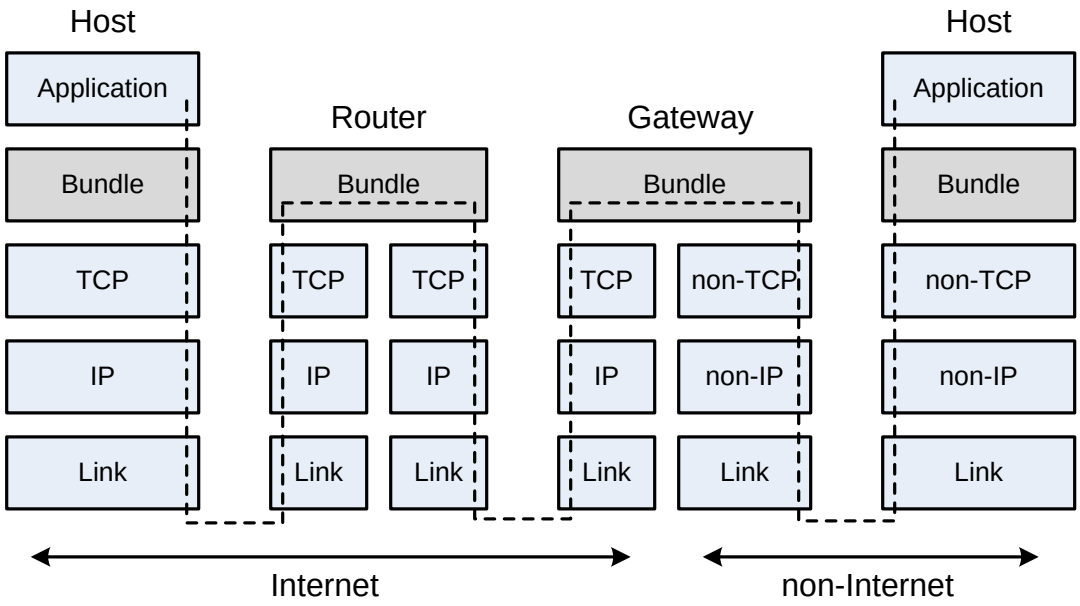

Figure 2.1: DTN protocol hierarchy

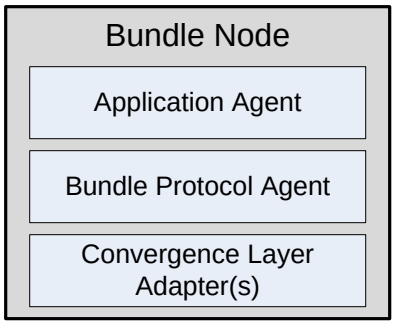

Figure 2.2: A bundle node classification

2.3.1 Basic Bundle Structure

Each bundle is a sequence of at least two or more *blocks*[1]. First block in the sequence is called primary block which contains basic information to route bundles to destination. No bundles can have more than one primary block. Other types of bundle blocks (e.g. bundle security block) may follow the primary block, to support extension of the bundle protocol. Last block of the sequence is the payload block that holds application data [16]. DTN incorporates some noteworthy encoding mechanisms to represent the bundle block fields, which are discussed below in a nutshell.

[1]Equivalent to the concept of headers used in other network protocols.

Self Delimiting Numeric Values (SDNV)

DTN defines Self Delimiting Numeric Values (SDNV) encoding scheme to represent number. Many block fields in the bundle protocol use SDNV, which is basically fairly a flexible way to encode non negative integer numbers of arbitrary magnitude, without consuming unnecessary space [23]. SDNV encoding is based on the basic encoding rules (BER) encoding of abstract syntax notation one (ASN.1) [24]. A SDNV is a numeric value encoded into a series of octets, where the most significant bit (MSB) of each octet is used as flag, leaving seven bits remaining to carry information. The flag is set to 0 for last octet and set to 1 for other octets of the SDNV. The following is an example of encoding the hexadecimal value 0x4234 to SDNV:

0x4234 = 0100 0010 0011 0100 is encoded as 10000001 10000100 00110100

The SDNV scheme efficiently represents very large and very small integer values. However the scheme is not ideal for representing numeric values that fall in the range from 128 to 255 [16].

Dictionary and Timestamp encoding

The dictionary is used in the primary block of a bundle to group variable-length EIDs together. The reference to the actual EID string within the dictionary is encoded as an offset of the length of two octets. First octet represents the offset of the scheme name and second octet points to the scheme-specific part or SSP of the EID. This mechanism is advantageous in the sense, if the same EID string is used more than once, there will be no extra overhead for the additional occurrences [14].

Timestamp fields in bundles use 8-byte to represent time. The high-order 4 bytes are used to encode coordinated universal time (UTC) in seconds since the start of the year 2000. The remaining 4 bytes hold nanosecond value to differentiate between bundles generated during the same second [12].

2.3.2 Convergence Layer Protocols

Since the bundle protocol is an overlay protocol, it requires some mechanism to communicate with heterogeneous underlying protocol stack (e.g. TCP/IP), that evolves the idea of convergence layer. The convergence layer provides abstraction to adapt lower layer protocols. The layer performs mapping between the bundle protocol and network-specific lower layer protocol. This allows the bundle protocol agent to run over wide range of network types. The Bundle Protocol Specification document [16] summarizes the services of the convergence layer. The bundle protocol agent expects the following services from the convergence layer.

- Sending a bundle to the node identified by a specific EID that is reachable via the convergence layer protocol.
- Delivering a bundle to the bundle protocol agent that was sent by a remote bundle node via the convergence layer protocol.

For TCP-like reliable transport protocol, the design of the convergence layer protocol is fairly simple. The convergence layer just needs to aware of connection management and message delimiting issues. For unreliable transport protocol like UDP, a separate implementation for ensuring reliability should augment in the convergence layer protocol.

There is an Internet draft exists for TCP-based convergence layer protocol (TCPCL) [25]. TCPCL specifies bundle transmission over TCP transport protocol with considering two aspects: connection setup and teardown, and bundle encapsulation. Before establishing a TCPCL connection between two communication nodes, a TCP connection is initiated. After successful completion of TCP connection procedures, an initial contact header is exchanged in both directions, which conveys TCPCL connection parameters and a singleton EID to identify bundle endpoint. When the TCPCL connection is established, bundle is sent in one or more segments in either direction. The length of each segment can be variable and is specified in the segment header. The starting and ending segments are identified through flag values in the segment header.

In TCPCL the receiving node sends acknowledgements when the bundle data segments arrive, as an optional feature. Through these acknowledgements, the sending

node can keep track of the number of bundles received. This enables the sender to perform reactive fragmentation in case of connection interruption. Another optional feature of TCPCL is that a receiver may tell the sender to stop transmission of the current bundle by sending a negative acknowledgement, after receiving a portion of the bundle data segment. A message may be sent optionally to keep the idle connections alive. TCPCL also defines a message to release the connection.

There is another draft exists that specifies the convergence layer for transmitting bundles over UDP [26]. Other convergence layer protocols have been suggested to support delivery of DTN bundle directly over a link layer, e.g. directly over Bluetooth, or directly over wireless Ethernet [27].

2.4 Routing in Delay-Tolerant Networks (DTNs)

All communications network must have the fundamental feature to route data from source to destination. For all the routing protocols proposed for MANETs (e.g., OLSR [28]and AODV [29]), it is implicitly assumed that the network is connected and there is an instantaneous end-to-end path exits between any source and destination pair. In such cases when packet arrives and no instantaneous end-to-end paths for their destinations can be found, they are simply dropped. These protocols do not work properly in the DTNs, which are characterized by frequent network partitions and intermittent connectivity. So new routing mechanisms should be developed for DTNs.

Routing is one of the key components in the DTN architecture. Currently there is no routing protocols defined to be used in conjunction with DTN. However various routing protocols have been proposed from research communities for DTNs and also implemented. Each of the proposed routing approaches has both advantages and disadvantages in particular scenario. No routing schemes proposed so far are considered ideal for DTNs.

A wide range of routing protocols has been studied in [30] based on two different types of DTNs, deterministic and stochastic. In the deterministic networks the future network topology is known beforehand or at least foreseeable, whereas the network topology is totally unknown or just could be estimated in case of stochastic

networks. The classification of various routing approaches in DTNs on the basis of deterministic and stochastic cases is depicted in Figure 2.3.

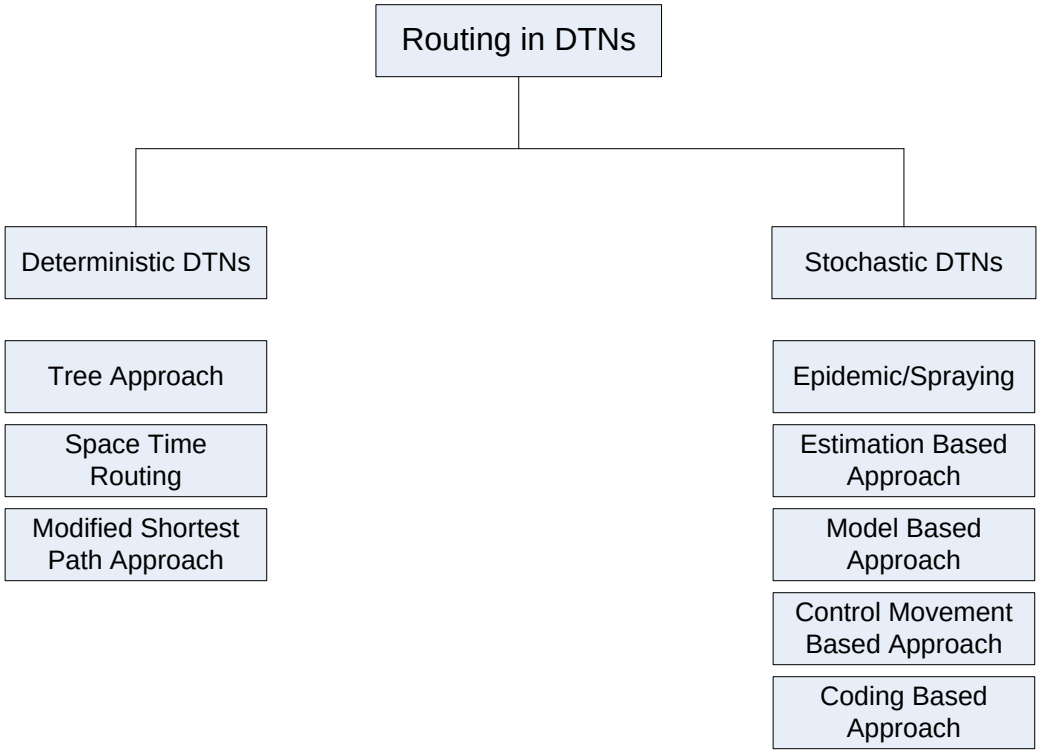

Figure 2.3: Classification of routing approaches in DTNs

2.4.1 Routing in Deterministic DTNs

Here we discuss some routing approaches briefly that work with the assumption of complete knowledge about future node movements and connections. For all the approaches under the deterministic case, an end-to-end route is determined before messages are actually delivered.

Tree approach

In [31] the authors have proposed tree approach to be used in the deterministic networks. Under the tree approach it is assumed that each host has the global knowledge of characteristics profile (motion and availability of the hosts) of others with respect to space and time. A tree is built from the source host by adding children nodes and time associated with nodes. Each node carries information about

all the previous nodes and the minimum time to reach them. To deliver messages to the intended destination, a shortest path can be selected from the tree by choosing the minimum time.

Space time routing

Rather than having knowledge of the characteristic profile for infinite time period, the authors in [32] have assumed that the characteristic profile can be accurately predicted over the time interval of T. They model the dynamic of the networks as a space-time graph and developed routing algorithms using dynamic programming and shortest path algorithm.

Modified shortest path approaches

Several routing algorithms for DTNs are proposed in [33] depending on the certain knowledge of the network (*knowledge oracles*). They define four knowledge oracles. The *Contacts Summary Oracle* contains information about aggregate statistics of the contacts. The *Contacts Oracle* provides information about contacts between two nodes at any given time. The *Queuing Oracle* gives information about instantaneous buffer occupancies at any node at any time. The *Traffic Demand Oracle* contains information about the present or future traffic demand.

Based on the assumption of which oracles are available, the authors in [33] present corresponding routing algorithms. For example, if all the oracles are known, a linear programming is devised to determine the best route. If only the Contacts Summary Oracle is available, Dijkstra with time invariant edge costs based on average waiting time is used to select the best path. If only the Contact Oracle is available, modified Dijkstra with time-varying cost function based on waiting time is used to find the route.

2.4.2 Routing in Stochastic DTNs

Here we present a brief overview of the routing schemes in the stochastic networks, where routing decisions may be simply to forward messages to any contacts within range. The other routing decisions may be based on history data, mobility patterns,

or other information.

Epidemic

In [34] the authors propose an epidemic routing protocol for intermittently connected networks. When a message arrives at an intermediate node, the node floods a copy of the message to other encountered nodes. Hence, messages are quickly disseminated through the connected portions of the network. In their scheme, when two nodes come into communication range with one another, they exchange only those messages which have not seen yet by either node.

While flooding-based schemes ensure high delivery rates, they are responsible for huge resource consumption which may be scarce in DTNs. To achieve resource efficiency, Spray-and-Wait [35] routing protocol limits the distribution of messages by setting a strict upper bound on the number of copies per message allowed in the network. The routing scheme sprays a number of copies into the network, and then waits till one of these nodes meets the destination.

Estimation Based Approach

A probabilistic routing protocol PRoPHET (Probabilistic Routing Protocol using History of Encounters and Transitivity) [36] uses knowledge of previous encounters for selecting suitable next hops to deliver a given message. PRoPHET first estimates a delivery predictability value, which in turn is used to decide whether a copy of the data item is forwarded to an encountered node. When two nodes meet, they exchange a delivery predictability vector containing the delivery predictability information for known destinations and update the predictability value.

Model Based Approach

In epidemic and estimation based routing schemes mobile nodes are assumed to move randomly without any specific knowledge of their trajectories. But in practice mobile nodes follow some certain known patterns. Model Based Routing (MBR) is proposed in [37], which uses knowledge of movement patterns to improve routing.

MBR relies on user profile to choose a relay that moves to the destination with higher probability.

Node Movement Control-Based Approach

In this approach, the trajectories of some nodes can be controlled to improve overall system performance metrics such as delay. In [38] the authors describe a Message Ferrying (MF) approach for data delivery in sparse mobile ad-hoc networks. MF utilizes a set of special mobile nodes called *message ferries* to provide communication services for nodes in the network. These *message ferries* move around the deployment area and take responsibility for carrying data between nodes.

Coding Based Approach

Erasure coding and network coding techniques have been recently proposed to improve routing in wireless ad-hoc networks and DTNs. The basic idea of erasure coding is to transform a message of k blocks into n (n > k) blocks such that if k or more of the n blocks are received, the original message can be successfully decoded. In [39], both analytical and simulation results show that erasure coding based forwarding in DTNs can significantly improves the worst-case delay.

The main concept behind network coding is that intermediate nodes combines some of the packets received so far and send them out as a new packet. A probabilistic forwarding approach based on network coding is proposed for DTNs in [40]. In their approach, after generating new packets using network coding, a coding vector is attached to each new packet. When a packet is received at a node, d new packets are generated and broadcast to neighbors. The receiver can reconstruct the packet once it has received enough packets.

2.5 OMA specified PoC

PoC provides a combination of VoIP telephony services and instant messaging style properties such as presence and messaging. The key advantage of PoC services is that a single user can reach an active talk group with just a button press, thus the user no longer needs to make call to each of the group. Like many VoIP solutions, the OMA

PoC solution is based on the classical Internet multimedia protocols (Figure 2.4).

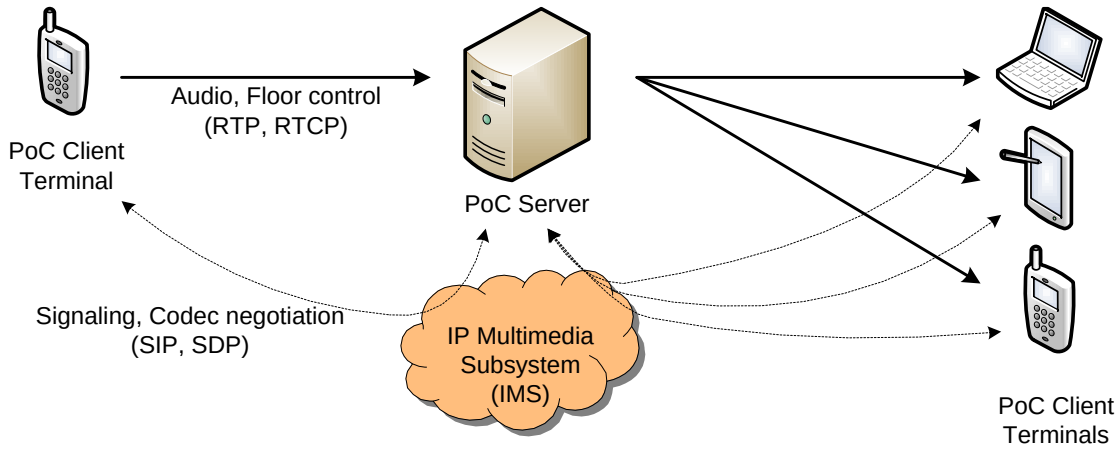

Figure 2.4: OMA specified PoC architecture

PoC Signaling

PoC signaling for session establishment is based on the Session Initiation Protocol (SIP) [41]. SIP is a text based end-to-end application layer protocol used to establish, modify and terminate multimedia (e.g., audio, video) sessions. SIP can run on top of several transport layer protocols such as TCP, UDP and SCTP. There are several functional entities comprised the SIP architecture includes: user agent client, user agent server, registrar server, proxy server, redirect server, location server and back-to-back user agent. Like Hypertext Transfer Protocol (HTTP), SIP is based on request-response model. It defines INVITE, BYE, REGISTER, OPTIONS, etc. request messages, several provisional and final response messages.

During establishing PoC session, Session Description Protocol (SDP) is used in a SIP message body to negotiate codecs and other media related parameters between two parties in the fashion of offer/answer model. Media parameters may be re-negotiated in certain cases while ongoing voice session.

PoC Speech

PoC speech is transmitted in Real-time Transport Protocol (RTP) [42] packets. RTP is an IP-based protocol providing support of real-time data such as audio and video

streams. RTP is typically run on top of UDP to make use of its multiplexing and checksum functions. To deliver media streams timely, RTP includes time stamping, sequence numbering and other mechanisms to take care of timing issues. The receiving PoC client uses the timestamp to reconstruct the original timing in order to play out data in correct rate. The RTP source identification allows the receiving application to know where the data is coming from.

RTP is designed to work in conjunction with the auxiliary control protocol, RTP Control Protocol (RTCP), to get feedback on the quality of data transmission and information about participants in the ongoing session. In the OMA PoC, RTCP provides floor control through arbitrating requests from PoC clients for the right to send media.

