A Distributed Approach to Context-Aware Networks

Prototype System Design, Implementation, and Evaluation

MARKUS SWENSON

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2007

COS/CCS 2007-03

A Distributed Approach to Context-Aware Networks

Prototype System Design, Implementation, and Evaluation

Markus Swenson

2007-02-11
Master of Science Thesis

School of Information and Communication Technology (ICT)
Royal Institute of Technology (KTH)
Stockholm, Sweden

Project conducted at Ericsson Research, Ericsson AB

Supervisor: Dr. Theo Kanter (Ericsson Research)
Examiner: Prof. Gerald Q. Maguire Jr.

Abstract

Utilizing context information and in networks, enabling network services to act upon context information, and exchanging context information with applications, constitutes an important new approach to designing communication systems and central to the research project named Ambient Networks. The Ambient Networks project is a part of the 6th Framework Project co-funded by the European Commission and carried out by industry and academia.

A system is said to be context-aware when it reacts to changes in context i.e., information which describes an entity's current situation. This new approach enables developments of systems that are more adaptive to user needs and behavior. As a result systems can provide a homogenous appearance which is important as more and more different network access technologies arise.

This thesis investigates, models, implements, and evaluates a distributed context-aware architecture for Ambient Networks, the Distributed Context eXchange Protocol (DCXP). The solution is a proof-of-concept that shows how a context-aware ambient network can benefit from a distributed approach. The current design is based on a peer-to-peer architecture that forms an overlay to distribute context information among the participating units. This distributed approach was chosen in order to balance the load and also enable a device to easily locate and fetch desired context information.

The evaluation of the proposed context-aware architecture addresses the issues of how such a system ties in with the ideas of Ambient Networks. The main result of this report is a prototype enabling nodes in an ambient network to exchange context information. Moreover, the results show that the prototype needs to be refined in order to work in larger scale networks.

Sammanfattning

Användning av miljö-beskrivande information, så kallad context information, i olika nätverk är en ny infallsvinkel i designen av kommunikationssystem och är av stor vikt i forskningsprojektet Ambient Networks. Målet är att context information ska kunna utnyttjas i nätverken av olika tjänster samt även dela informationen med applikationer. Ambient Networks projektet är en del av det sjätte EU finansierade ramprogrammet där industrin och den akademiska världen deltar.

Ett nätverk eller system klassificeras som context medvetet, context-aware, när det tar hänsyn till förändringar i sk. context information. Context information eller miljö-beskrivande information beskriver en enhets nuvarande situation. Detta möjliggör utveckling av system som ”lyssnar” på användaren och anpassar sig efter dess behov och beteende. Ett praktiskt exempel skulle kunna vara att användare upplever det som ett homogent system trots att det finns flera underliggande access teknologier.

Den här uppsatsen undersöker, designar, implementerar och utvärderar en distribuerad context-aware arkitektur för Ambient Networks, Distributed Context eXchange Protocol (DCXP). Lösningen visar hur ett ambient network kan nyttja en distribuerad lösning för att hantera context information. Designen bygger på att de deltagande noderna skapar ett virtuellt nät, overlay, för att mellan sig dela på context informationen. Den här lösningen valdes för att balansera belastningen jämt mellan de deltagande noderna samt att på ett enkelt sätt för varje enskild node kunna lokalisera och hämta önskad context information.

Utvärdering av den föreslagna lösningen visar på hur den kan integreras med den övriga utvecklingen som skett inom Ambient Networks projektet. Det huvudsakliga resultatet av arbetet är en prototyp som möjliggör för noder i ett ambient nätverk att utbyta context information. Vidare visar även resultatet att prototypen bör vidareutvecklas för att fungera i större sammanhang.

Acknowledgements

First of all, I would like to thank Prof. Gerald Q. Maguire Jr., my thesis's examiner for valuable feedback, support and suggestions of how to improve the work whenever I had a question to ask.

I would also like to express my gratitude to my advisor at Ericsson, Dr. Theo Kanter, for his support and feedback during this period. Especially for giving me the opportunity to work with such an interesting area at such an interesting company as Ericsson.

Table of contents

List of figures

List of Abbreviations

ACS	Ambient Control Space
AN	Ambient Networks
ANI	Ambient Networks Interface
CC	Context Coordinator
CDC	Connected Device Configuration
CIB	Context Information Base
CIO	Context Information Object
CM	Context Manager
CUA	Context User Agent
DHT	Distributed Hash Table
DCXP	Distributed Context Exchange Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
JVM	Java Virtual Machine
MD	Message Digest
P2P	Peer-to-peer
SHA	Secure Hash Algorithm
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UCI	Universal Context ID
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
XML	Extensible Markup Language

1 Introduction

This chapter outlines the problem area. It also states the limitations and goals of this thesis. It provides an introduction to the subject, together with the expected achievements of this thesis.