2.6 Summary

In this chapter, we have presented Delay-Tolerant Networking (DTN) concepts. DTN is designed to address communication requirements specific to challenged networking environments, which include intermittent connectivity, long or variable delay, high error rates and low bit rates. In such scenarios, traditional Internet protocols may perform poorly or may not work at all.

In DTN, application data is carried over variable-length bundles. The bundles may be stored persistently in the intermediate nodes until the next contact becomes available. This allows DTN not to rely on stable end-to-end path for communication at any point of time unlike Internet. Bundle protocol, which we have discussed here, is the primary protocol of DTN. The bundle protocol operates as an overlay on top of wide range of network types, while the convergence layer performs mapping between bundle protocol and network-specific lower layers. Several convergence layer adapters (CLAs) can be integrated in a bundle node which enables bundle protocol agent (BPA) to interoperate with heterogeneous underlying networks.

Routing is one of the biggest open issues in DTNs. In this chapter, we have also introduced various routing approaches based on deterministic and stochastic DTNs. Deterministic routing schemes work on the assumption of having complete or partial knowledge of the network topology in advance. On the other hand, the information

about the network topology is totally unknown for stochastic cases. However none of the routing approaches are treated as perfect for DTNs.

The OMA specified PoC architecture has been introduced briefly in this chapter. The traditional PoC services rely on Internet protocols which require stable end-to-end path for successful operations. This may subject to abnormal communication experience in the mobile DTNs where end-to-end path may not exist at any given time. In this thesis we develop DT-Talkie which enables mobile users to communicate in the infrastructure-less and other challenged environments in the walkie-talkie fashion. In the next chapter, we describe the high-level architectural concepts of the DT-Talkie.

3 System Architecture

As the Open Mobile Alliance (OMA) specified Push-to-talk over Cellular (PoC) service works over either cellular networks or operator independent wireless networks, it is not very well-suited for infrastructure-less and challenged networks. In the regular PoC services, several round-trip messages are required to exchange in order to establish a PoC session. After establishing the session, voice data is transferred back and forth in the semi real-time fashion between endpoints either directly or via infrastructure. These services work well in the scenarios (e.g., Internet) when round-trip delays are significantly short. But in a DTN environment of high end-to-end delay and high error rates, the PoC session might not be established in the first place, or huge packet loss may lead to degraded voice quality. Overall, the traditional approach of voice communication would be unrealistic in the disrupted environments.

Our aim is to define requirements for practical voice communication in the mobile DTNs, in which speech quality is not affected, but the voice session can be less interactive due to delay. To avoid unwanted round-trip message exchanges for session establishment, session parameters and optionally other contents can be bundled along with recorded voice messages. In order to achieve this, we must define how different data chunks can be aggregated together into larger data structures, and how these structures are placed into bundles. It might happen that the mobile endpoints experience better link connectivity while ongoing voice session. In such cases we must provide means for speeding up the session interactivity.

In this chapter, we provide a high-level overview about the DT-Talkie system and then proceed to discuss the system design broadly. We also describe the fragmentation approach that is applied to the DT-Talkie. Finally some mechanisms to mitigate the codec interoperability issues are suggested.

3.1 System Concepts of the DT-Talkie

The existing PoC solutions are mainly infrastructure-based and rely on stable end-to-end path for successful communication. But the mobile nodes may roam in the environments, where infrastructure is not available or unreliable end-to-end path does exist. In such cases, those PoC solutions may exhibit poor performance or

may even fail completely. So a new approach is necessitated to define for enabling voice communication in the infrastructure-less and other challenged networking environments. In this thesis, we devise a system called DT-Talkie in which the DTN concept of asynchronous message forwarding is applied to transport voice messages in the infrastructure-less and challenged networks.

Figure 3.1 depicts the general processing steps of the DT-Talkie. Basically in the DT-Talkie, voice messages are captured and then encoded using a preconfigured codec. It might happen that the destination endpoint does not have support of the codec to decode and playback voice messages. One solution to deal with the problem can be that two endpoints negotiate on a common codec prior to voice message transfer like traditional PoC services. But this requires several round-trip message exchanges, which is not feasible in the high delay environment. In order to handle the issue we primarily provide support of more codecs to the DT-Talkie which can make the system more codec-interoperable. We also suggest some other approaches concerning codec interoperability issues, which are discussed later in this chapter.

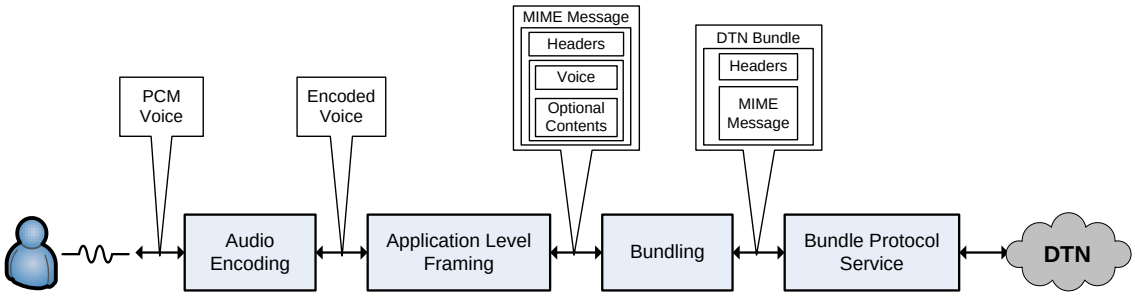

Figure 3.1: General processing steps of the DT-Talkie

However after encoding the voice messages, the next step is to encapsulate them into bundles. Rather than simply bundling the unstructured voice messages, we use a framing mechanism which enables aggregation of other optional contents in conjunction with the voice messages. The bundles are then sent using the DTN bundle protocol services, which are forwarded reliably on the way to the destination in the store-carry-forward fashion. The bundles are decapsulated after receiving to get the voice messages, which are played out after decoding.

After briefly describing higher level architecture of the DT-Talkie, now we provide in-

depth discussion about the system design with the aid of a one-to-one communication scenario. Suppose user A attempts to send a voice message to User B and both have DTN communication capabilities.

3.1.1 Audio Encoding

DT-Talkie captures voice messages from the audio source (e.g., microphone) of user A. The captured voice messages can be sent in either uncompressed or compressed format. The uncompressed audio formats, often referred to as PCM formats, do not use any compression mechanism. This means all the audio information is available at the cost of large file size. A WAV audio file is an example of uncompressed audio. The compressed audio can be of two types: lossless and lossy. Lossless audio compression applies to an uncompressed audio file without any loss of audio information, whereas lossy compression technique results better compression through eliminating redundant and unnecessary audio information. Since lossy compression discards only those parts of audio which cannot be perceived by human auditory system, it has very little impact on audio quality. For example MP3 and G729 audio files use lossy compression. DT-Talkie supports transmission of voice messages both in uncompressed (WAV) and compressed (MP3 and G729) audio format. In this thesis, there is no mechanism employed in the DT-Talkie to negotiate audio encoding format before delivering voice messages. Rather we assume a default audio format for a particular voice session, which is preconfigured in both side of user A and user B.

3.1.2 Application Layer Framing

DT-Talkie allows sending optionally other contents (e.g., image, electronic business card) together with a voice message. The image can be either User A's profile picture or instant snapshot from the devices camera. The electronic business card (vCard) [43] is often attached to email messages, but can be significantly used in the DT-Talkie to provide information related to sender's name, email address, phone numbers and so on. Application layer framing mechanism is applied to place the voice message and other contents into a single structure. We use MIME (Multipurpose Internet Mail Extension) [44] for this purpose.

MIME is an IETF defined standard, which enables electronic mail program and web-

browsers to send and receive non-ASCII messages (e.g., image, audio and video) via the Internet. A MIME message includes both data and metadata. MIME metadata consists of HTTP-style headers and MIME boundary delimiters. Different types of contents can be encapsulated in different parts of a multipart MIME message and the *Content-Type* header field specifies the type of the content in each part. In such multipart messages the *Content-Type* header also includes a boundary attribute that is used to delimit the message parts. MIME provides the support of custom user-defined headers (prefixed with "X-") to be used for application-specific purposes. Based on MIME standard, S/MIME provides cryptographic security services (e.g., authentication, encryption) for the applications that transport MIME data.

Figure 3.2 presents an example MIME message in short form which is generated in a DT-Talkie session. We add *X-Bundle-Destination* header in the MIME message to refer two types of DT-Talkie communication. The value of the above header is either singleton for one-to-one or multinode for group communication. The value of the *X-Bundle-Type* header can be either message when the DT-Talkie sends a full-length voice message or fragment if the fragments of the voice message are sent. The encoded voice message is aggregated as a body part of type *audio/G729* and the image as another body part of type *image/jpeg*. The business card information is aggregated in the MIME message with type *text/x-vcard*. The vCard contains a formatted name string which can be used as a display name and an Internet email address.

Some protocols cannot carry binary data, or data with a line length of greater than 1000 lines - specifically Simple Mail Transfer Protocol (SMTP) has both those restrictions. The MIME messages contains binary data need to be transformed in such way so that the messages will be transportable by those restricted protocols. To achieve this, a binary-to-text encoding mechanism is required to apply. Base64, one of the MIME supported encoding scheme, is used to transform binary data (e.g., images, audio) into a text string that contains only US-ASCII characters. But the pitfall of the scheme is that it increases data volume by 33% [44]. In our case the restrictions do not apply because we use DTN transport which supports transmission of binary data without any limitation; thus reduces overall message overhead.

```
X-Bundle-Destination: singleton
X-Bundle-Type: message
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="=-UL99WySTLs0AAPGPLgtN"

--=-UL99WySTLs0AAPGPLgtN
Content-Type: audio/G729

1Í□□GŸTS§QâXJqßÚÑ□‰saÍ^cÃÝÛ"Gl/Œ ^Å^□□Øð[;a□öÁÆ 'ÿ□hT©—
_ŽBÃT^li˜Ov)ÿQäuú^+!š¥öùÙ"=8□A€¤òn={Tf□å:\yðâÔû□kÈú{äýIsßm \
.....
--=-UL99WySTLs0AAPGPLgtN—
Content-Type: image/jpeg

□|]ªë□eõÙ»+ö8PIi*nC6î`Œ¨aŒ|Êsšâü Wc7‰üG§ê7¶ö§GÓbka$ó□□Á□Û¶äV!w`□~
µÅ□□øÁŸ`ƒK¾û4:¥»KöR¾Ê¹□B àþcŠó½ [MBå,m.o4Éæ´0HË6øç
.....
--=-UL99WySTLs0AAPGPLgtN
Content-Type: text/x-vcard

BEGIN:VCARD
VERSION:2.1
FN:Forrest Gump
EMAIL;INTERNET:forrestgump@example.com
.....
END:VCARD
--=-UL99WySTLs0AAPGPLgtN
```

Figure 3.2: Sample MIME message generated in a DT-Talkie session

3.1.3 Bundle Addressing

After aggregating encoded voice and image into a MIME message, the DT-Talkie application encapsulates the MIME message into a DTN bundle. Now the bundle is needed to be addressed for transmission. As described before, All the endpoints in the DTN domain are identified by an URI-like endpoint identifier (EID), which has the general form of <scheme name>:<scheme-specific part>. Every node has a unique singleton EID but can register to any number of multicast EIDs. When an application wishes to receive bundles destined to a particular EID, it registers the corresponding EID with its local DTN node.

Basically the bundle protocol is not limited to a specific URI scheme - any valid scheme (e.g., dtn:, http:, mailto:) can be used in an EID. The scheme name defines a set of rules that determine how the scheme-specific part should be interpreted [16].

However the "dtn:" scheme is adopted as a default scheme in the DTN2 reference implementation using the structure of "dtn://node-id/application-id" (described in the earlier versions of the bundle protocol) [45], where node-id identifies a particular dtn daemon and application-id identifies a particular application that uses the above daemon to interface to the DTN system.

Since our DT-Talkie implementation uses DTN2, so we must use "dtn:" for all EIDs. "<host>.dtn" is used as node-id and "dttalkie" is used as application-id (e.g., dtn://nokia-n810.dtn/dttalkie) to identify DTN nodes running the DT-Talkie application. Other implementations of the DTN bundle protocol might choose to use different URI schemes for bundle addressing, rather than the "dtn:" scheme. The impact of using different URI schemes on our application is minimal.

3.1.4 Bundle Routing and Delivery

The DT-Talkie on the side of user A sends the bundle in the mobile DTNs, using the services provided by the bundle protocol. We use Epidemic routing protocol to forward the bundle along the path from the source to the destination. Epidemic routing protocol is chosen because of its simplicity and higher message delivery rates. Nevertheless we will observe the behavior of DT-Talkie in the simulation environment using other DTN routing protocols in Chapter 5. After sending the bundle, it traverses through the network in the store-carry-and-forward manner, until the destination node is reached.

As soon as user B receives the bundle, it is decapsulated to extract the MIME message. The DT-Talkie then parses the MIME message to get the encoded voice message and other optional contents if any. The *Content-Type* header of the received MIME message provides information about the codec through which the voice message was encoded. The DT-Talkie then decodes the voice data using the codec information and starts playback in the audio sink (e.g., speaker) of user B. Finally appropriate action is applied to the optional contents (e.g., image is shown in the GUI).

The aforementioned approach is applicable for user B if he wants to answer user A and in this way the users can exchange several messages until the end of a voice session. When the first exchange of voice messages between user A and user B is

taken place, the DT-Talkie considers that two user are actively communicating in the session. In this active voice session, the voice messages from other users are stored locally instead of playing out immediately. So the active users are not interrupted while communication and they can playback the stored voice messages later after the active voice session is over.

3.1.5 Group Communication

The epidemic routing protocol which we use primarily in the DT-Talkie one-to-one communication, is a perfect fit for asynchronous style of group communication. Group communication may be applicable in many potential DTN applications where mobile nodes are required to collaborate closely in the infrastructure-less environment. For example, in the disaster situations, a rescue worker wants to inform other workers of a group about the current local condition through sending voice messages.

To perform group communication in the DT-Talkie, the same concept of one-to-one communication is applied with the exception that the voice messages are destined to a multicast EID. We define the structure of multicast EID as dtn://<group-name>.dtn/dttalkie (e.g., dtn://netlab.dtn/dttalkie), which is equivalent to singleton EID. If the DT-Talkie enabled users want to receive voice messages from a particular group, they must register with the corresponding EID.

3.2 Voice Message Fragmentation

In some scenarios it is not always good idea to send large voice messages in one transmission. As we mentioned before that the DTN bundles are variable in length, so the bundle size may vary from kilobytes to megabytes and even gigabytes. Generally large messages lead to longer transfer times and the contact duration in the opportunistic DTN environment may be too short to reliably transmit a large single message. This implies splitting a large message into smaller pieces through fragmentation to enable communication over short-lived links.

In the usual one-to-one DT-Talkie communication, users listen to a received voice message and attempt to record another voice message for the next transmission. So to get the next voice message, the receiver used to wait for a while (sender's listening time + sender's recording time + transmission time). For example, if user B received

a voice message from User A which he listen for 1 minute and the duration of the new recorded voice message is 2 minutes, so User A will have to wait for 3 minutes plus transmission time to listen the new voice message sent from User B. This might be feasible in the scenarios where the network suffers from poor connectivity (e.g., high end-to-end delay). But link connectivity may get improved while ongoing DT-Talkie session. In such cases it might make sense to split up a large voice message into smaller fragments and send them over different bundles. Thus we can speed-up the session interactivity. In this thesis the concept of fragmentation is employed in the application layer with the assumption of well link connectivity. Defining mechanism to dynamically adapt to the link conditions is beyond the scope of this thesis.

3.2.1 Different Fragmentation Schemes

It is worthwhile to know the ways a voice message can be fragmented and which one is feasible to use for the DT-Talkie scenarios. First, as it is discussed earlier that DTN includes the support of proactive fragmentation which allows division of application data into multiple smaller fragments and transmit each fragment as an independent bundle. The destination DTN node is responsible for extracting the fragments from incoming bundles and reassembling them into original bundle. Then the DTN node hands the original bundle to the application. Enabling proactive fragmentation in the DTN layer does not help to step up the speed of the DT-Talkie session interactivity, because the receiving DT-Talkie application gets the full voice messages to playback from its DTN node as they are sent. From the DT-Talkie applications point of view, this approach is not different than the one at which voice messages are sent and received without enabling fragmentation in the DTN node.

Second, the application layer can take the responsibility to split up a voice message into multiple fixed-sized fragments and send them as different bundles. This is a better approach than the previous one in the context of session interactivity speed. In this scheme, the receiving DT-Talkie application plays out all the fragments as soon as they are received rather than playing out the full voice message. Thus the session can be made more interactive. But this approach is not well-suited in some scenarios. Basically each voice message contains sequence of variable-length talk-spurts (sentences) and silence periods (Figure 3.3). In this fragmentation scheme, each fragment may contain fraction of a voice message and any loss of fragments may lead unmeaningful communication.

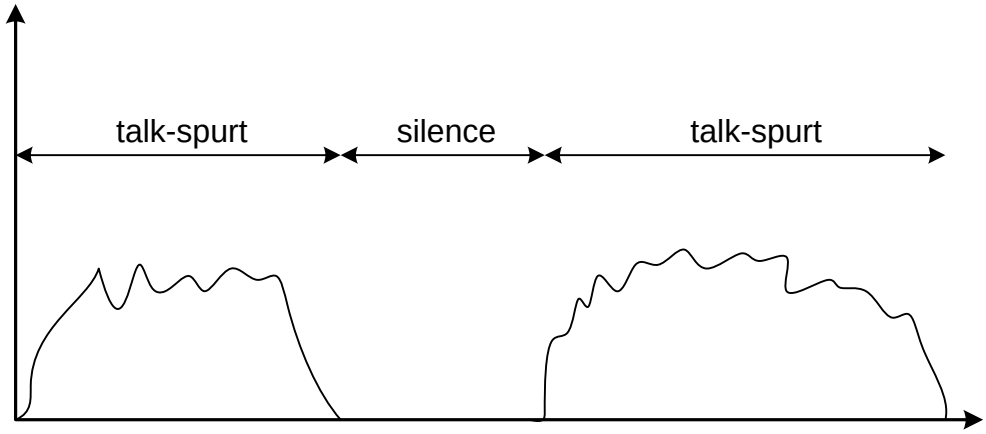

Figure 3.3: Signal representation of a voice message

Third and last, variable-length meaningful fragments can be generated through separating the talk-spurts from the voice message. This mechanism provides more interactive communication and keeps the communication meaningful to the users in case of loss of fragments. We suggest the last approach in this thesis.

3.2.2 MIME Encapsulation of a Fragment

Each fragment is required to carry additional metadata along with original data. We encapsulate each fragment in different MIME messages with three extra headers, which include *X-Msg-No*, *X-Frag-No* and *X-Last-Frag*. The headers *X-Msg-No* and *X-Frag-No* are monotonically increasing decimal number. The value of the *X-Last-Frag* header is true for last fragment and false for other fragments of a particular voice message. Figure 3.4 represents a MIME message in which the value of the *X-Bundle-Type* header is set to fragment. This means the MIME message carries a voice message fragment. By observing three additional headers, it can be said clearly that the fragment is the third fragment (last fragment as well) of the first voice message.

Addition of metadata allows the receiving DT-Talkie application to play out the fragments in correct order and do some treatment if disorderliness in the received fragments is encountered. For example, User A sends a voice message as three fragments to User B. User B receives the first fragment and it is played out immediately. If third fragment receives before second fragment, third fragment is stored in a queue. When second fragment arrives, the application plays the fragment out and finally the third fragment is played out. Moreover in Chapter 5, we carry out

```
X-Bundle-Destination: singleton
X-Bundle-Type: fragment
X-Msg-No: 1
X-Frag-No: 3
X-Last-Frag: true
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="=-UL99WySTLs0AAPGPLgtN"

--=-UL99WySTLs0AAPGPLgtN
Content-Type: audio/G729

1Í□□GŸTS§QâXJqßÚÑ□‰saÍ^cÃÝÛ”Gl/Œ ^Åˆ□□Øð[;a□öÁÆ ’ÿ□hT©—
_ŽBÃT^li˜Ov)ÿQäuúˆ+!š¥öùÙ"=8□A€¤òn={Tf□å:\yðâÔû□kÈú{äýIsßm \
.....
--=-UL99WySTLs0AAPGPLgtN—
```

Figure 3.4: MIME encapsulation of a voice message fragment

some simulations to observe the characteristics of the DT-Talkie in fragmentation mode under various circumstances.

3.3 Codec Interoperability Issues

Enabling codec interoperability is one of the most considerable issues in the context of the DT-Talkie. In the traditional voice communication where stable end-to-end connectivity is assumed, a session is established through negotiating codec parameters prior to voice data transfer, which requires round-trip exchange of several messages. We do not assume this due to avoid unnecessary round-trips as they can be costly in the challenged scenarios. On the other hand, while heterogeneous endpoints participate in the DT-Talkie session, some of them may not have support of a particular codec[1]. So there would be a chance that voice messages are left non-played. In order to deal with those issues, we must define some approaches that make the DT-Talkie codec interoperable without exchanging some extra messages.

However we imply two mechanisms in this thesis to maximize codec interoperability of the DT-Talkie. We explain the mechanisms using a one-to-one communication scenario where user A and user B are communicating in a voice session. First, as user A does not have any idea about which codecs are supported by user B, he can

[1]Since G.729 or G.723.1 is a proprietary codec for example, it may not be available in some open source platforms.

encode voice messages using all of his supported codecs and send them with MIME message encapsulation. Figure 3.5 depicts an example MIME message which can be used in this mechanism. The MIME message contains three different body parts of type G729, MP3 and Speex, which are used to encode the same voice message. We suppose that user B has a support for the G729 codec[1]. So after receiving the MIME message, user B can pick up the G729 audio part and then playback. If user B wants to send a voice message as a reply to user A, he use the same G729 codec to encode the voice message. After receiving the reply message, user A now have the knowledge of G729 codec which is supported in the side of user B and all the remaining voice messages in the session will be transferred using the G729 codec. The downside of this approach is that if the recipient does not have support of any codecs which were used at the sender's side, the sent voice message is missed to playback and the session is no longer continued.

Second and finally, users can agree upon a common codec through first exchange of voice messages between each other. In the first exchange, voice messages can be transferred in the uncompressed PCM format (e.g., WAV), as it is supported by a wide variety of platforms. For example, if user A attempts to communicate with user B, he first sends an uncompressed voice message in conjunction with a list of his supported codecs, which can be encapsulated in a MIME message. Figure 3.6 shows a MIME message of this kind, which contains a voice message in WAV format and XML [46] content[2] to represent a list of supported codecs. In the XML content, the *supportedCodecsList* is the root element and the *codec* element is the child of the root element. The codec element has two attributes: *name* and *type* which carry the name and the mime type of a codec respectively. However in reply, user B also sends a voice message to user A in the uncompressed PCM format along with his list of supported codecs. In this first exchange between user A and user B, they negotiate on a common codec and in the later exchanges all the voice messages are encoded using the negotiated codec. If they do not find any common codec, the session can be continued through transporting uncompressed PCM voice messages in the later exchanges. In this approach the session does not break up because of unsupported codecs, which is seen as more advantageous over the previous approach.

[1]User B may support other codecs, but he always extract one suitable audio part from the MIME message.

[2]We do not mandate to enlist the codecs using XML format. Any structured representation could be used.

```
X-Bundle-Destination: singleton
X-Bundle-Type: message
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="=-UL99WySTLs0AAPGPLgtN"

--=-UL99WySTLs0AAPGPLgtN
Content-Type: audio/G729

1Í□□GŸTS§QâXJqßÚÑ□‰saÍ^cÃÝÛ”Gl/Œ ^Å^□□Øð[;a□öÁÆ 'ÿ□hT©—
_ŽBÃT^li˜Ov)ÿQäuú^+!š¥öùÙ"=8□A€¤òn={Tf□å:\yðâÔû□kÈú{äýIsßm \
.....
--=-UL99WySTLs0AAPGPLgtN—
Content-Type: audio/mpeg

□Ê□ŒâÒ(□□´èUf½@3Fj□x,E…,UAÁÀÆ□©,âÐ¡©(AÑ'Io‹Òd˜BY´‰œ9Œ¹°ØõH□Sà□¥›
Ä^D9X;~ je.□□□□Pß€yŒIŠ0°PI~Xò4˜ □□,«□Mf^†Ì□a[#=^1T□ýL[
.....
--=-UL99WySTLs0AAPGPLgtN
Content-Type: audio/x-speex

Û□eÿÿÿÿÿ‰□ò□í¼tÊU¸_="¾¶@>V—ê®¶——(Ã
Fƒ□ó□8)IsÎ¿3³uÔþ«.üÝØ½[¬Ç¡Žäú¥□ Z§(Ã
»„□òü™óÐý=ùY¸pê□Î¦|¢3õ€cQí~}Ñ□žÿ§(ÝýdÅ¸v}Fïî□yó:¨éürëª□|"s#□ù
.....
--=-UL99WySTLs0AAPGPLgtN
```

Figure 3.5: MIME encapsulation of three audio parts representing the same voice message

3.4 Summary

In this chapter, we have presented the overall concepts of the thesis. The system architecture of the DT-Talkie is discussed comprehensively using a one-to-one communication scenario. Basically captured voice messages are encapsulated in the bundles and sent them over DTN infrastructure. There might have issues regarding codecs support to play out the received voice messages, since we do not use any negotiation mechanism. However adding support of more codecs can be considered as a remedy. We have defined MIME framing technique which allows other contents to be sent with voice messages. Group communication is carried out in the DT-Talkie using the ideas of one-to-one communication with very few exceptions.

Voice message fragmentation has been suggested in the well-connected scenarios in order to speed-up the session interactivity. We have introduced different fragmenta-

```
X-Bundle-Destination: singleton
X-Bundle-Type: message
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="=-UL99WySTLs0AAPGPLgtN"

--=-UL99WySTLs0AAPGPLgtN
Content-Type: audio/x-wav

RIFF$ ÿWAVEfmt      @  €>    data  ÿ    C O 7 @ : > D A E < = 6
? ; : 8 : < < > @ > 9 7 6 < : : 7 5 3 2 0 3 5 ; ; < 9 9 9 7 8 7
...
--=-UL99WySTLs0AAPGPLgtN—
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<supportedCodecsList>
       <codec name="MP3" type="audio/mpeg"/>
       <codec name="G729" type="audio/mpeg"/>
       <codec name="Speex" type="audio/x-speex"/>
       .....
</supportedCodecsList>
--=-UL99WySTLs0AAPGPLgtN
```

Figure 3.6: MIME encapsulation of uncompressed audio and XML content

tion schemes. The chosen fragmentation mechanism separates talk-spurts through detecting silences in a voice message and sends them as fragments. We have also talked about two approaches to negotiate codec between endpoints without the cost of some extra round-trip message exchanges.

Theoretical work that is presented in this chapter has little significance unless it is realized in practice. The concepts discussed here serve as a basis for the next chapter, which provides implementation details of the DT-Talkie.

4 Implementation

After presenting the conceptual framework in the previous chapter, we now pay our attention to the implementation of the DT-Talkie. Our goal is to produce the fully functional delay-tolerant push-to-talk application that can be used for small-scale deployments. The aim is not to create production-grade software, but a proof of concept to get practical realization of the ideas presented in the previous chapter.

The DT-Talkie application is intended to be implemented on Linux and an open source based mobile computing platform. We integrate Graphical User Interface (GUI) to the application with the vision to improve the end-user experience. DTN reference implementation (DTN2) is used to get the bundle protocol services and the DT-Talkie is realized as a separate application program.

In this chapter, we briefly introduce our intended software platform and the technologies which form the foundation of the implementation. The high-level application architecture and the implementation steps of the DT-Talkie functional components are also discussed in this chapter. The intention is to present the implementation with sufficient detail so that this chapter, the source code and its inline comments will allow the readers to understand how the application works and how it could be further developed. Then we give an implementation overview of the DT-Talkie fragmentation mechanism and finally the graphical elements of the application user interface are discussed.

4.1 Background

Before approaching to the in-depth discussion of the DT-Talkie application architecture, it is worthwhile to have some background knowledge about the application development platform and other dependent technologies, which are introduced in the following subsections.

4.1.1 Maemo

Maemo [47] is a software development platform provided by Nokia for their series of Internet Tablets (e.g., N800, N810 and N900). Maemo is based on Debian GNU/Linux operating system kernel. The Linux kernel is the central software com-

ponent of the system. It builds the abstraction layer for the system devices and provides memory management, process management, networking services, file management and various other services. Maemo devices run the recent Linux 2.6 kernel. The ARM/OMAP-based Linux kernel of the Maemo platform implements hardware-specific device and bus drivers on top of the core kernel's virtual services.

The user interface is based on XWindow System with the Matchbox window manager. The application programming interface (API) on top of XWindow is a GTK+ widget toolkit with the Hildon extensions. GNOME is an application framework for desktop Linux systems. From the GNOME project, maemo has inherited many central components, such as the GTK+ GUI toolkit, the GStreamer multimedia framework, the GConf application configuration management, the GnomeVFS, and the XML library.

The Hildon framework provides components on top of the GNOME components to support stylus-based usage, high display pixel density, hardware buttons, control panel, status bar, task navigator, and home applets. It also provides the backup/restore framework, the help framework, and an application manager.

The debian-oriented scratchbox cross compilation toolkit [48] provides a sandbox environment for Maemo application development. Scratchbox-compatible rootstraps, which contain all the development libraries and header files required for application development, are available for both x86 and ARMEL (ARM emulator). This enables almost all development and debugging tasks on the x86 PC host environment, with final validating and packaging being for ARMEL.

Maemo is built around the debian packaging system and mainstream Debian tools that provide the software package management infrastructure. This also facilitates easy updating of the development environment when new versions of the software components are made available. Maemo application programming interfaces (APIs) are natively supported for the C language. Maemo also has C++ and Python bindings for its core APIs. Unofficial bindings for other languages and environments are also available.

4.1.2 GTK+

GTK+ (GIMP Toolkit) [49] is a cross-platform widget toolkit for creating graphical user interfaces. It is called GIMP toolkit because it was originally designed for a raster graphics editor called the GNU Image Manipulation Program (GIMP). GTK+ was adopted as the default graphical toolkit of GNOME and Xfce, two of the most popular Linux desktop environments. While it was originally used on the Linux operating system, GTK+ has been expanded to support Microsoft Windows and other UNIX-like operating systems: Solaris, Mac OS X, BeOS and others. GTK+ is written entirely in C, and the majority of GTK+ software is also written in C. There are GTK+ bindings for many other languages including C++, Python, PHP, Ruby, Perl, C#, and Java.

GTK+ is fully object oriented although written in C. It uses classes and callback functions implemented as structures and pointers to functions. GTK+ is built on top of a number of other libraries such as GLIb, GObject, Pango, ATK, GDK and Xlib. *Glib* provides low-level data structures, types, threads, an event loop, an object system and dynamic loading. *GObject* implements an object-oriented system in C without requiring C++. *Pango* provides layout and rendering of internationalized text. *ATK* supports screen readers and alternative input devices. *GDK* performs the actual rendering to the display. It abstracts from the display so that it can run on top of X11, Win32, or Cocoa. *Xlib* provides the low-level graphics functionalities on Linux and UNIX systems.

4.1.3 GStreamer

GStreamer [50] is a cross-platform multimedia framework for developing streaming multimedia applications. The GStreamer framework is designed to facilitate building applications that handle audio or video or any kind of data. One of the most obvious uses of GStreamer is to build a media player. GStreamer supports a very wide variety of formats, including MP3, Ogg/Vorbis, MPEG-1/2, AVI, Quicktime, mod, and more. The framework is based on plugins that provides the various codecs and other functionalities. The plugins can be linked and arranged in a pipeline which defines the flow of the data.

GStreamer has been ported to a wide range of operating systems which include

Linux (x86, PowerPC and ARM), Solaris (Intel and SPARC), Mac OS X, Microsoft Windows and OS/400. The GStreamer core library is written in the C programming language with the type system based on GObject. It has bindings for programming-languages like Python, C++, Perl, GNU Guile and Ruby.

The most important component in GStreamer is the *element*. Each element is provided by a plugin. Elements communicate by means of *pads* which can be viewed as plugs or ports. Pads are used for linking one element to other elements through which data can flow to or from those elements. Normally they have specific data manipulation capabilities and can restrict the data types that can pass through it.

Source elements generate data, for example reading from disk or from a sound card. Sink elements accept data but do not produce anything, for example disk writing, soundcard playback, and video output. Filters and filter-like elements have both input and output pads. They receive data on their input (sink) pads, and will provide data on their output (source) pads. A bin works as a container where elements can be grouped together. A pipeline is a higher level bin. When the pipeline is in the playing state, data flow starts between the pads of the elements.

4.1.4 DTN2

DTN2 is the reference implementation of the DTN bundle protocol developed by DTNRG. It is designed with the vision to provide the platform for researchers to do experiments and evaluate the protocol designs. DTN2 also aims to be high-quality, production-ready code for real-world deployments [51].

DTN2 includes support for TCP, UDP, Bluetooth, Ethernet, Sneakernet and External convergence layers. DTN2 supports several link types such as always available links, on-demand links, opportunistic links and links with scheduled contacts. There are several routing schemes implemented in DTN2, such as Static, Flooding, Neighborhood, Delay-Tolerant Link State Routing, PRoPHET. For new routing schemes, DTN2 has a XML based External router interface. Persistent storage is used to store bundles, network state information (e.g. routing tables) and application state information (e.g. registrations). DTN2 has the support for BerkeleyDB, MySQL, PostgreSQL, file system and memory to provide persistent storage services. For

neighbor discovery, DTN2 adds support for Bluetooth, IP and Bonjour discovery mechanisms.

DTN2 is written primarily in C++ and ported to Linux, Solaris, Win32 (Cygwin), Linux on PDA (ARM), FreeBSD and Mac OS X. It provides APIs which enable applications to access the bundle protocol services. There are also other implementations of the bundle protocol exists such as IBR-DTN [52], Java BP-RI [53] and DASM [54], but these focus on embedded systems, Java and Symbian respectively and do not provide as much functionality and are not as mature as DTN2.

4.1.5 Other Libraries

GMime library provides mechanism for creating, editing and parsing the Multipurpose Internet Mail Extension (MIME) messages and structures. It is built upon GLibs GObject system allowing for a lot of flexibility.

Hildon library allows creating widgets and themes specific to the Maemo. It is primarily a set of GTK+ extensions which focuses on providing a finger-friendly interface and other mobile device oriented functionalities.

4.2 DT-Talkie Application Architecture and its components

Four major functional components serve as a basis for the DT-Talkie application. They are GUI, Voice R/P (Recorder/Player), MIME C/P (Creator/Parser) and Bundle S/R (Sender/Receiver). In Figure 4.1, the high-level overview of the application architecture is represented.

GTK+ and Hildon are used in the GUI component to render the application user interface. The Voice R/P component relies on GStreamer to record and playback voice messages. GMime library is used by the MIME C/P component to carry out creating and parsing the MIME messages. The Bundle S/R component is dependent on DTN2 (Version 2.6.0) in order to send and receive bundles. The Maemo platform provides GTK+, Hildon and GStreamer libraries by default. We just port GMime and DTN2 libraries to the Maemo platform. The implementation of all the functional components is described in detail later in this section.

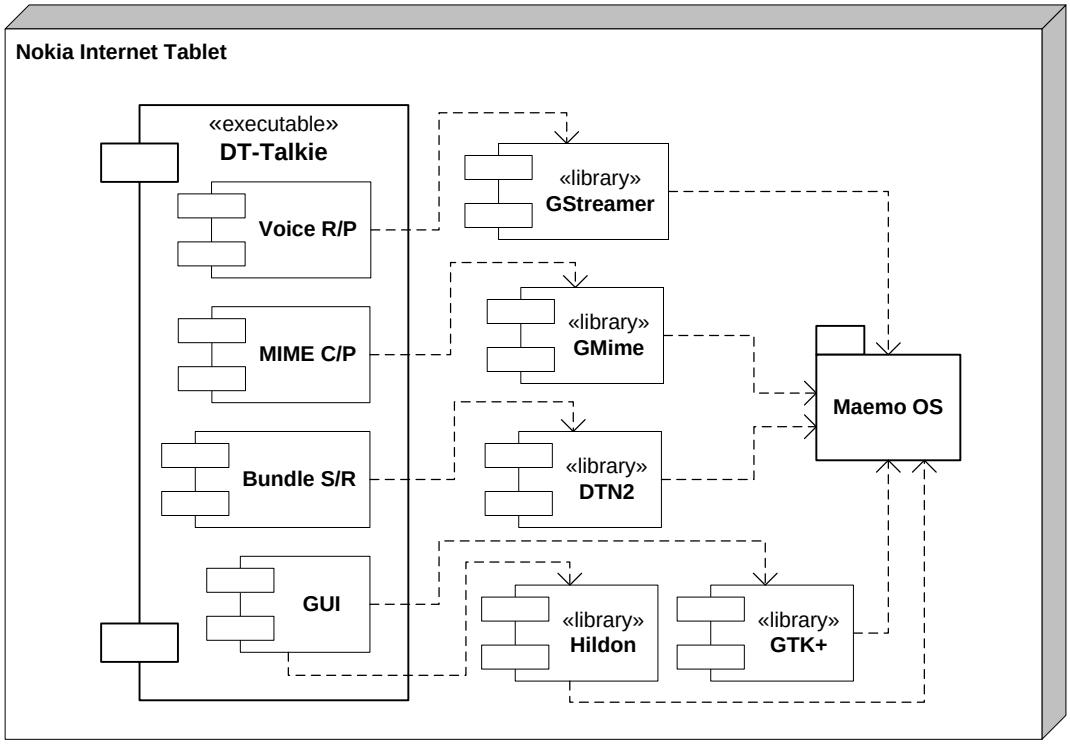

Figure 4.1: High-level architecture of the DT-Talkie application

4.2.1 State flow of the DT-Talkie application

Figure 4.2 shows the state flow diagram of the DT-Talkie application. In the application we define a structure that holds the most important data. The data reflects the current state of the DT-Talkie application and propagates to different methods. After initializing the application state data, a thread is spawned to receive new bundles. Then GTK+ main loop starts running in which the main window along with other widgets are rendered and the handlers process the input when users interact with the widgets using stylus and presses the hard keys. Before starting voice recording a user selects an EID of the destination (voice messages can be destined to either individual user or a group of users) from the application GUI. The EID list of individual users and groups of users are pre-stored in the local filesystem. However voice recording starts when the user presses a hard key[1] for the first time and it continues in a GStreamer loop[2]. The recording is stopped by pressing the hard key

[1]We use full screen hard key of the Nokia Internet Tablets.