Networks today are very complex systems which require a lot of technical knowledge in order to run, maintain, and (re)configure. This is expensive for operators and more and more operators are outsourcing their network management and maintenance to specialists such as Ericsson, who by the end of 2005 managed networks with more than 50 million subscribers [1]. In order to make networks beyond 3G (i.e. beyond UMTS and CDMA-2000) more adaptive to their surrounding environment and give them a greater degree of self-organization and self-management, explicit use of network context information is being introduced to these systems. Context information is information that describes the situation of an entity, where this entity can be a person, a physical object, a network object, etc. A network that reacts to changes in context is said to be context-aware. Apart from improved management this context information also provides means for networks to optimize their use of network resources.

> *"The Ambient Network project [2] aims at an innovative, industrially exploitable new network vision based on the dynamic composition of networks to avoid adding to the growing patchwork of extensions to existing architectures." [3]*

Network context information also enhances the services offered to users as this context information can be provided to higher layer services and applications. This is one of the reasons why the Ambient Network project is taking place and one of its work packages (WP-D) is devoted to network context information.

The Ambient Network project has introduced a name for their system architecture that handles this context information: ContextWare. It consists of middleware that acquires, disseminates, and mediates context information between context sources and context consumers, taking into account the dynamic properties of Ambient Networks.

The use of network-related context information enables novel ways of using communication devices, -services, and –systems. With Ambient Networks the user is no longer limited to using one device at a time with limited possibilities for swapping networks or inter-working between devices. In order to provide a seamless user experience in a heterogeneous network world, access to network-related context information is crucial. The lack of network-related context information and a means to collect and distribute this information will be met by the software to be provided by the Ambient Network project's WP-D. The Ambient Network project proposes a solution using an approach that enables different networks to cooperate transparently without earlier offline negotiated contracts between network owners/operators. The project tries to provide users with a feeling of a homogenous network, but without adding more complexity for users or administrators; thus an autonomic approach must be used. This

forces even the network to be context-aware. By using the available context the network can make decisions, such as which route to choose and how to provide the user with better service. When users use wireless networks the environment may change frequently and by using available context information both the network and user applications are able to adapt to these changes – hence becoming context-aware. The Ambient Network project proposes making decisions in the network layer regarding allocation of resources to services and network composition based upon context information. This way, diverse networks can be made to look homogeneous to the user and the user's applications.

1.1 Problem statement

The Ambient Networks project proposes that a distributed system be used due to its characteristics, i.e., distributed storage capabilities, its ability to deal with churn, storage redundancy, and scalability. Previous work affiliated with the Ambient Network project investigated and developed a central client-server solution to distribute context information within and to other ambient networks [4]. This solution were inspired by techniques from SIP's Event Notification Framework [5] of how to subscribe and notify entities about changes in the subscribed information. In order to minimize the work of the core network handling all of the relevant context information, research is now examing a distributed solution. This specific thesis project investigates how to make use of peer-to-peer (P2P) techniques as a means of implementing a distributed solution.

The task of this thesis is to investigate, model, structure, and evaluate a distributed approach (inspired by P2P protocols) for context information aggregation and dissemination in the Ambient Network.

This thesis is structured in terms of the following tasks:

- Model and design a P2P inspired version of ContextWare for ambient networks. This approach should include the functional entities identified by WP-D and Ambient Interfaces, to facilitate integration with other activities in the Ambient Networks project.
- Develop a protocol for handling context information in a distributed fashion. This protocol is used by the entities in an ambient network in order to manage, collect and disseminate context information. The protocol must be able to dynamically compose and decompose different ambient networks.
- Develop a prototype of the proposed ContextWare architecture. The prototype should be able to run on handheld devices such as PDAs running a Java (specifically Java ME) application. Generate the relevant context information needed for the prototype.
- Evaluate the prototype using the designed protocol to determine that the distributed ContextWare is a feasible solution for different kinds of ambient networks and provides a more optimized solution than a central server can provide. Measurements of specific interest are latency, latency when two networks compose, and management overhead caused by the distributed architecture. This should provide answers if the protocol are redundant, scalable and able to deal with high churn.
- Propose further work based on the tests and evaluation of the prototype.

The result of this thesis is a distributed architecture and protocol that allows ambient networks to exchange and manage context information within a single ambient network and with other ambient networks. The proposed solution is evaluated using measurements made when running the prototype.

1.2 Purpose

The purpose with this thesis is to enhance the context-aware support in the Ambient Control Space based upon aggregation and disseminating network context information in a distributed fashion. A prototype implementing a ContextWare architecture using the newly designed protocol are one of the key deliverables.

1.3 Delimitation

This thesis is not looking into how the distributed context information is used or how the policies for distributing the information are managed. The thesis is not considering security issues.

2 Background

This chapter introduces the Ambient Networks project together with other related background information. It explains the concept of an ambient network and defines what context information is, and how such information could be stored, retrieved and used.

2.1 Ambient Networks

The Ambient Networks project is co-funded by the European Commission as a part of the European Community's Sixth Framework Program and its main objective is "Mobile and Wireless Systems Beyond 3G". Both industry and academia are also committed to the project. Among the larger companies, Ericsson, British Telecom, Alcatel, and Vodafone are represented. The goals of the project are to define a network architecture that embraces a wide range of existing and new techniques. This enables a dynamic composition of networks which will provide a homogenous feeling over a heterogeneous range of access technologies. Another goal is based upon the hypothesis that if networks are adaptive and self-configuring, then the effort required building, configuring, and maintaining them is reduced. The project is divided into work-packages (WP) where each work-package has its own area of interest. One of the original packages is of specific relevance to this thesis: WP6 "context aware networks". Phase 2 of the Ambient Networks project started 2006-01-01. In phase 2 context aware networks were moved to work package D as Task 1.

2.1.1 Ambient Networks Architecture

The networks within an Ambient Network range from small personal area networks to large scale networks such as wide area cellular networks and satellite networks. The Ambient Networks project design focus is on offering common control functions to a large number of different applications and air interfaces in order for users to choose from a greater range of network operators and network technologies. This makes it possible to enhance users' ability to more efficient use provided services and enable service providers to deliver easy to use services. In order to do so a common control layer is defined, the Ambient Control Space (ACS) [6]. The Ambient Control Space consists of Ambient Interfaces together with a connectivity network is called an ambient network. A view of how an ACS in an ambient network connects services to a wide range of heterogeneous network technologies is shown in Figure 1.

Below some key ideas of the Ambient Networks project are described [6]:

- Heterogeneous networks

  Instead of limiting the available technologies used for accessing networks the Ambient Networks project introduced a common control layer that can handle all kinds of access technologies based on a user's requirements and preferences. This makes it easier for users as the access methods are hidden from the applications.

- Network composition

  The network architecture must support functions such that networks can compose on-the-fly without earlier agreements

needed. Therefore real-time negotiation is needed between two networks when they compose in order to agree on services and policies. This feature enables users to easily access new networks and services as they move around.

– Context Information

Although context information is not something new introduced by this project, how it is to be used is new. The importance of collecting, distributing, and managing context information within and across domains is crucial in order to improve service delivery, adapt to changes in the environment, and give the users a better experience. This involves not only providing the network with access to this information, but also providing this information to higher-layer applications.

– All-IP networks

In the heterogeneous network world that the Ambient Networks project assumes all networks are IP-based, hence they provide a network layer that guarantees basic connectivity between different networks.