[2]The GStreamer loop is basically a GLib event loop, which is used for the purpose of getting messages, errors, and other important information while running.

for the second time. The recorded voice message and an image[1] are aggregated in a MIME message. The MIME message is then encapsulated in a bundle which is sent in the mobile DTNs.

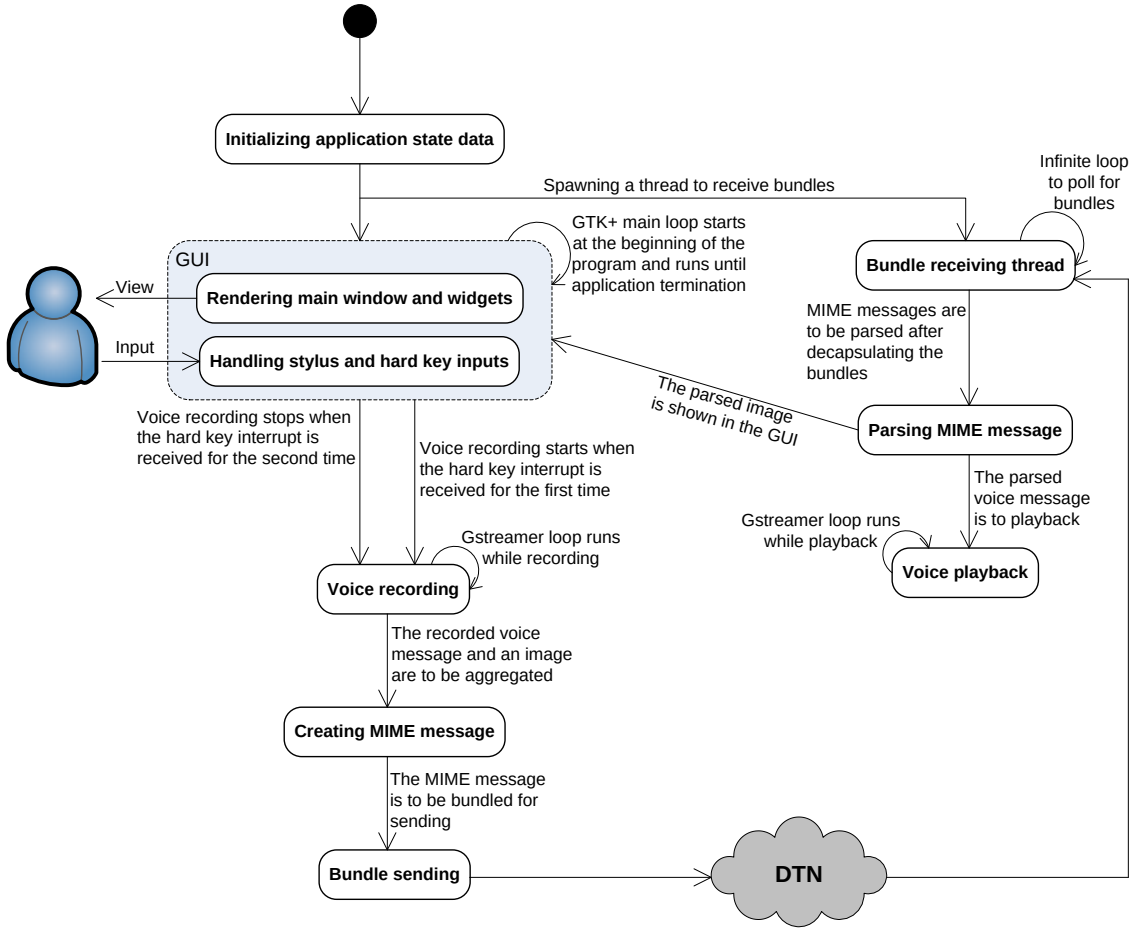

Figure 4.2: DT-Talkie application state flow diagram

On the other hand, the bundle receiver thread continuously polls for new bundles in an infinite loop. As soon as a bundle is received, it is decapsulated to get the MIME message. The extracted MIME message is then parsed to get the voice message which is played out in another GStreamer loop. The parsed image is also shown in the GUI.

[1]In the current implementation we only aggregate users profile picture together with voice messages, not other contents.

4.2.2 Bundle S/R

The Bundle S/R component consists of two modules: bundle sender and bundle receiver. Both modules connect to the DTN daemon (dtnd). dtnd is a background process that provides the actual communication of bundles between hosts. It must be run on a host in order to send, receive, or forward bundles. The processing steps of the bundle sender and bundle receiver modules are discussed as follows.

Bundle Sender

Figure 4.3 illustrates the sequential execution of dtnd APIs when the bundle sender is called for sending bundles. In order to send a bundle, first the bundle sender invokes `dtn_open()` method to open a new connection to the running dtnd daemon, which initializes a new handle to the daemon on success. Then the bundle (which we attempt to send) headers are populated using the bundle spec structure. This structure is initialized with destination, source and replyto EIDs, bundle priority, expiration time and delivery options. `dtn://host.dtn/path`

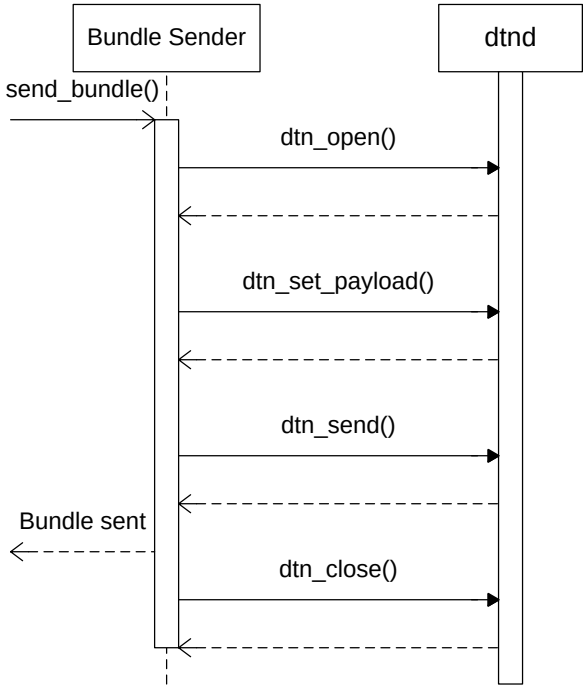

Figure 4.3: Sequence diagram of bundle sending functionality

Then bundle payload structure is populated with the invocation of the `dtn_set_payload()` interface in order to send a bundle. The payload contents can be copied either from memory or from file. In this case we use local file and specify the filename in the interface as a parameter. After this step the actual transmission process is accomplished with the `dtn_send()` interface, which takes the opened handle and the prepared payload structure as parameters. If the bundle sending succeeds, the bundle sender module reports an info message in the GUI about success of bundle transmission. Finally `dtn_close()` method is called to close the opened handle.

Bundle Receiver

The bundle receiver module executes in a different thread, which is spawned at the beginning of the program. The execution steps of the bundle receiver are illustrated in Figure 4.4. Like the bundle sender, a new handle is initialized to open a connection to the running dtnd daemon. Now the registration to the local DTN node needs to be performed in order to receive bundles to a particular EID. To achieve this, the registration info structure is set with EID[1], registration expiration time and flag. Then `dtn_register()` method is used to create registration in the active state and bind to the opened handle.

We use the `dtn_recv()` interface that is called periodically in an infinite loop to get new bundles from the bundle router. The bundle spec and the bundle payload structures are filled with the bundle data using the interface when a bundle is received successfully. Then the bundle is further processed by decapsulating to extract the MIME message which is parsed to separate the voice message for playing out and the image for showing in the GUI. Finally the payload structure is freed from memory to avoid memory leak. The opened handle is closed when the application terminates.

4.2.3 Voice R/P

The Voice R/P module comprises of Voice Recorder and Voice Player modules. Even though the DT-Talkie application supports voice recording in MP3, G729 and WAV coding techniques, we discuss the processing steps of the voice recorder and the voice player modules using an example of MP3 codec. `gst_init()` interface is in-

[1]The EID is specified in the DT-Talkie configuration file.

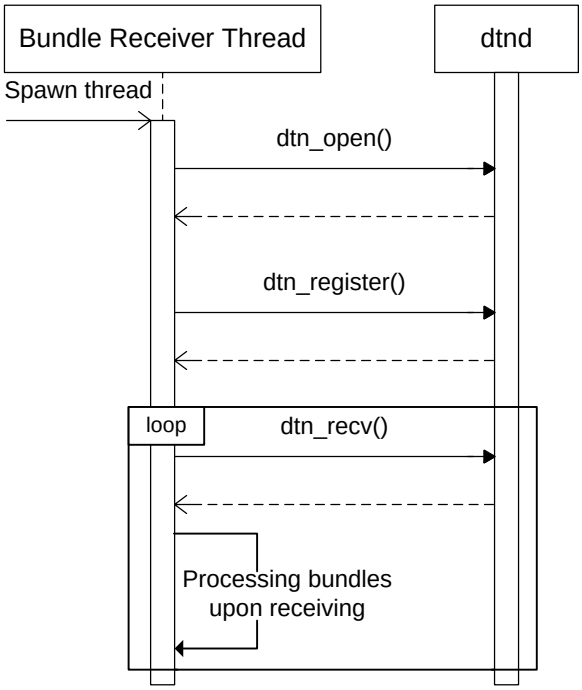

Figure 4.4: Sequence diagram of bundle receiving functionality

voked at the beginning of the program to perform the necessary initialization of the GStreamer library as well as parses the GStreamer-specific command line options. Two GStreamer loops are created which run while voice recording and playback. We present individual discussion of both modules in the following.

Voice Recorder

In the Voice Recorder module, a new GStreamer pipeline is created and a message handler is attached to the pipeline bus. When a message (e.g., end-of-signal, error) is posted on the bus, the default main context calls the handler to perform action on the message.

Three different elements are created which include source, coder and sink. GStreamer must have the support of plugins which are used for creating elements[1]. We use *dsp-pcmsrc* plugin for creating source element to receive raw PCM audio from the DSP device. The coder element is created with *lame* plugin to translate the raw audio into MP3 format. Then *filesink* element is used in order to create the sink element

[1]The availability of a specific plugin can be queried through *gst-inspect tool*.

for writing the encoded audio data into a local file.

After creating three GStreamer elements, all of them are added into the pipeline and linked together. Then the pipeline is set to "playing" state and the GStreamer loop starts running. This time the voice recorder pipeline captures raw voice data from the user, encodes and saves them in a file. Figure 4.5 depicts the voice recorder pipeline. The loop continues unless any explicit call to quit the loop takes place. Upon termination of the loop, the pipeline is set to "null" state and the reference of the pipeline is cleaned up.

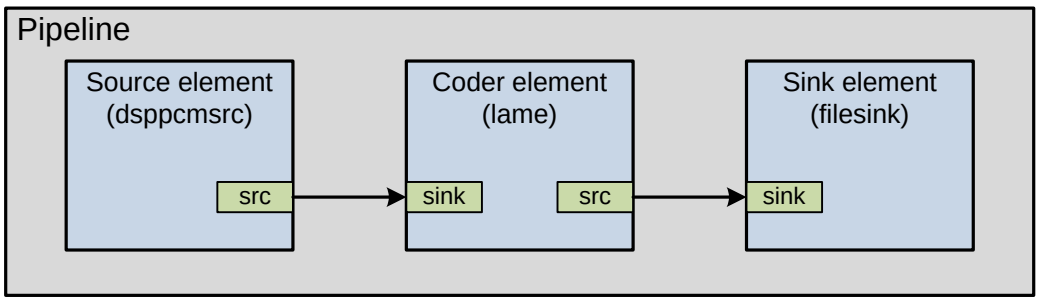

Figure 4.5: Voice recorder pipeline

Voice Player

The processing steps in the voice player module work the same way as the voice recorder. But the source, coder and sink elements are created in different way. In this case *filesrc* plugin is used for source element to receive audio data in MP3 format from a local file. MP3 audio is decoded into raw PCM audio data using the *mad* plugin in the coder element. Then the sink element with *dsppcmsink* plugin transfers the raw audio stream to the DSP device for playing out the audio. Figure 4.6 shows the pipeline used for the voice player.

4.2.4 MIME C/P

MIME Creator and MIME Parser are the two modules of the MIME C/P component. At the beginning of each module, `g_mime_init()` method is called to initialize the GMime library. GMime file based stream object is created in both modules to read and write MIME messages. In the following we discuss the functional steps of the Mime Creator and the Mime Parser separately.

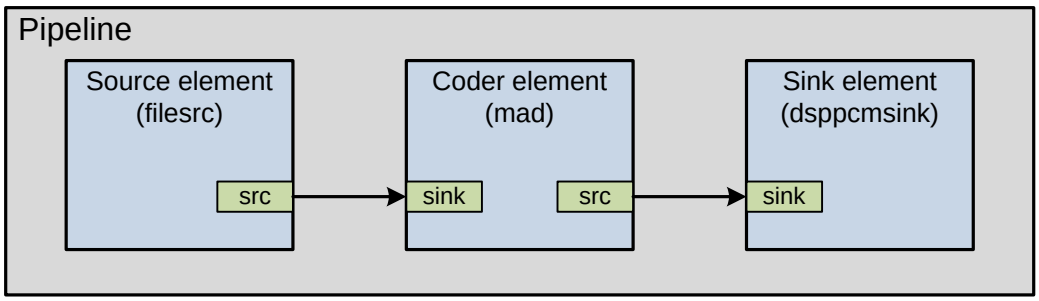

Figure 4.6: Voice player pipeline

MIME Creator

The MIME creator module creates a new MIME message object which is initially empty. On the MIME message object, we set a header named *X-Bundle-Destination* with the value either *singleton* if the communication is one-to-one or *multinode* in case of group communication. After this step, an empty MIME multipart object is created with a default content-type of *multipart/mixed* in order to fill the body of the MIME message. The multipart object can contain multiple parts with different types of contents. For this implementation we only create two empty MIME parts; one with the content-type of *audio/G729* to carry G729 encoded voice messages[1] and another with *image/jpeg* to carry profile picture in JPEG format. Now we populate the two parts with recorded voice and profile image data without applying any content encoding mechanism. Then the two parts are added to the multipart object and the multipart object is set to the message object. Finally the message object is written to the file based stream.

MIME Parser

The MIME parser module creates a new parser object preset to parse the file based stream. Then it constructs a MIME message from the parser, which contains headers and a multipart body. Through extracting the *X-Bundle-Destination* header of the message, the application can get the idea whether a voice message is intended for a single user or for group users. Then the module iterates through all the parts of

[1]The content-type can be *audio/mpeg* for MP3 or *audio/x-wav* for WAV encoded voice messages.

the message, get the audio and image contents and saves them in separate files for further processing.

4.2.5 GUI

At the beginning of the GUI module, all initialization work is done by calling `gtk_main()` to use GTK+ and all of its supporting libraries. It begins by setting up the GTK+ environment, including obtaining the GDK display and preparing the GLib main event loop and basic signal handling. Even though most of the GTK+ widgets[1] work seamlessly in the Maemo GUI environment, we create `HildonProgram` and `HildonWindow` widgets to fit the DT-Talkie application into the environment. Any Hildon application is based on the `HildonProgram`, which is inherited from GObject and represents an application running in the Hildon framework. The `HildonWindow` is inherited from the `GtkWindow` which represents a top-level window of an application running in the Hildon framework.

Then some basic GTK+ widgets, such as button, label, list, frame, scrollbar and text entry, are created over the hildon window. To organize the basic widgets, we use specialized non visible widgets called layout containers. These containers work as parent widgets. In the GUI module a table widget is used to organize child widgets in rows and columns. We also use vertical box and horizontal box widgets to pack child widgets into a single column and a single row respectively.

Basically all GTK+ GUI applications are event-driven in nature. In GTK+ an event is a message from the X Window System. When a user performs some action like clicking a mouse or typing a keyboard, an event is fired. When the event reaches a widget, it reacts to the event by emitting a signal. In this module specific signals are caught from the application and connected them to callback functions for performing particular actions based on the type of the signals. In the DT-Talkie application, recording and sending voice messages are carried out through two consecutive presses of the fullscreen hard key of the Nokia Internet Tablets. So we also implement a callback function to handle the signal which is emitted when the hard key is pressed.

Finally `gtk_main()` is called to start a main loop, which continuously checks for

[1]Widgets are basic building blocks of a GUI application. For example: button, check box or scrollbar.

newly generated events. The loop ends when the application is terminated.

4.3 DT-Talkie Fragmentation

We have discussed earlier the fragmentation approach that is applied in this thesis. In the approach fragments are generated through separating the talk-spurts from a voice message and sent them as different bundles. In this section we discuss how the implementation of the talk-spurts separation technique is carried out. We implement the technique in a separate module for testing purposes. Since this implementation is not yet matured, currently we do not integrate the implementation to the mainstream DT-Talkie application.

Talk-spurts Separation Mechanism

Silence periods in the voice message are considered as markers in the process of separating talk-spurts from the voice message. We use Voice Activity Detection (VAD) technique to detect silence periods in the voice message. VAD is a mechanism which is used to detect presence and absence of human speech in an audio signal. In VoIP and mobile telephony applications, VAD plays an important role to reduce bandwidth usage and network traffic by transmitting audio packets only if speech is detected. We do not incorporate any new VAD algorithm into our system. Rather the built-in VAD feature of the G.729 codec is used to meet our goal. DT-Talkie only provides fragmentation support in the one-to-one communication scenario and when the G.729 codec is enabled. Figure 4.7 illustrates how the talk-spurts are extracted while recording the voice message and sent as fragments. In DT-Talkie with fragmentation mode, voice data is stored as soon as voice capturing starts. If any silence is detected DT-Talkie stops storing voice data and considers the previously stored voice data as a talk-spurt (talk-spurt 1). Just before starting of another talk-spurt (talk-spurt 2) after the silence period, talk-spurt 1 is encapsulated to send as a bundle. This process continues unless the user halts the recording explicitly.