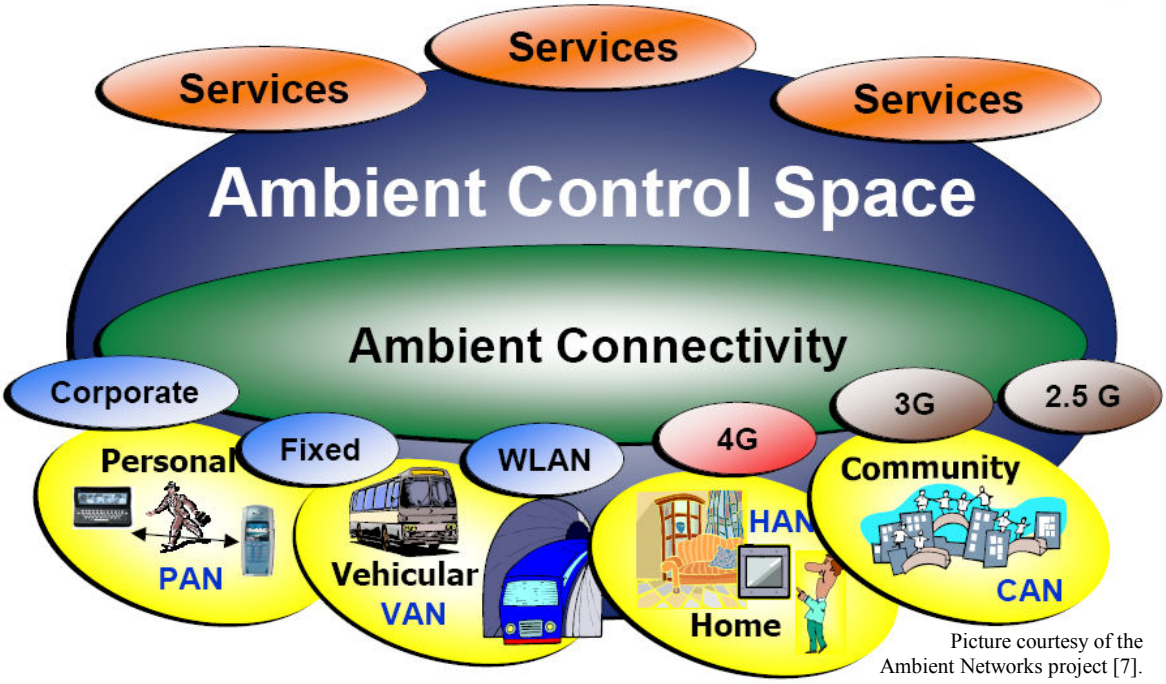


Figure 1: Overview of an ambient network

2.1.2 Ambient Control Space

An Ambient Control Space (ACS) is the core of an ambient network and it acts as a control layer on top of different network technologies. It provides a means of internetworking between different technologies and offers services such as mobility, QoS, and security. All Control functions or Functional Entities as they sometimes are referred to, in an ACS cooperate in order to implement the complete control functionality. The number of control functions can differ from ACS to ACS, but each ACS must support a minimum set of functions in order to operate properly, this defines the so called minimum ACS. This is the smallest complete ACS and it contains at least the mandatory functions, such as plug and play management, basic security, continuous connectivity, and composition control, while optional functions such as ContextWare are absent. A single device could host its own ACS. In this

thesis the ContextWare functional entity is the most important entity as it handles context dissemination and aggregation.

2.1.3 Ambient Interfaces

An Ambient Control Space provides all the control functions, but without being able to offer them to others they are of little use. Depending upon the target, there are three different interfaces defined for an ACS to communicate via. Each of these interfaces provides a single reference point to the outside for their specific purposes, thus making it easier for applications and services to make use of their functionality.

- Ambient Network Interface (ANI)

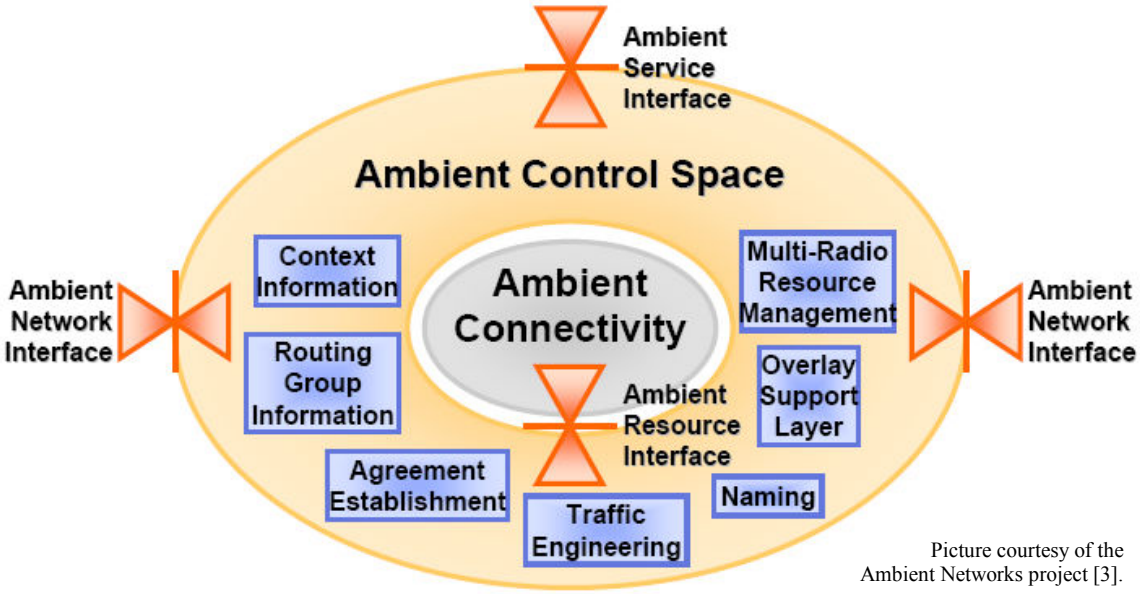
  This interface handles communication between two different ambient networks or within an ACS between different functions. Via this interface it is possible for two networks to create and share a common control space, i.e., to form a composition of two or more ambient networks.

- Ambient Resource Interface (ARI)

  The Ambient Resource Interface is used in order for an ACS to manage resources within the networks. The resources could be switches, routers, radio access-points, etc. The ARI is positioned between the ACS and the connectivity layer.

- Ambient Service Interface (ASI)

  Upper-layer applications and services use this interface in order to communicate with the ACS's control functions. Within a node this interface is located between an application and the ACS. Context information is available for applications using this interface. It also provides means for higher-layer applications to establish and maintain connectivity without having to be concerned with the underlying access technology.



Picture courtesy of the Ambient Networks project [3].

Figure 2: The three Ambient Networks Interfaces

2.1.4 Context-Awareness in Ambient Networks

In order for an ambient network to be dynamic, self-managed, and provide users with ubiquitous network access, context information needs to be collected and distributed. The collected context information is not only available to the network, but also available to higher layers applications and services in order to provide a more user-oriented environment. The main challenge we will focus on in this thesis is how protocols and networking functions can adapt based upon this context-information.

Ambient Networks support a common framework for handling context information both inter- and intra-domain. Today networks are missing this functionality as they do not use any context information. ContextWare is a synonym introduced by the Ambient Networks project for the system architecture that handles context management. ContextWare in the Ambient Networks project handles and distributes the context information within an ACS. It mediates between context consumers and sources in order to make context handling efficient.

Context information can be divided into two categories: user-related and network-related information. The user-related information includes location, identity, preferences, etc. While the network-related information is network identity, available QoS, etc. All the collected information describes the context of an entity. A central issue of user-related context information is how to control this information with respect to privacy issues. Controlling context information is going to be implemented with help of policies. This particular subject is under investigation in another thesis project carried out at Ericsson by Nupur Bhatia [8].

The ContextWare entity is further split up into entities with more specific tasks to solve. These are described below [9]:

- Context Coordinator

  The context coordinator's main purpose is to coordinate the information that is exchanged (i.e. it handles registrations of context sources), it acts as a gateway into the ContextWare architecture. It also authenticates and authorizes registrations and as well clients' use of context information. The coordinator is an active party when domains compose, when it performs the negotiations needed when composing and decomposing.

- Context Manager

  The Context Manager's main responsibility is to manage the data within Context Information Bases (CIBs) . It also handles updates of the context information to clients from sources and as well as process context information, for example, aggregation and filtering.

Along with the entities described above there exist other functions that either use the entities above or help them.

- Context Information Base (CIB)

  The Context Information Base (CIB) is a logical entity representing a distributed repository for context information collected from context sources. This is mainly used by the Context Manager and for example when sources delegate the context information or when it needs to be translated into other formats.

- Context Source

  A context source is the provider of original context information in an ambient network.

- Context Client

  A context client is embedded within a context sink in order to make it context-aware (i.e. it consumes the information provided by a source and acts upon it). Every entity that would like to be context-aware must implement context client functionality. Clients can be more or less advanced depending upon which services they wish to support and could be either user applications or functional entities of the ACS.

The Ambient Networks project has also defined a simple context protocol. In order for addressing context a new identifier has been defined, Universal Context Identifier (UCI), this is described in more detail in section 2.1.5. The protocol contains five primitives:

- REGISTER

  With this message a context source registers UCIs of its context information with the Context Coordinator.

- RESOLVE

  When a context client wants to get context information from an UCI this request is sent to the Context Coordinator.

- GET

  A context client fetches context information from a context source using this message.

- SUBSCRIBE

  With this message a client can subscribe to updates of some specific context information.

- NOTIFY

  The subscribed context information is delivered with help of this message when an update has occurred.