Considering the length of silence period between two talk-spurts is a significant aspect to concern. The application has no idea when the next talk-spurt will start after detecting the silence and which part of the voice data will be considered as a talk-spurt. In a recent study [55] the authors, after investigating the characteristics

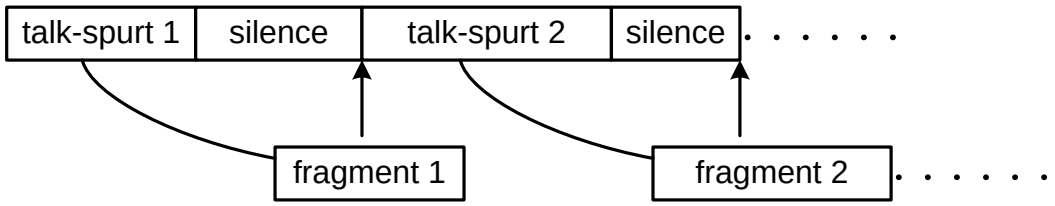

Figure 4.7: Fragments creation through separating talk-spurts

of packetized voice stream for different codecs, have figured out that the mean silence duration for G.729 codec[1] is 1.5621 seconds. Based on their observation we use 1.5 seconds as threshold for the silence duration. This means talk spurts, which start after the silence period of length greater than the threshold value, are treated as different.

4.4 DT-Talkie Screenshot

In order to provide decent experience to the users, we implement an intuitive and user- friendly GUI for the DT-Talkie application. The screenshot of the application, while running in the Nokia N810 device, is shown in Figure 4.8.

The top-level window mainly consists of four frames, which include Individual Contacts, Group Contacts, Pending List and Notification Box. The Individual Contacts frame contains a list of singleton EIDs that are used to transport bundles in the one-to-one communication scenario. The list of multicast EIDs resides in the Group Contacts frame, which are used to send bundles towards a group of users. The "Join" and "Leave" buttons beneath the Group Contacts frame are used for joining to and leaving from a specific group. Basically the application registers and unregisters the specified multicast EID with the local DTN node, when joining and leaving operations are triggered. The user can traverse through both contact lists using the "Increase/Decrease" hard key of the Nokia Internet Tablets. In between the contact lists there are a text entry box to input an EID, an add-up button to add the EID to the individual contact list and an add-down button to insert the EID in the group contact list.

Usually the voice messages are immediately played out as soon as they are received and deleted after playing out so that the resource constrained Internet tablets do

[1]We just pick the result for G.729 codec among others.

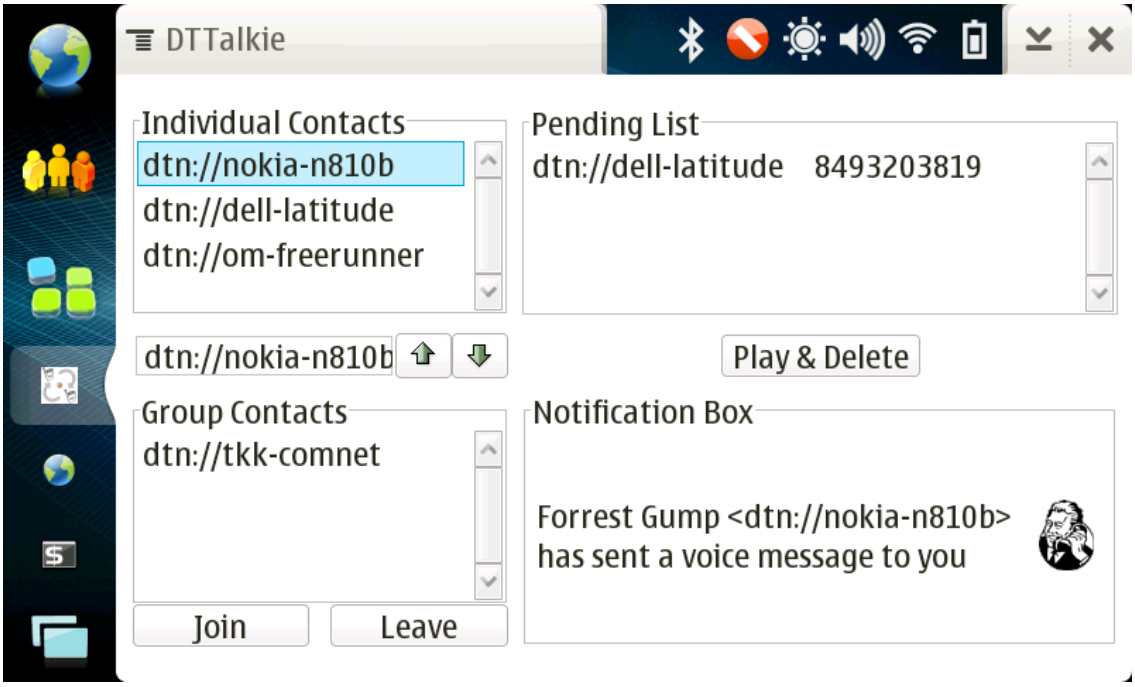

Figure 4.8: Screenshot of the DT-Talkie application

not run out of storage. The intention of adding the Pending List frame in the main window is to list all the non played voice messages. This means when two users are actively communicating and a third user sends a voice message to any of the active users; the voice message is enlisted as pending rather than immediate playback. So the user can listen to the voice message later and delete using the "Play & Delete" button underneath the Pending List frame. The Notification Box frame is used to notify users while sending and receiving voice messages. Upon receiving a voice message, the encapsulated image (if any) with the voice message is also shown in the frame.

4.5 Summary

In this chapter, we have given a detailed description of the DT-Talkie implementation, which verifies the ideas presented in this thesis. The current state of the application is at the proof of concept level. But the application architecture is designed is such a way that opens the door to evolve our application to a level as production-grade software through further development.

The DT-Talkie application relies on DTN2 reference implementation to get the Bundle Protocol (BP) services. But any BP implementation that provides the same services can be used. Our application supports three codecs currently. Integration of more codecs support could increase the codec interoperability of the application.

It is worth noting that the DT-Talkie application is also executable in other Unix/Linux based platforms, such as Openmoko, Mac and Linux PC [56]. We just need to port the application to specific platform. Even though GTK+, GStreamer and GMime support Windows, but DTN2 does not - hence, our implementation cannot be run on Windows machine.

Evaluating an application is also necessary to carry out after the implementation process to observe how the application performs in different scenarios. In the next chapter, we will discuss about the performance evaluation of the DT-Talkie application through conducting a set of simulations using different routing protocols and using various mobility scenarios.

5 Performance Evaluation

After discussing the DT-Talkie implementation details in the previous chapter, we now focus on evaluating the performance of our system. To achieve this a set of simulations are carried out in different scenarios. Basically simulation is an effective way to gain better understanding the characteristics of a real system. Different movement models and different DTN routing protocols are used to conduct the simulations. Despite of having both one-to-one and group communication support in the DT-Talkie, we only consider the one-to-one communication scenario for simplicity while performing simulations.

In this chapter, we briefly describe a software-based simulation tool that we use to carry out the simulations. We give a brief overview of two mobility models including one simple model and one realistic model. Then a detailed description of the simulation setup is presented in this chapter. Finally we discuss about the results observed from the simulations.

5.1 Simulation Tool - ONE

Numerous software-based simulators exist to analyze the behavior of DTN routing and application protocols. We use the Opportunistic Network Environment (ONE) [57] simulator for our evaluation. The ONE is a highly customizable network environment which combines movement modeling, routing simulation, visualization and reporting in a single program. The core of the ONE is an agent-based discrete event simulation engine. To generate node movements, different mobility models ranging from simple models to more realistic models are implemented in the ONE. It can import mobility data from real-world traces or other mobility generators.

Six well-known DTN routing protocols are supported in the ONE, which are used to route messages between mobile nodes in the simulation environment. It allows generation of application messages inside the simulation either through message event generators or from external event files. All the messages inside the simulation world are unicast with a single source and destination host. The ONE can be extended by including new mobility models and routing protocols. Node mobility and message passing in real-time are visualized in its graphical user interface. Various reports and post-processing tools are used to collect and analyze simulation results. The ONE

simulator can also be used in conjunction with DTN2 for testing and evaluating real-world DTN applications.

5.2 Mobility Models

A mobility model governs the way nodes are moving and how their location and speed change over time. Such models play a very important role to simulate a new MANET or DTN protocol and evaluate the protocol performance. Even though there is several mobility models supported in ONE, we limit our discussion to simple RWP model and more realistic WDM model, which are used to conduct simulations in this thesis.

5.2.1 Random Waypoint Model

The Random Waypoint (RWP) [58] model is an elementary synthetic model frequently used in ad-hoc network simulations. In this model, a node chooses a random waypoint on the simulation area and moves to that waypoint with the speed drawn from a uniform distribution. When the node arrives in the destination waypoint, it pauses for random amount of time and continues in the same fashion.

5.2.2 Working Day Movement Model

The Working Day Movement (WDM) model [59] has been developed with the aim of increasing the reality of node mobility. It models typical human movement patterns during working weeks. The WDM model presents everyday life of average people by modeling three major activities: 1) being at home in the morning, 2) working at offices in the day and 3) going out for shopping in the evening. These three activities are defined in home, office and evening activity sub-models. The WDM model also employs three different transport models when nodes move between home, office and evening activity. The nodes without cars move by following walking or bus sub-model. The car sub-model defines movement of the nodes which have cars. The support of different transport models adds additional heterogeneity which has impact on the performance of e.g., routing protocols.

5.3 Simulation Setup

Inside the ONE simulation environment, mobile nodes typically move following the rules of a mobility model. The way messages are generated inside the simulation varies from application to application. A routing protocol governs the way messages are forwarded inside the simulation environment towards the destination. Various parameters related to the mobility model, the routing protocol and message generation are required to specify in the ONE configuration file before running the simulations. Before discussing simulation parameters for different scenarios, we briefly state the message generation, destination node selection procedure and the performance metrics.

5.3.1 Message Generation for ONE

We implement a message generator in ONE to produce DT-Talkie style voice messages inside the simulation environment. The messages can be either full-length voice messages (message mode) or fragments of the voice messages (fragmentation mode), depending on the input parameter in the ONE configuration file. As we discussed earlier that a voice message comprises both talk spurts and silences. In the fragmentation mode, we take the silences as marker to separate the talk spurts and consider those talk spurts as fragments. Either full-length voice messages or fragments of voice messages are always generated on the basis of duration drawn from a distribution. For example, in the simulations the generator chooses duration for a voice message. Then it divides the message duration into talk spurts and silence periods. A recent study [55] suggests that Pareto distribution can be used as an accurate model for the talk spurt and silence period. So we choose Pareto distribution to model the duration of voice messages, talk spurts and silences while generating messages.

5.3.2 Destination Node Selection

Generally we choose the destination nodes randomly anywhere from the simulation area. But the real-life observation reveals that most of the cases walkie-talkie like communications takes place at shorter distance such as in the offices or in the construction areas. This assumption is applied to some of our simulations and we observe increasing message delivery rates when destination nodes are selected from

shorter distance. To achieve this, firstly, we specify a *communication radius*[1] (e.g., 50 m) of a source node, at which messages are supposed to send to any of the nodes within that boundary. Secondly, we calculate Euclidean distance from the source node to other nodes in the simulation area. Only the nodes that are within the communication radius are taken into consideration. Finally we pick up a random node drawn from uniform distribution among the considered nodes. Figure 5.1 illustrates how a destination node is chosen at a particular point of simulation time when communication radius is specified.

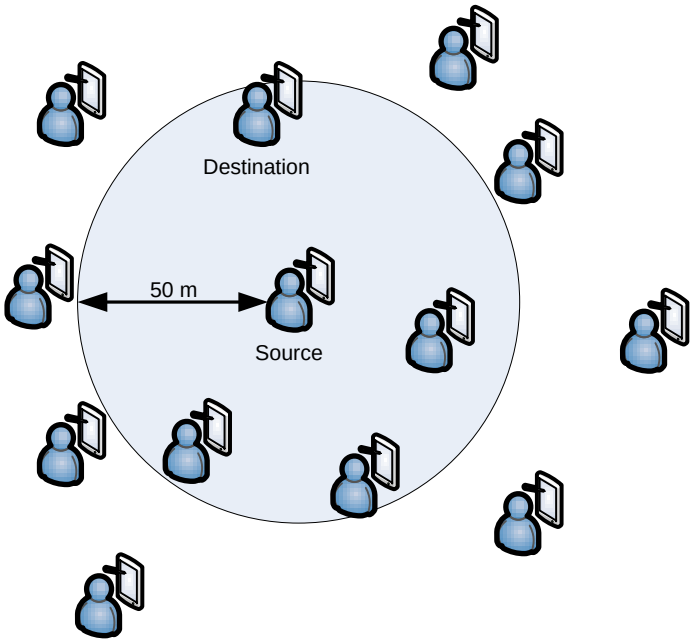

Figure 5.1: Selection of a destination node when communication radius is specified corresponding to a particular source node

5.3.3 Routing Protocols

In the simulation environment, messages are forwarded on the basis of routing protocols. Epidemic routing protocol is primarily used to examine its impact on the delivery probability and the delivery delay. In this routing scheme, a node spreads the same copy of a message to all other nodes in its vicinity, which turns out higher message delivery rate and lower end-to-end transmission delay. But the downside of the routing protocol is that it highly consumes system resources (e.g., bandwidth,

[1]Communication radius should not be confused with transmission range of each mobile node (e.g., 10 m for Bluetooth case)

memory, power). The way which we can minimize the resource consumption is to limit hop-count and Time-To-Live (TTL). The hop-count refers the number of nodes a message can traverse along the path from the source to the destination. The longevity of a message in the network is defined by the message TTL. So we use different hop-count and TTL values as parameters in our simulations and observe how the DT-Talkie reacts with those varying parameters.

We also use binary version of Spray-and-Wait (SnW) routing protocol to conduct the simulations and compare the simulation results with those using epidemic protocol. Basically Spray-and-Wait protocol achieves better delivery rate with low resource utilization by setting a strict upper bound on the number of copies per message allowed in the network. Maximum 10 copies per message are set while using SnW routing protocol in our simulations.

5.3.4 Performance Metrics

To evaluate the performance of the DT-Talkie, we mainly consider two metrics: the *delivery probability* and the *delivery delay*. The delivery probability is defined as the ratio between the number of total messages received and the number of total messages sent in the simulation environment. The delivery delay is calculated as the time that a message takes to transmit from the source to the destination. Both metrics are observed in two different modes: (1) message mode (when messages are sent in full-length) and (2) fragmentation mode (when messages are sent as fragments). In the mobile DTNs when a voice message is sent as fragments, all the fragments of the voice message may or may not reach the destination. But the received voice message might be intelligible if a small number of fragments (e.g., 1 fragment) are missing in the voice message. So we also look at the rate of delivered messages with no loss of fragments and with loss of certain number of fragments.

In our simulation setup, destination nodes can be selected from a certain communication radius in addition to selecting randomly from anywhere in the simulation area. In this thesis we study the effect of different communication radius values (from short to long) on the delivery probability and the delivery delay, when different routing protocols (Epidemic and SnW) and different mobility models (RWP and WDM) are used.

Usually in most of our simulations, messages are sent in one-way direction (from the source to the destination). But in reality we see that voice messages can be exchanged back and forth between two DT-Talkie users. This is exploited in some of our simulations where two users can have different number of interactions. Figure 5.2 depicts a voice session between two users in which three interactions have taken place. Besides delivery probability, we specify two more metrics in this setup: the *session completion rate* and the *session completion time*. The session completion rate is calculated as total number of sessions completed over total number of sessions created and the session completion time is defined as the time elapsed to complete a session. We observe the delivery probability, the session completion rate and the session completion time in both RWP and WDM mobility scenarios and using both Epidemic and SnW routing protocols. For simplicity, the performance metrics are studied only when full-length voice messages are sent in order to understand the behavior of the DT-Talkie while interaction between two users. The metrics analysis in the fragmentation mode would be considered as a possible future work.

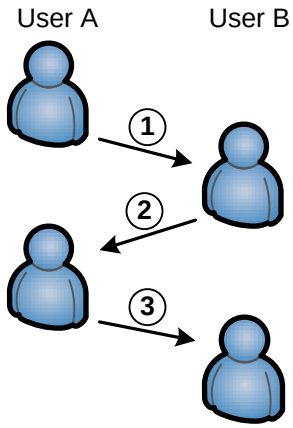

Figure 5.2: A voice session with three interactions

5.3.5 Simulation Parameters

We have carried out simulations in different scenarios using both RWP and WDM mobility models. Different parameters are required to set in order to initialize the simulations in the ONE simulation environment. In the simulations we use warmup period and cooldown period in addition to simulation time. The warmup period is used for the movement models to reach a steady state. During the cooldown period none of the generated messages get recorded, only the received messages are taken into account. In our simulations a default configuration file is used for the default

settings of all the scenarios and another configuration file is used for scenario specific settings. We assume mobile users are moving in the simulation area with DT-Talkie enabled modern mobile devices and the devices have Bluetooth connectivity of 10 m radio range with 2 Mbps data rate. We limit our scope to the Bluetooth case as WLAN radios with 100 m radio range have less impact on the elementary interaction characteristics [57]. The simulation area can be either open space or a part of the Helsinki city area. The mobile devices have upto 100 MB free buffer space for storing and forwarding messages. The buffer size is large enough to never get full. For all scenarios we conduct five simulations using different seed values and plot the average result.

RWP

As a baseline, we use the RWP with 100 mobile nodes modeled as pedestrians and 1000×1000 m^2 simulation areas (open space). For the simulations using this movement model, we use six hours simulation time and two hours for both warmup period and cooldown period. The mobile nodes are moving in the open space at random speeds of 0.5-1.5 m/s with pause times of 0-120 s. Both speeds and pause times are uniformly distributed.

WDM

For the WDM model, we choose the default scenario for section 5 in [59], at which nodes move in the Helsinki city area. But the number of nodes is reduced from 1029 to 323 by shrinking the group sizes that the basic contact characteristics remain. Since WDM models daily routines of working days, we select simulation duration of one working day plus six hours warmup period and two hours cooldown period.

Voice Messaging

For both cases of RWP and WDM, our own implemented message generator requires some parameters to initiate. We choose voice message duration, fragment duration and silence duration in the range of 5-15 s, 2-3 s and 1-2 s respectively. All of the durations are drawn from a Pareto distribution. Pareto coefficient is set to 0.5 for all the cases. There is a parameter that specifies the way messages are generated (either as full-length voice messages or as fragments of voice messages). In every

500 simulation seconds the message generator attempts to create voice messages for sending. For some simulations the destination nodes are selected from a particular distance, which is defined by the communication radius parameter.

This yields a light load on the overall system. Using a light load is intentional because we are interested in the achievable performance given different mobility and messaging models, less in comparing different routing protocols and buffer management schemes.

5.4 Simulation Observations

All the simulations are divided into four groups on the basis of four different settings that are discussed below.

5.4.1 Simulations Group 1

In the first group of simulations, destination nodes are selected randomly from anywhere of the simulation area. In this group we observe the performance metrics with respect to increasing TTL values (minimum 20 minutes and maximum 120 minutes), when the maximum hop-count is set to 10.

In Figure 5.3 and Figure 5.4, we see that the delivery probability in both message and fragmentation modes increases with incrementing TTL values. In the RWP scenario, the performance of the Epidemic routing protocol to deliver messages is higher than the SnW in every TTL value, but to deliver fragments the SnW performs better than the Epidemic when the TTL value is more than 40 minutes. In the WDM scenario, the impact of both routing protocols on the delivery probability is almost equivalent and the delivery probability is very low in both message and fragmentation mode. Only about 5% messages and 3% fragments are delivered with maximum TTL value. This delivery probability is such low that would be impractical to use the DT-Talkie application.

In the fragmentation mode, variable number of fragments is generated depending on the number of talk-spurts in a voice message. Our analysis reveals that in the fragmentation mode some messages are received without any loss of fragments, some are received with loss of single fragment and the rest of them are received with loss of

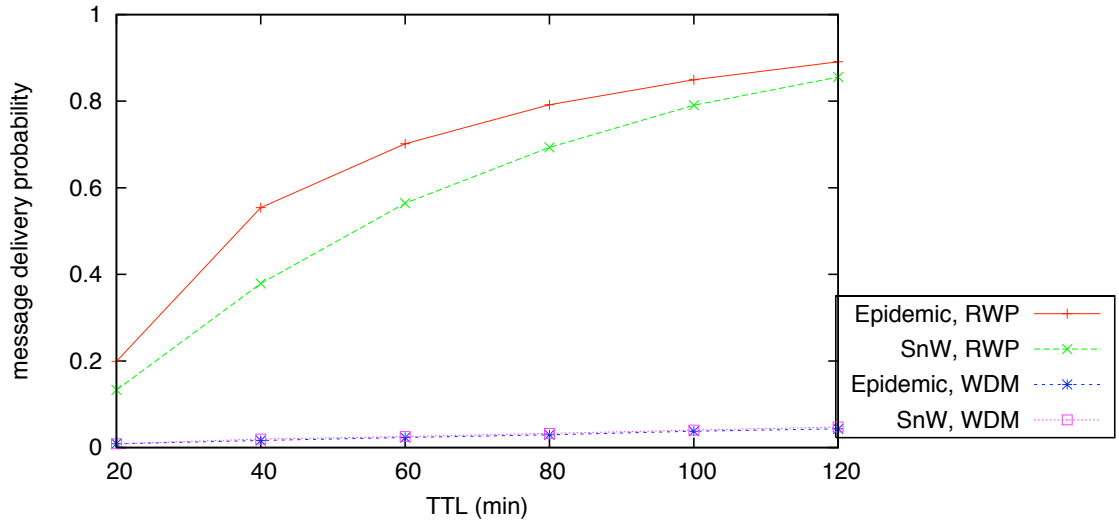

Figure 5.3: Message delivery probability (message mode, maximum 10 hop-count)

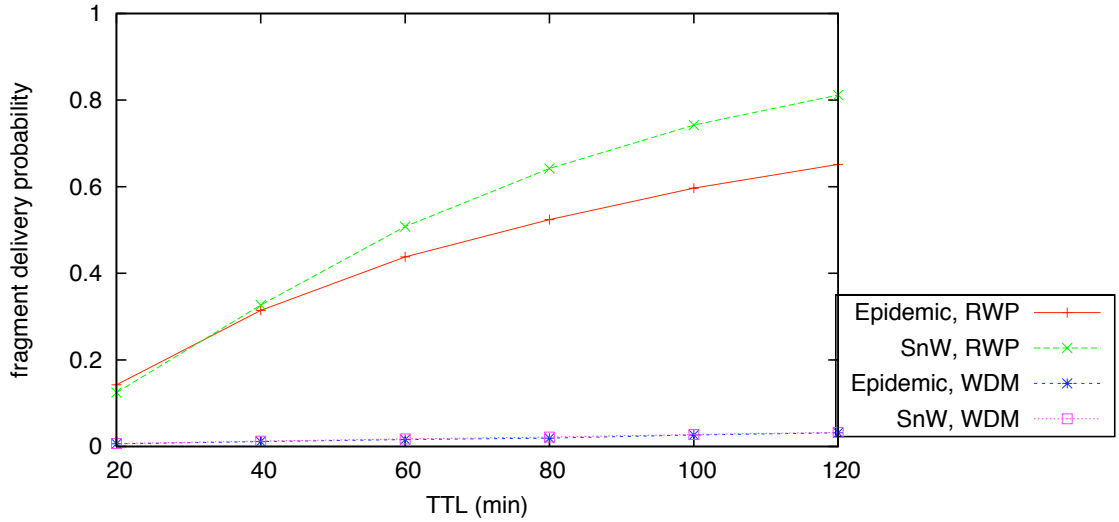

Figure 5.4: Fragment delivery probability (fragmentation mode, maximum 10 hop-count)

two or more fragments. As we discussed earlier that small number of fragment loss may not have so much effect on the quality of a voice message. So the voice messages received with no loss of fragment or with single loss of fragment are assumed as intelligible. We simulate this and plot the results in Figure 5.5. In case of RWP scenario and SnW routing protocol, 53% and 84% messages are delivered with loss of $\leq$1 fragment when the TTL value is 1 hour and 2 hours respectively.

In Figure 5.6 and Figure 5.7, we depict the average delay to transmit both messages and fragments. From the results of the simulations, we find that both messages and

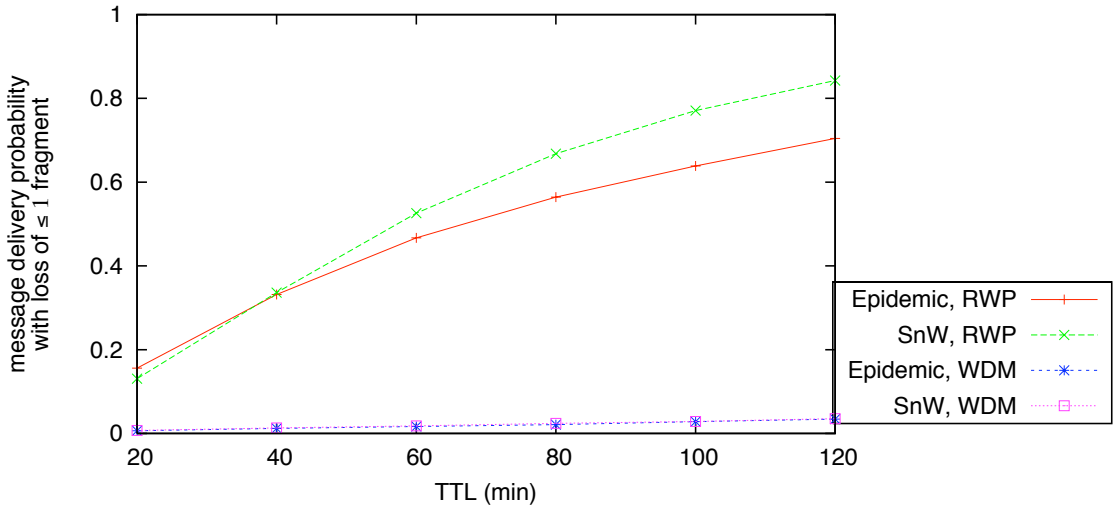

Figure 5.5: Message delivery probability with loss of ≤1 fragment (fragmentation mode, maximum 10 hop-count)

fragments take more time to reach the destination using Epidemic than using SnW in the scenario of both RWP and WDM.

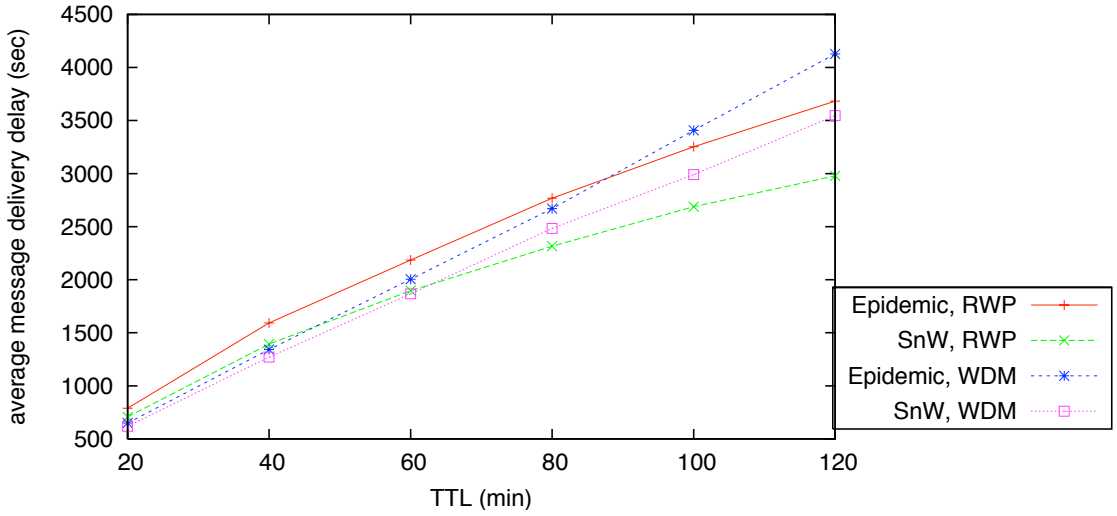

Figure 5.6: Average message delivery delay (message mode, maximum 10 hop-count)

5.4.2 Simulations Group 2

In the second group of simulations, we also select destination node randomly from anywhere of the simulation area as the first group. But the performance metrics are examined in terms of increasing hop-count limit values (minimum 2 and maximum

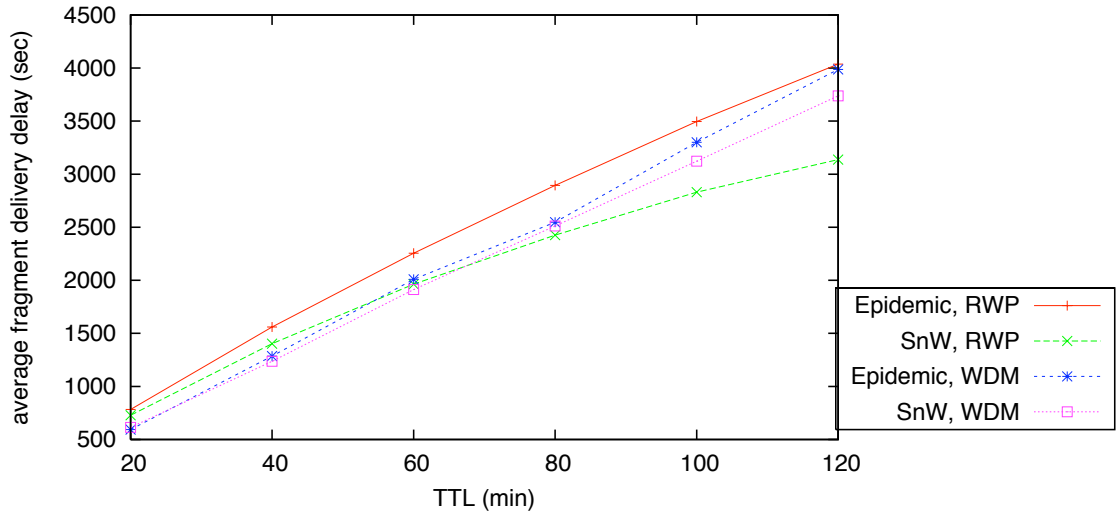

Figure 5.7: Average fragment delivery delay (fragmentation mode, maximum 10 hop-count)

10), when the TTL value is set to maximum 120 minutes.

From Figure 5.8 to Figure 5.10, we see little impact on the delivery probability if the hop-count is 6 or more in both cases of RWP and WDM scenarios and using both epidemic and SnW routing protocols. This means most of the messages or fragments are delivered to the destination through traversing not more than 6 hops.

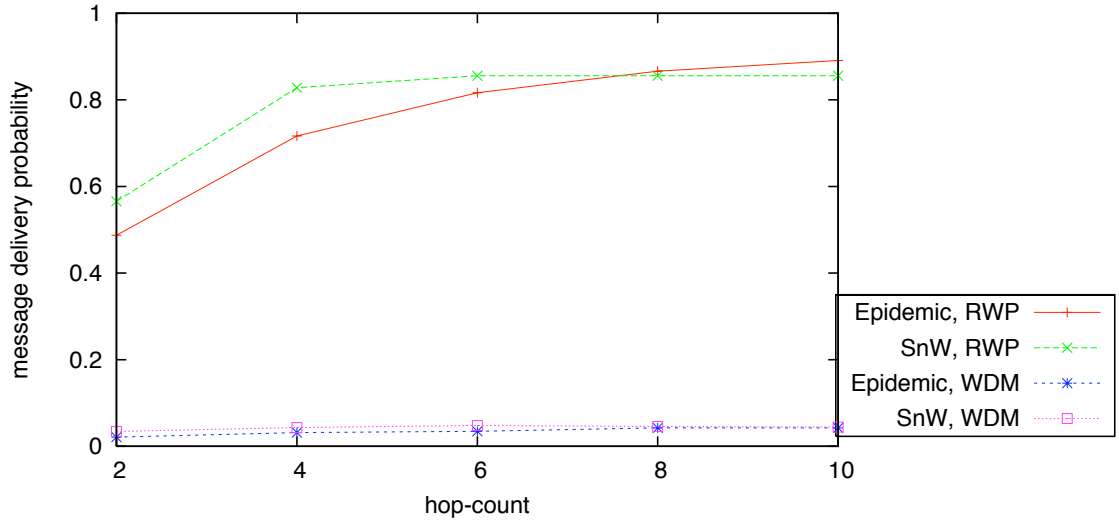

Figure 5.8: Message delivery probability (message mode, maximum 120 minutes TTL)

In Figure 5.11 and Figure 5.12, when messages or fragments are routed using SnW,

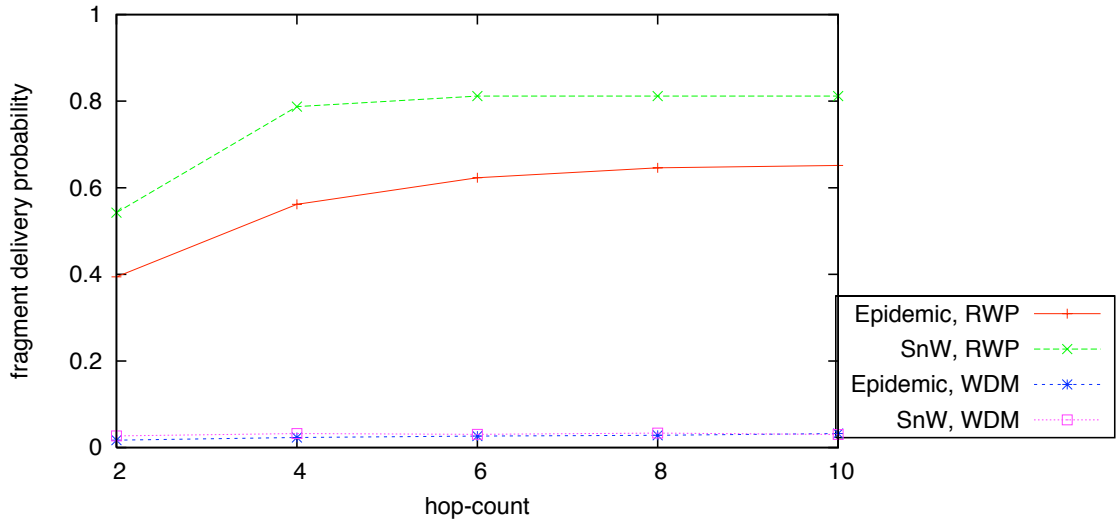

Figure 5.9: Fragment delivery probability (fragmentation mode, maximum 120 minutes TTL)

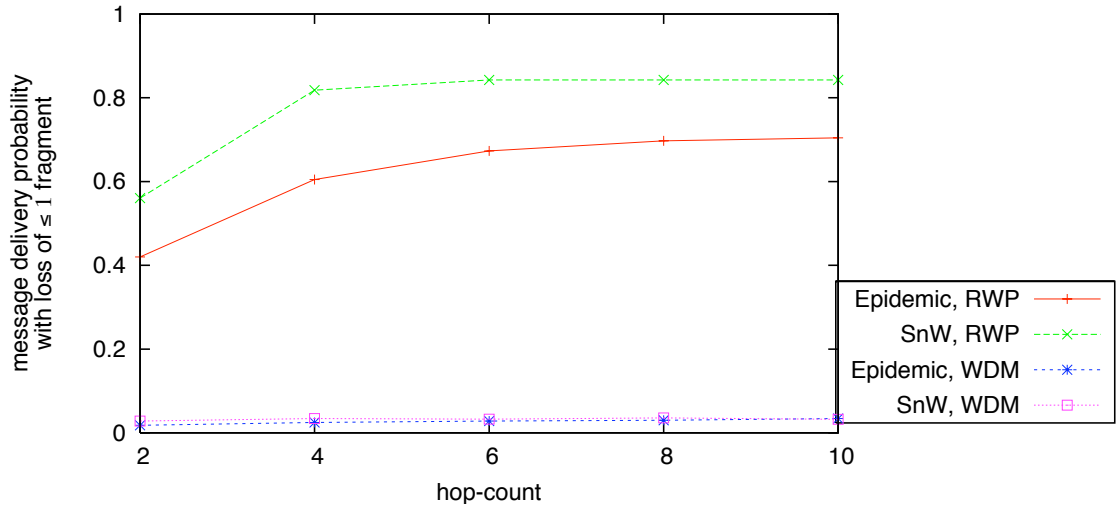

Figure 5.10: Message delivery probability with loss of $\leq$1 fragment (fragmentation mode, maximum 120 minutes TTL)

they take less transmission time on average than using epidemic to reach the destination in both mobility scenarios. Usually increasing hop-count values leads to higher deliver delay. But in some cases the delivery delay can be lower with respect to higher hop-count values. Because when we allow more hops to transmit a message, more copies of the message are dispersed in the network. This increases the probability to meet the destination and allows to deliver the message quickly.