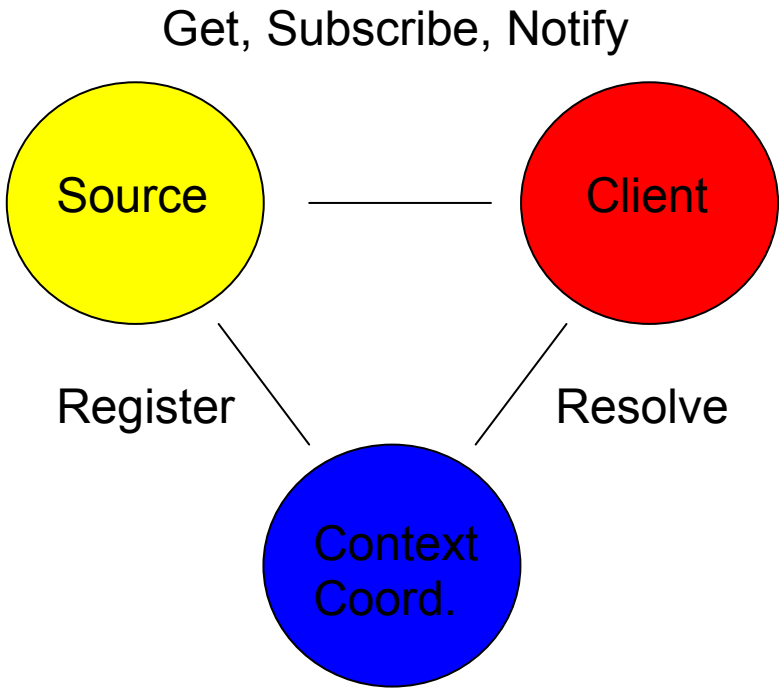


Figure 3: Relationship between the ContextWare functional areas and protocol primitives

Presented in the figure above is how the messages are used in interactions between source, client, and the context coordinator. The connection between the source and client might also utilize a proxy or database in between them in order to unburden the source. The protocol has to be extended in order to fulfill the needs of this thesis project. SIP [14] with its Specific Event Notification [5] extension was built on a similar base, thus the same fundamental messages are found in both.

2.1.5 Addressing context in ContextWare

The Ambient Networks project proposes the use of a new model for addressing context information [12], a Universal Context ID (UCI) is a new type of Uniform Resource Identifiers (URI) [10]. An addressing scheme is needed in order for any client to be able to locate context information and as well for sources to be able to publish it. A context source provides at least one context object and register the UCI of this object with a Context Coordinator. A UCI is a way to uniquely identify a specific context type, but not its location. There still exist open questions about how a node knows which UCI to utilize for specific context information and also how available UCIs are be distributed. Examples of such URIs are:

ctx://domain.org/path?options

ctx:/path?options

The first example shows the URI when an entity needs to ask about context information outside its naming domain; while the later is used when the requested context information is within the same domain.

A UCI only represents a specific context type, this means that there could exists multiple sources providing the same type of context information, i.e. there are several instances of the same type of context object.

The concept of UCIs is not fully developed yet; especially as the UCI namespace tree, paths and components are not yet specified. A lot of other questions are also unsolved and among them, is how to locate a device with the specific context information which is requested.

2.2 Context-Aware Networks

A heterogeneous network environment demands adaptive services based on user characteristics, services can no longer rely on a single network, as the users can access multiple (different) networks which vary in properties using one device. In order for a service or network to be adaptive it needs information to base its decisions on. This information could be context information, this is defined by Dey [11]:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

This definition is useful when real world objects as perceived by humans are in focus. But in Ambient Network the focus is more on network information.

When a network, service, or application makes use of context information and adapts itself according to the received information it is said to be context-aware. The term context and

context-aware is used as Dey points out as a term for referring to location, identities of people and changes to this information. Others explain it with illustrative explanations or synonyms such as environment or situation; this does not cover the whole area of context and therefore a new definition was introduced.
The Ambient Networks project classifies context information into four different categories [12]:

- Human user context
  This includes information about a user's surroundings, such as location, identity, presence, or networks.
- Device context
  Device context describes features of a user's device. It could range from screen size and resolution to IP address.
- Network context
  General information about the network such as network identity, bandwidth, and supported services.
- Flow context
  Context in this category express the view of the interaction between user and network. Information could include which type of application is running that produces the flow and state of links or nodes that transport the flow.

The context information could then be characterized based upon the relevance, churn ratio, how context is stored, how context is delivered, etc. This characterization is needed when context information is managed within a network. Different categories of context information are more important for some and less important for others. Problems that arise are how context information should be handled as how, which context information will be stored where, how it will be spread, etc.

- Storage
  *Information could be either **permanent** or **temporal**. Permanent information is for example username which is permanent and no updating is needed. Temporal on the other hand is likely to be changed and could be location.*
- Evolution
  *Evolution is how fast information changes and it is divided into **static** or **dynamic** information, where static represents information that is seldom changed and dynamic represents a more rapidly changed approach.*

- Relevance for a service or application
  *This could be either **necessary**, which represents information needed for a service in order to run properly, or **accessory**, which is only information that provides extra information in order to provide a better service.*
- Interaction
  How the interactions between a context source and consumer take action. Either the source periodically pushes out the context to the consumer or the consumer has to pull the information from the source.

2.3 Adaptive & Content-Aware Services

The Adaptive & Content-Aware Services (ACAS) project [13] has divided the service architecture framework into three different levels. The lowest level, the sensor level, is where the context information is produced by sensors and expressed in a context information language. The top level is the application level where the user's application can benefit from context information. In between these two layers is the general level which consists of the context information network and is responsible for acquisition, managing, and distributing of context information.

The core of ACAS's context information network is Context Management enabled Entities (CME). Applications connect to these entities to receive appropriate context information. The context information is delivered to the CMEs from the sources. Each device implements a local CME for collecting and managing its context information. A local CME connects to another CME which could represent a user's, a network's, or a domain's complete set of devices in order to distribute local context information and receive information from other devices. The network of context information can expand by inter-connecting CMEs. In order for the user's personal CME to offer subscriptions and context information to other applications (which are assumed to be executing on hosts attached to the Internet), the personal CME is expected to run on reliable (probably stationary) machines.

The ACAS project chose the IETF protocol SIP with SIMPLE's extensions to develop their framework. Each CME uses SIP/SIMPLE to address and access other CMEs; while using the Content Data eXchange Protocol (CDXP) to transport the actual context information.

The ACAS approach is in general a peer-to-peer method where different CMEs tries to find each other and exchange relevant context information. Although the CMEs can be distributed and even replicated, they act as super nodes in a peer-to-peer network where applications and sources can use them as gateways to other CMEs.

Figure 4 shows the concept ACAS has developed. An application gets information from a source which is connected to another user's CME. The user CMEs are then interconnected via a network CME in order to exchange information.

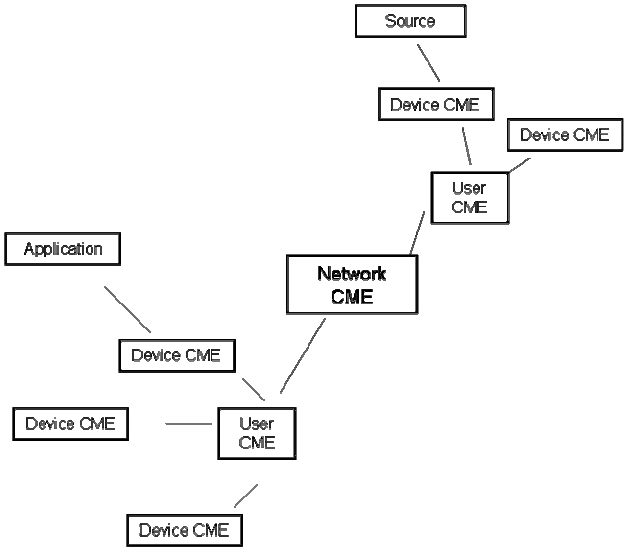


Figure 4: ACAS network view.

3 Related Work

This chapter discusses work related to this thesis. It starts with a general approach to distributed systems and then move into more specific areas such as a distributed hash table algorithms and peer-to-peer systems.

3.1 Context Exchange Protocol (CXP)

Sergio Quintanilla Vidal [4] in his master's thesis designed, implemented, and evaluated an architecture and a protocol for the Ambient Networks project. He adapted the CXP protocol from ACAS into a protocol for exchanging context information between entities. His solution was a client-server scenario where each node must communicate via a central server in order to receive or send any information; i.e. direct communication is not possible between pairs of nodes. The protocol was partially influenced by SIP [14], but was optimized for less powerful devices by having the protocol only handle a small number of different messages.

When Vidal did his report the UCI discussion had not progressed as far as described in section 2.1.5. Therefore did Vidal come up with a different naming scheme. In his solution the location of an entity and the location of context information were easily visible. The first part of the address was similar to an e-mail or SIP address (user@domain); while the second part used a taxonomy tree, much as the Simple Network Management Protocol (SNMP) does. A full address, a so called context URI, locates an item of context information and could look like:

router12@company.org/network/performance_load

This addressing approach does not fit the Ambient Networks project as it interferes with the intentions of having the URI act as unique and well-known names for context object types. It is important to remember that a UCI is the link between client and sources, i.e. whenever a client wants to access specific context information it has to know the UCI of the object type representing this information.

3.2 Distributed Systems

Distributed systems are a collection of independent computers, processors, or processes that in some way cooperate, for example communicate with each other in order to exchange information. A popular project is the SETI@home project [15] which encourages users to download an application which analyzes a small amount of the total amount of observational data from space in search for life outside the earth. As people use this application without compensation it is a very effective and cheap way of analyzing large amounts of data compared to buying a supercomputer. Gerard Tel [16] indicates that distributed systems may be used, for the following advantages:

- Information exchange

  The need for exchange of information between computers has grown since the early days of computers. These demands were

created as people wanted to exchange data in order to be more effective in their profession and because not every local site had sufficient resources for all the local demands. This motivated the creation of many different types of networks, such as local-area networks, wide-area networks, etc.