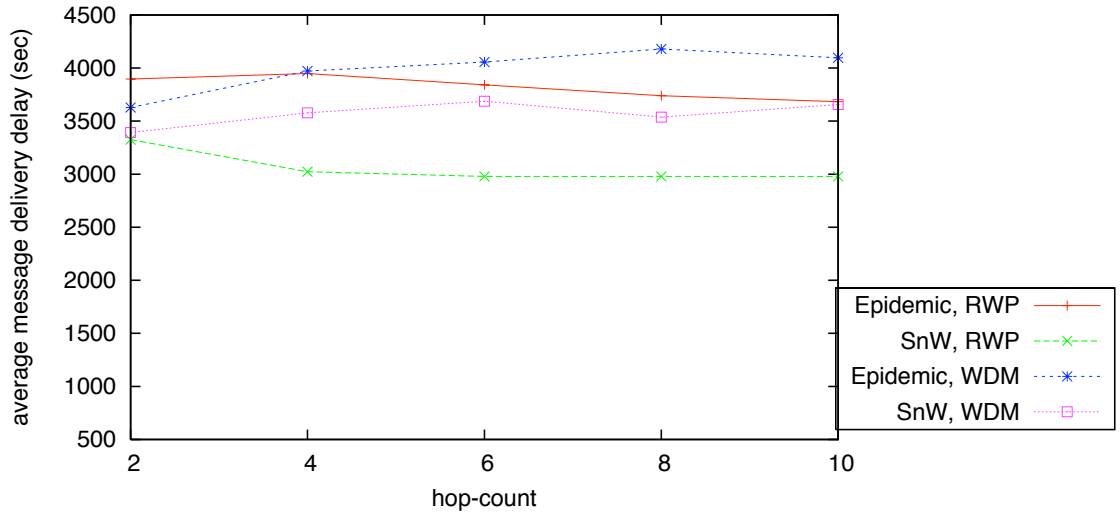

Figure 5.11: Average message delivery delay (message mode, maximum 120 minutes TTL)

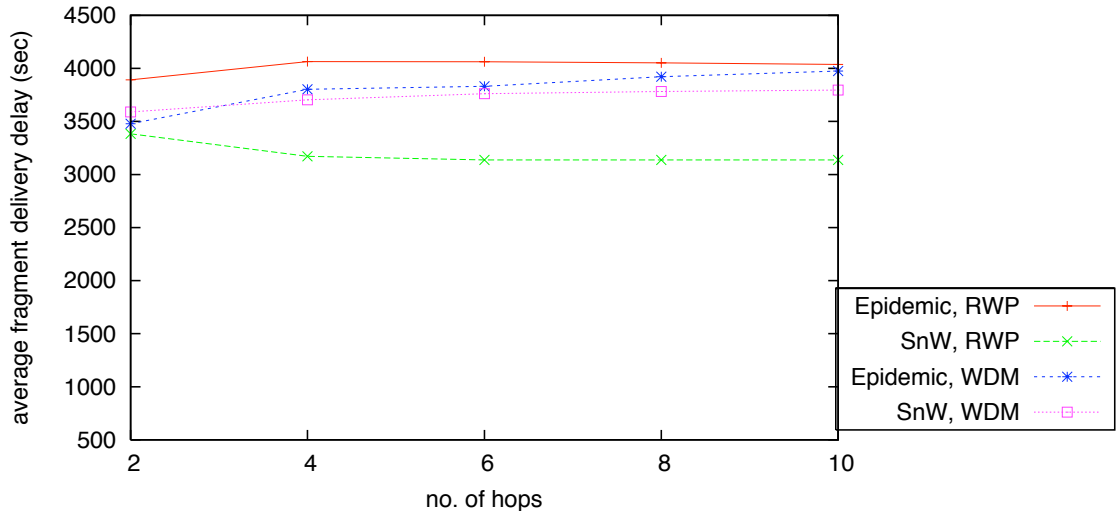

Figure 5.12: Average fragment delivery delay (fragmentation mode, maximum 120 minutes TTL)

5.4.3 Simulations Group 3

Unlike first and second groups, destination nodes are selected randomly from a particular distance of the simulation area, which we call communication radius. The performance metrics are studied over 4 varying values of communication radius: 50, 100, 500 and 1000 meters. We set the TTL and the hop-count values to maximum 120 minutes and to maximum 10 respectively.

From Figure 5.13 to Figure 5.15, the delivery probability, using both Epidemic and SnW routing protocols in the RWP scenario, does not heavily rely on the communication radius parameter. On the other hand, we observe significant effect of the communication radius values on the delivery probability in the WDM scenario. The delivery probability in the WDM scenario is higher when the communication between the source node and the destination node takes place in closer distance. In the message mode when using WDM scenario and using Epidemic routing protocol[1], 48% messages are delivered if the communication radius is set to 50 meters and the delivery rate reduces to 13% in case of 500 meters communication radius. In the fragmentation mode, 47% and 11% messages are delivered with loss of $\leq$1 fragment for 50 and 500 meters communication radius respectively.

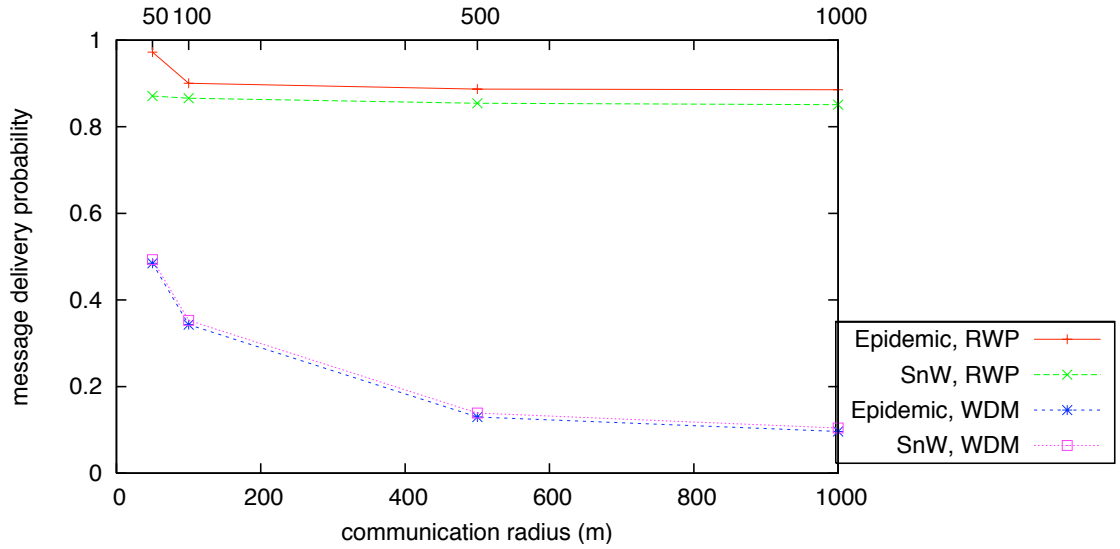

Figure 5.13: Message delivery probability (message mode, maximum 120 minutes TTL, maximum 10 hop-count)

In the WDM scenario of Figure 5.16 and Figure 5.17, shorter communication radius leads to less delivery delay for transmitting either messages or fragments. In this scenario, SnW provides low transmission delay than Epidemic routing protocol in every value of communication radius parameter. With the setting of 50 meters communication radius, the delivery delay is almost half of the delivery delay, which was calculated in the first simulations group with 120 minutes TTL value and 10 hop-count limit.

[1]In the WDM scenario, SnW has almost similar impact as Epidemic on the delivery probability.

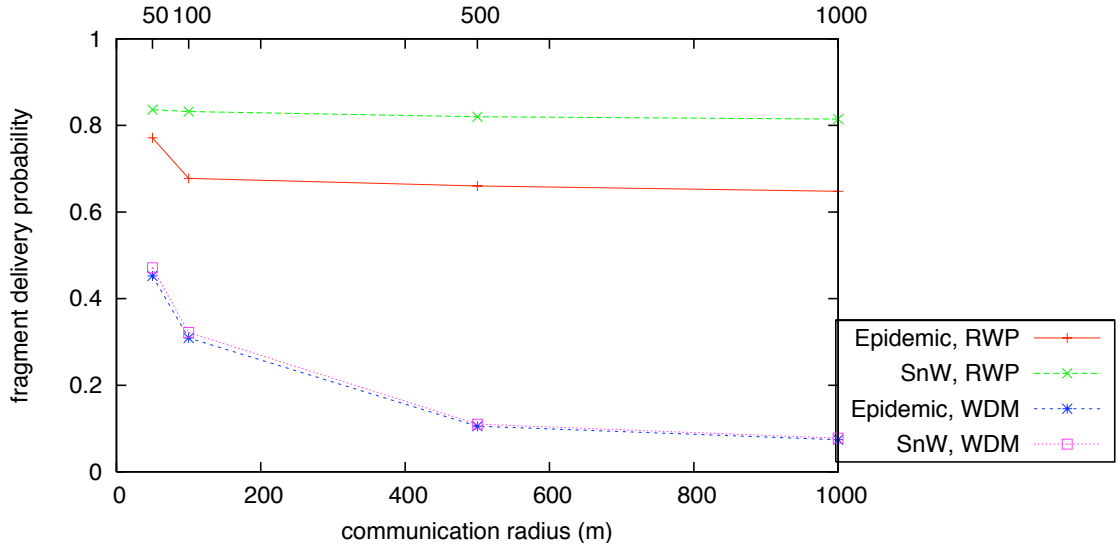

Figure 5.14: Fragment delivery probability (fragmentation mode, maximum 120 minutes TTL, maximum 10 hop-count)

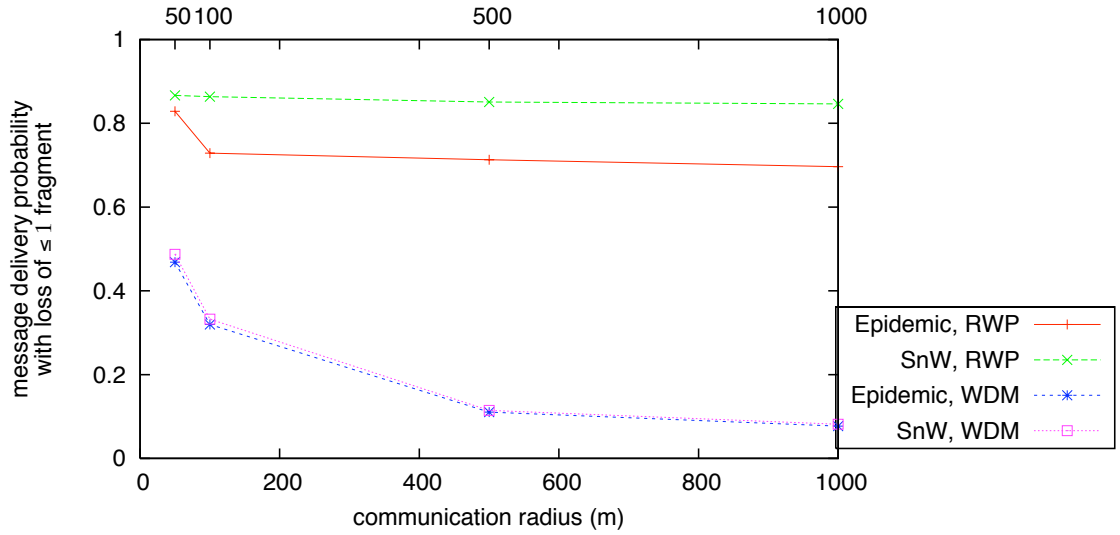

Figure 5.15: Message delivery probability with loss of $\leq$1 fragment (fragmentation mode, maximum 120 minutes TTL, maximum 10 hop-count)

5.4.4 Simulations Group 4

In all of the above groups of simulations, the performance metrics are measured on the basis of one way communication from the source to the destination. In the fourth group of simulations, we study the delivery probability, the session completion rate and the session completion time in terms of different number of interactions (minimum 2 and maximum 6) associated with a voice session. We set the communication

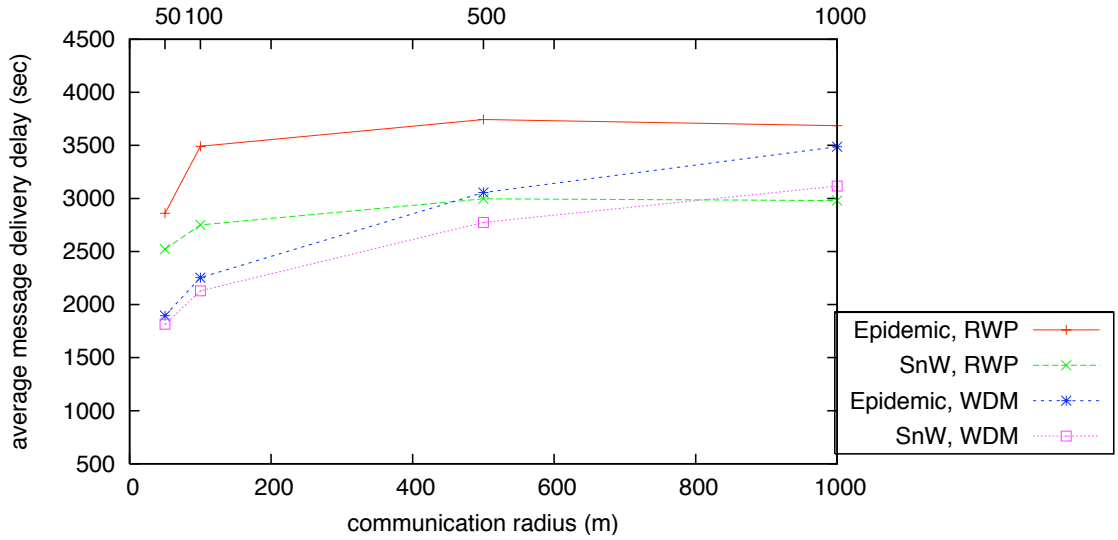

Figure 5.16: Average message delivery delay (message mode, maximum 120 minutes TTL, maximum 10 hop-count)

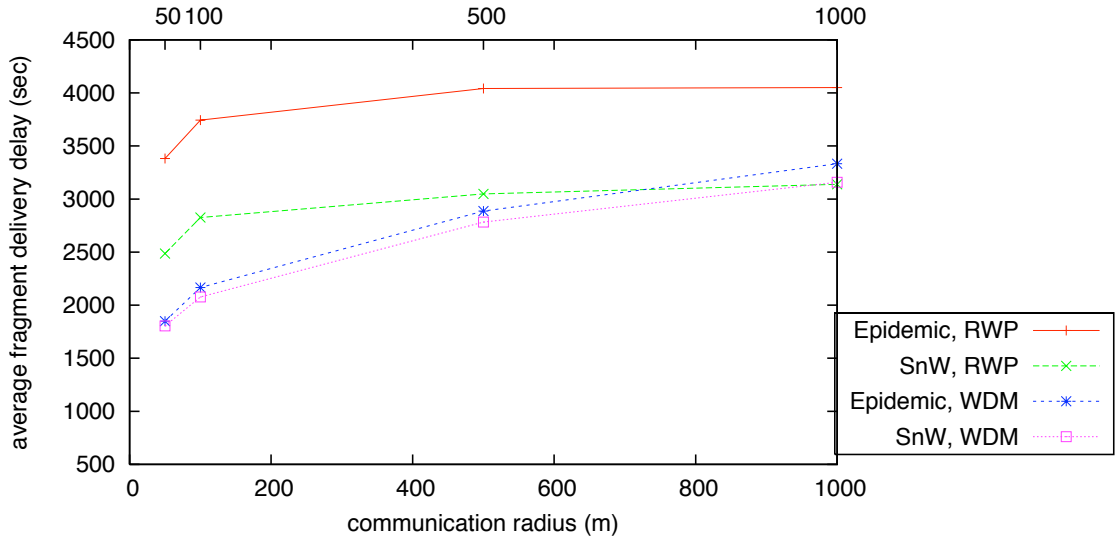

Figure 5.17: Average fragment delivery delay (fragmentation mode, maximum 120 minutes TTL, maximum 10 hop-count)

radius to 50 meters, the TTL to maximum 120 minutes and the hop-count limit to maximum 10. To conduct the simulations for this group, we consider only when voice messages are sent in full-length.

In Figure 5.18 the message delivery probability in the WDM scenario increases with increasing number of interactions. In the RWP scenario when SnW routing protocol

is used, we see that messages reach the destination with high delivery probability than using Epidemic routing protocol.

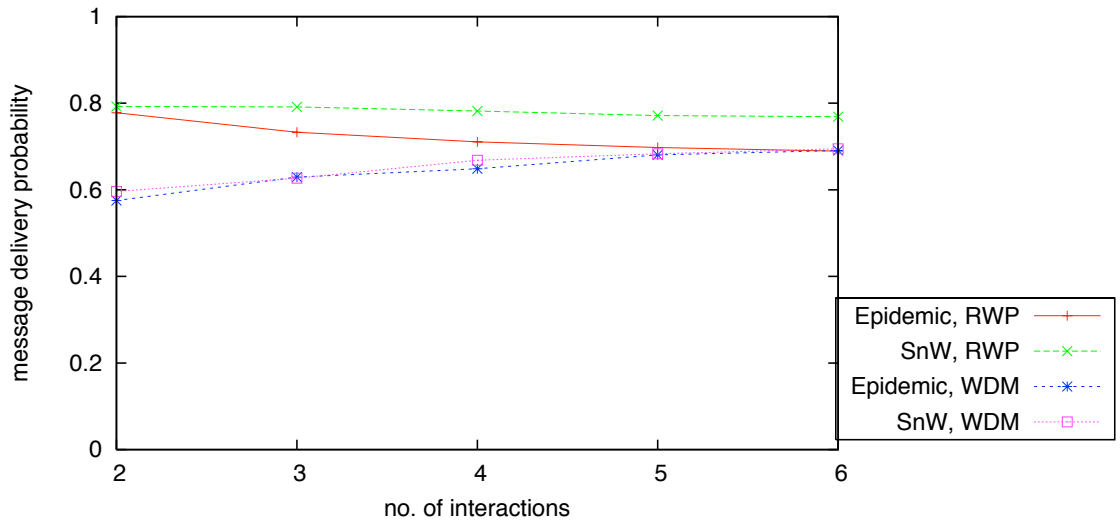

Figure 5.18: Message delivery probability (50 meters communication radius, maximum 120 minutes TTL, maximum 10 hop-count)

In Figure 5.19 the session completion rate is decreasing with increasing number of interactions in both mobility scenarios and using both routing protocols. In this simulation setup, a source node (node A) chooses a destination node (node B) from 50 meters communication radius and messages are exchanged back and forth between two nodes based on the number of interactions. As the nodes move arbitrarily in the RWP scenario, node A and node B may not always stay in the same communication radius. So it can happen that after single exchange of messages between node A and node B, other exchanges may hardly complete when the interactions number is high - thus turns out low session completion rate in case of higher interactions number. On the other hand in the WDM scenario when the nodes interact within 50 meters communication radius in the offices[1], they mostly remain in the same radius and sessions are succeeded most of the time even when interactions number is high. That's why the voice sessions in the WDM scenario are completed in higher rate using Epidemic routing protocol than in the RWP scenario, when 4 or more number of interactions in a voice session is taken place. In the RWP scenario about 24% and 9% sessions are completed with the settings of 4 and 6 interactions number respectively and in the WDM scenario about 26% and 20% sessions are completed are completed with the same settings of interactions number. The performance of

[1]We consider offices as an example because the delivery success is observed in the WDM scenario mostly during working hours of a day.

SnW routing protocol is almost equal to Epidemic routing protocol in the WDM scenario, but SnW provides higher session completion rate compared to Epidemic in the RWP scenario.

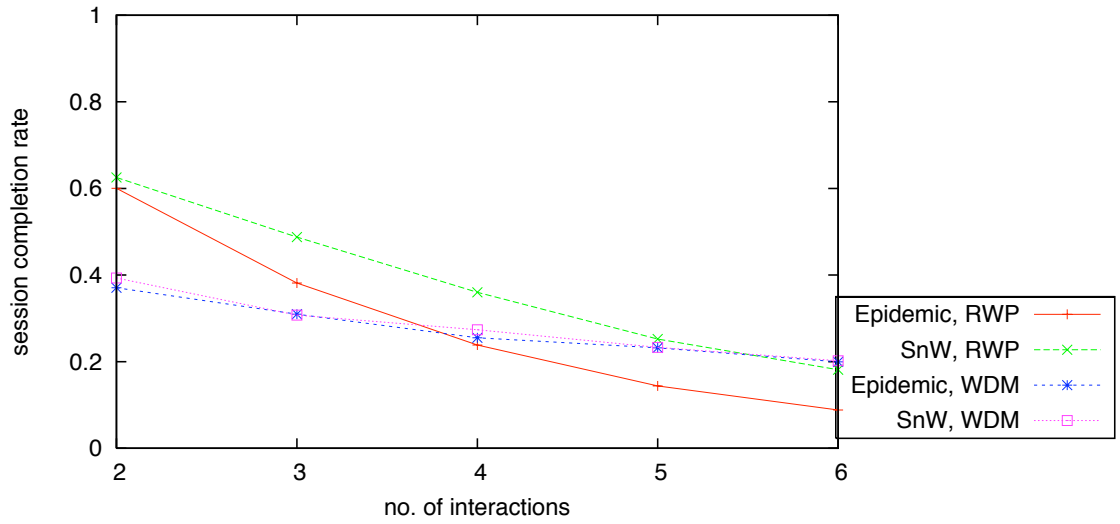

Figure 5.19: Session completion rate (50 meters communication radius, maximum 120 minutes TTL, maximum 10 hop-count)

In Figure 5.19 each voice session takes less time to complete in the WDM scenario than in the RWP scenario for each number of interactions. In the WDM scenario we also observe little increase in the session completion time with respect to increasing number of session interactions.

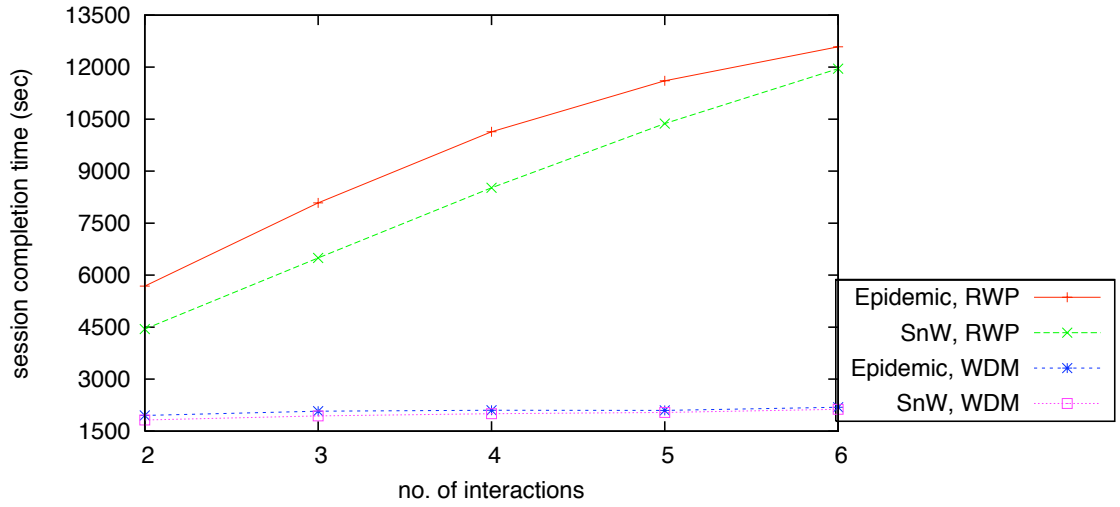

Figure 5.20: Session completion time (50 meters communication radius, maximum 120 minutes TTL, maximum 10 hop-count)

5.5 Summary

In this chapter we evaluate the performance of the DT-Talkie through conducting four groups of simulations using the ONE simulator. In these groups of simulations epidemic and spray-and-wait routing protocols are used in both RWP and WDM mobility scenarios. No routing protocols are seen as perfect in all of the simulation scenarios. But in most of the scenarios SnW routing protocol performs similar like or better than Epidemic routing protocol.

While analyzing the results of the simulations, we observe that Epidemic and SnW routing protocols have nearly same impact on the performance metrics in the WDM scenario. In case of SnW messages or fragments take less transmission time to get delivered to the destination than the case of Epidemic in both RWP and WDM scenarios. We also observe that in the RWP scenario and using Epidemic, messages are delivered with high probability then SnW. On the other hand if the messages are sent as fragments then using SnW the delivery probability becomes higher than using Epidemic.

Selecting destination nodes from closer distance in the WDM scenario yields high delivery probability and low delivery delay than the scenario where destination nodes are selected from anywhere of the simulation area. In some simulations when nodes interact with each other closely in voice sessions, then the sessions in the WDM scenario complete with higher rate for larger number (e.g., 4 or more) of interactions and take less time to complete than in the RWP scenario.

After studying the impact of various simulation settings on the performance metrics, we realize that the DT-Talkie application is more feasible to use practically in the scenario where users stay within a relatively short communication boundary and their movement is not so arbitrary; generally in the offices. In such case any of the routing protocols, Epidemic or SnW, can be used as we observed pretty much similar impact of both routing protocols in the more realistic WDM scenario. The DT-Talkie can also be used in the conference scenario. Typically when people participate a conference, mostly they stay close to each other but move randomly. In that scenario SnW routing protocol is more appropriate to use in the DT-Talkie than Epidemic.

This chapter ends the discussion of the work done in this thesis. In the next chapter, we draw conclusion of the thesis and give some ideas about possible future works.

6 Conclusion

In this thesis, we have presented an approach for voice communication in the mobile DTNs. We have discussed why the traditional IP-oriented voice communication is not suitable in the challenged networking environments and defined necessary requirements to adapt voice communication in those environments. All over the thesis our approach is basically compared with the traditional Internet protocols based Push-to-talk (PTT) system.

Regular PTT services are mainly infrastructure-based and require stable end-to-end path for successful communication. But the mobile users may travel in the environment where infrastructure is not available or frequent disconnection is common. In such cases those PTT services may exhibit poor performance or may even fail completely. Moreover the packet loss characteristics of wireless networks may subject to unintelligible speech, which badly affects the overall user experience. To resolve the above issues we have devised a system called DT-Talkie, which enables individual and group users to communicate in the infrastructure-less and other challenged networking environments in the walkie-talkie fashion. The DTN concept of asynchronous reliable message forwarding is applied to our system. In the DT-Talkie the speech quality is kept intact but the voice session can be less interactive due to delay.

In the DT-Talkie voice messages are encoded after capturing and then carried in the DTN bundles. Rather than direct mapping of voice messages into bundles that can be done trivially, we have incorporated MIME application-layer framing mechanism. This mechanism allows other contents (e.g., image, vCard) to be sent optionally along with voice messages. Unlike traditional PTT services, the DT-Talkie does not assume any codec negotiation approaches (e.g., Offer/Answer Model) before transferring voice messages. The aim is to avoid unnecessary round-trips because they can be costly in the DTN environments. But with this assumption voice messages can be missed to playback in the receiving endpoint due to lack of sufficient codecs support. So in this thesis we have suggested some techniques to accomplish codec negotiation without the cost of some extra round-trip message exchanges. Furthermore we have discussed an application layer fragmentation scheme to transport individual talk-spurts of a large voice message as separate bundles instead of sending a large voice message bundle. This is applied in the stable scenarios with the vision to increase the speed of the session interactivity.

We have implemented the DT-Talkie application that validates the concepts presented in this thesis. The application is primarily developed for the Maemo-based Nokia Internet Tablets. The application architecture relies on DTN Reference Implementation (DTN2) to get the bundle protocol services and other open source technologies to render graphical components, to record and playback voice messages, and to create and parse MIME messages. In order to provide the DT-Talkie support to the heterogeneous clients, we have also ported the application to other Unix/Linux based platforms.

The performance evaluation of the DT-Talkie has been carried out using the ONE simulator in the simple RWP and more realistic WDM scenarios. We have used Epidemic and the Spray-and-Wait routing protocols in both mobility scenarios and observed the performance metrics while voice messages are sent in both message and fragmentation modes. After analyzing the simulation results we have found that in the WDM scenario both epidemic and spray-and-wait routing protocols show almost equal performance and communication in closer distance results high delivery probability and low delivery delay. After simulating session interactions between mobile nodes, we have figured out that in the WDM scenario more sessions are completed with less session completion time than in the RWP scenario in case of higher number of interactions.

In reality the DT-Talkie application can be used in the offices where people remain at shorter distance and they move in nearly nonrandom fashion. In this case either Epidemic or SnW would be perfect as a routing protocol for the DT-Talkie. Moreover people in the conference may use the DT-Talkie for communication. As the movement of the people in the conference is arbitrary, so in such scenario SnW routing protocol is more suitable to use in the DT-Talkie than Epidemic.

An obvious step for future development of the DT-Talkie is to integrate codec negotiation feature and fragmentation support with the capability of link adaptation. In the current implementation, two DT-Talkie users communicate on the basis of a preconfigured codec, which seems insensible because in reality one user may not have the idea about which codecs are supported by other user. The suggested codec negotiation mechanisms in this thesis can serve as a valuable basis to resolve the

codec interoperability issues. On the other hand we have implemented the DT-Talkie fragmentation mode in this thesis as a separate functionality with the assumption of better link connectivity. But in the practical scenarios the link performance can be varied while ongoing DT-Talkie session. So it could be potentially worthy if the application has the ability to observe the link characteristics (e.g., by examining the round-trip time) and make some decisions about on what link conditions the DT-Talkie will switch to fragmentation mode. However having the codec negotiation feature and the fragmentation support with link adaptivity would make the DT-Talkie more useful piece of software.

While performing simulations, the load in the network was intentionally low to have the achievable performance. But it would be worthwhile to see how the DT-Talkie behaves under heavy load condition. Two routing protocols, Epidemic and Spray-and-Wait, have been used in the current simulations to assess the performance of the DT-Talkie. Conducting simulations using other routing protocols, such as PRoPHET, could give a better understanding of the DT-Talkie system.

Enabling security support in the DT-Talkie application could be another interesting avenue to pursue. The application may be jeopardized by malevolent security attacks (e.g., DoS attack) due to the resource-scarcity that characterizes many DTNs. Epidemic routing protocol, currently used in the DT-Talkie, floods voice messages in the network and the messages may travel an arbitrary path of hosts before reaching its ultimate destination. In such case malicious users in the middle can eavesdrop the voice messages or alter the voice bundles if the security services (e.g., authenticity and confidentiality) are not integrated in the application. Those security services can be applied in the bundle layer using the bundle security protocol, even though it is still a work-in-progress and there are some significant open issues (e.g., key management) remaining to be determined. On the other hand since the DT-Talkie application uses MIME encapsulation, the security features could be easily employed in the application layer using S/MIME with the ideas from the previous work done in [60].

As the expectation of the Internet-based communication stretches out to even more diverse network environments, the Delay-Tolerant Networking can play a significant role among future networking technologies. The research in the DTN field is pro-

gressing rapidly. The success of the DT-Talkie heavily relies on the maturity of the DTN technology. However after resolving the remaining issues of the DT-Talkie, it might be a prospective voice communication application for the mobile DTNs.

References

[1] Open Mobile Alliance. Push to talk over Cellular (PoC) Architecture. OMA-AD-PoC-V1 0-20060609-A, June 2006.

[2] Lin-Yi Wu, Meng-Hsun Tsai, Yi-Bing Lin, and Jen-Shun Yang. A Client-Side Design and Implementation for Push to Talk over Cellular Service. Wireless Communications and Mobile Computing, June 2007.

[3] A. Parthasarathy. Push to talk over Cellular (PoC) Server. IEEE conference on Networking, Sensing and Control, March 2005.

[4] Kim P., Balazs A., Broek E., Kieselmann G., and Bohm W. IMS-based Push-to-Talk over GPRS/UMTS. IEEE Wireless Communications and Networking Conference, March 2005.

[5] S.K. Raktale. 3PoC - An Architecture for Enabling Push to Talk Services in 3GPP Networks. IEEE International Conference on Personal Wireless Communications, January 2005.

[6] Rui Santos Cruz, Mário Serafim Nunes, Guido Varatojo, and Luís Reis. Push-To-Talk in IMS Mobile Environment. International Conference on Networking and Services, April 2009.

[7] Valter Rönnholm. Push-to-Talk over Bluetooth. Hawaii International Conference on System Sciences, January 2006.

[8] Jiun-Ren Lin, Ai-Chun Pang, and Yung-Chi Wang. iPTT: Peer-to-Peer Push-to-Talk for VoIP. ACM International Wireless Communications and Mobile Computing Conference, December 2008.

[9] Chai-Hien Gan and Yi-Bing Lin. Push-to-Talk Service for Intelligent Transportation Systems. IEEE Transactions on Intelligent Transportation Systems, September 2007.

[10] L.-H. Chang, C.-H. Sung, H.-C. Chu, and J.-J. Liaw. Design and implementation of the push-to-talk service in ad hoc VoIP network. IET Communications, May 2009.

[11] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H.Weiss. Delay-Tolerant Network Architecture. RFC4838, 2007.

[12] Stephen Farrell and Vinny Cahill. Delay- and Disruption-Tolerant Networking, 2006.

[13] Delay-Tolerant Networking Research Group. *http://www.dtnrg.org/*.

[14] Kevin Fall and Stephen Farrell. DTN: An Architectural Retrospective. IEEE Journal on Selected Areas in Communications, June 2008.

[15] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC3986, January 2005.

[16] Keith L. Scott and Scott C. Burleigh. Bundle Protocol Specification. RFC5050, November 2007.

[17] K. Fall, S. Burleigh, A. Doria, J. Ott, and D. Young. The DTN URI Scheme. Internet-Draft, March 2009.

[18] Forrest Warthman. Delay-Tolerant Netwokrs (DTNs): A Tutorial v1.1, March 2003.

[19] Kevin Fall, Wei Hong, and Samuel Madden. Custody Transfer for Reliable Delivery in Delay Tolerant Networks. Intel Research Technical Report IRB-TR-03-030, July 2003.

[20] S. Farrell, S. Symington, H. Weiss, and P. Lovell. Delay-Tolerant Networking Security Overview. Internet-Draft, March 2009.

[21] Kevin Fall. A Delay-Tolerant Network Architecture for Challenged Internets. Proceedings of ACM SIGCOMM 2003, Computer Communications Review, Vol 33, No 4, August 2003.

[22] S. Symington, S. Farrell, H. Weiss, and P. Lovell. Bundle Security Protocol Specification. Internet-Draft, November 2009.

[23] W. Eddy. Using Self-Delimiting Numeric Values in Protocols. Internet-Draft, November 2009.

[24] Abstract Syntax Notation One (ASN.1). ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002, 2003.

[25] Michael Demmer and Jörg Ott. Delay Tolerant Networking TCP Convergence Layer Protocol. Internet Draft draft-demmer-dtnrg-tcp-clayer-00.txt, Work in Progress, October 2006.

[26] H. Kruse and S. Ostermann. UDP Convergence Layers for the DTN Bundle and LTP Protocols. Internet-Draft, November 2008.

[27] Lloyd Wood, Wesley M. Eddy, Will Ivancic, Jim McKim, and Chris Jackson. Saratoga: a Delay-Tolerant Networking convergence layer with efficient link utilization. International Workshop on Satellite and Space Communications, September 2007.

[28] Thomas Clausen and Philippe Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 2326, October 2003.

[29] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. Experimental RFC 3561, July 2003.

[30] Z. Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. IEEE Communications Surveys and Tutorials, January 2006.

[31] Radu Handorean, Christopher Gill, and Gruia-Catalin. Accommodating Transient Connectivity in Ad Hoc and Mobile Settings. Second International Conference on Pervasive Computing, April 2004.

[32] Shashidhar Merugu, Mostafa Ammar, and Ellen Zegura. Routing in Space and Time in Networks with Predicable Mobility. Georgia Institute of Technology Technical Report, 2004.

[33] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in Delay Tolerant Networks. Proceedings of the ACM SIGCOMM 2004 Conference, Portland, OR, USA, 2004.

[34] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical Report CS-200006, Duke University, April 2000.

[35] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259, 2005.

[36] Anders Lindgren, Avri Doria, and Olov Schelen. Probabilistic routing in intermittently connected networks. In *The First International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR)*, 2004.

[37] C. Becker and G. Schiele. New Mechanisms for Routing in Ad Hoc Networks. 4th Plenary Cabernet Workshop, October 2001.

[38] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks. In *Proceedings of the ACM Mobihoc, Tokyo, Japan*, May 2004.

[39] Yong Wang, Sushant Jain, Margaret Martonosi, and Kevin Fall. Erasure Coding Based Routing for Opportunistic Networks. In *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[40] Jörg Widmer and Jean-Yves Le Boudec. Network Coding for Efficient Communication in Extreme Networks. In *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[41] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[42] Henning Schulzrine, Stephen Casner, Ron Frederick, and Van Jacobsen. RTP: A Transport Protocol for Real-Time Applications, July 2003. RFC 3550.

[43] vCard (Version 2.1). *http://www.imc.org/pdi/vcard-21.txt*.

[44] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. RFC 1521, September 1993.

[45] Jörg Ott, Teemu Kärkkäinen, and Mikko Pitkänen. Application Conventions for Bundle-based Communications. Internet-Draft, November 2007.

[46] XML 1.0 (Fifth Edition). *http://www.w3.org/TR/2008/REC-xml-20081126/*.

[47] maemo.org. *http://maemo.org/*.

[48] Scratchbox. *http://www.scratchbox.org/*.

[49] GTK+. *http://www.gtk.org/*.

[50] GStreamer. *http://gstreamer.freedesktop.org/*.

[51] DTN Reference Implementation. *http://www.dtnrg.org/wiki/Code*.

[52] IBR-DTN. *http://www.ibr.cs.tu-bs.de/projects/ibr-dtn*.

[53] Off-World Communication Protocols Research Project Bundling Protocol. *http://irg.cs.ohiou.edu/ocp/bundling.html*.

[54] DTN@TKK Comnet. *http://www.netlab.tkk.fi/ jo/dtn/index.html*.

[55] T. D. Dang, B. Sonkoly, and S. Molnàr. Fractal Analysis and Modelling of VoIP Traffic. 11th International Telecommunications Network Strategy and Planning Symposium, June 2004.

[56] Md. Tarikul Islam. DT-Talkie: Interactive Voice Messaging for Heterogeneous Groups in Delay-Tolerant Networks. IEEE Consumer Communications and Net-working Conference, January 2009.

[57] A. Keränen and J. Ott. Increasing Reality for DTN Protocol Simulations. Technical report, Helsinki University of Technology, Networking Laboratory, 2007.

[58] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. *Mobile Computing*, 353:153–181, 1996.

[59] Frans Ekman, Ari Keranen, Jouni Karvo, and Jörg Ott. Working day movement model. In *Proceedings of the 1st SIGMOBILE Workshop on Mobility Models for Networking Research*, May 2008.

[60] Md. Tarikul Islam. Secure Email Gateway Access for DTN. Special Assignment, 2008.