- Resource sharing

  Sharing of resources could involve all kinds of resources. The most popular resources shared are printers and disk space. Another approach of resource sharing involves replication, where data is replicated over several nodes instead of their being only one copy. This help when a node leaves the network, thus it may still be possible to access the data despite the absence of one or more copies (up to the limits of replication). This help to protect against denial-of-service attacks, as if one node is out of service due to a large number of requests, it still might be possible to utilize the requested resource at another node.

- Increased reliability through replication

  In order to increase reliability a single node can be replicated (i.e. one node is replaced – so that two or more nodes do the same work as the original node). When one of these nodes fails the system still works if and only if the other nodes are still operating correctly.

- Increased performance through parallelization

  Tasks that require a lot of computational power benefit if processing power can be aggregated. This can be achieved by distributing the sub-tasks across a number of computers.

- Simplification of design through specialization

  Instead of putting all functionality in every node there could be nodes which specialize in doing a specific task. This result in a system built up of different modules, this may be easier to manage and implement. On the other hand it might cause less replication as there are only a smaller number of specific nodes which holds some functionality.

- Scaling

  Distributed systems have the advantage of scaling as number of nodes increases, as each node brings additional processing when joins the network, hence it adds resources to the whole network.

Distributed systems have continued to evolve and in recent years their architecture is increasingly utilizing direct node to node communication, so called peer-to-peer networking. Typically the reason for considering all nodes to be equal is to avoid dependency upon a central server. However, there exist peer-to-peer solutions which also utilize central servers. Additionally, instead of a physically central server all nodes together create a logical central server with the same functionality, but in a distributed fashion. The main characteristics of these new systems are that all participants contribute resources when they form a network, preferably self-organizing as nodes join and leave the network. A clear advantage is that there is no central point of failure (i.e. no central server). This is further described in the following sections where the first section (3.1) describes peer-to-peer systems in general, then the three following sections (3.2 - 3.4) describe first to third generations of peer-to-peer networks.

3.3 Peer-to-Peer

Peer-to-peer (P2P) systems are a distributed system where each node participating in a network has the same potential functionality and responsibility. P2P techniques have evolved during recent years and can now be found in several applications. The area where most people have heard about P2P is in file sharing applications using the BitTorrent protocol [17], but P2P techniques are also used in applications such as Skype [18] and SIP [14]. An important idea behind P2P oriented architecture is the avoidance of a central server, although mixes exist where a central server handles some tasks and peer-to-peer techniques is used for other tasks. SIP is often used together with a central server for finding other users, but the actual session content is sent peer-to-peer.

Today, many instant messaging applications use a central server solution, where every client connects to the central server and all further traffic is routed via this server. This demands a lot of computing power in order for such a central server to respond to a very large number of clients. In contrast in a P2P environment each peer (also known as a client or node) acts as both a client and as a server. All the nodes together provide the service, without needing a central server. Each node might not be able to provide all services, but the collection of nodes cooperatively provides all the services.

In practice P2P nodes form an overlay, as they are interconnected via an IP network, but act as if they were directly connected in a virtual network on top of this IP network. An approach [19] that Skype and Kazaa use is to combine ordinary nodes with super-nodes, the later nodes provide some important features. Super-nodes are often used to help ordinary nodes that are behind NAT-boxes or firewalls to communicate and route traffic; this is possible because super nodes each have a global public IP address. Additionally it should be noted that Skype is not completely peer-to-peer as all logins are performed by a set of central servers.

The main problems with P2P networks are to handle how nodes should join and leave the network. This could happen very frequently, but the system must remain stable and continue to provide users with service. Locating data is a crucial issue in a P2P network as nodes and users might move and leave the overlay network. To solve these problems algorithms are introduced and implemented to provide effective data distribution. These algorithms must take into account that nodes might disappear and rejoin at any time, potentially with different network addresses. Therefore the data that one node stores must be replicated by others so that the system can adapt quickly when the topology changes or the network needs to be able to function without this information. These algorithms have evolved a lot since the first P2P networks appeared. Other issues that algorithms must solve are how to distribute the data of a P2P network among the participating nodes, in order to provide load balancing. Another problem in a P2P environment is that malicious nodes could insert incorrect information into the network, which could lead to denial of service to some nodes.

3.3.1 Bootstrapping

Bootstrapping is a problem that appears in every P2P system and must be solved. This concerns how to enable a new node to join an overlay. The main problem is how to find the network address of at least one node already in the overlay. A node ‘A’ can not be sure that a node ‘B’, which it contacted the last time it joined the network, is still connected (this is due to the dynamic nature of P2P networks). Two methods are presented below. Recently, the Ambient Networks project proposed a new Functional Entity, which handles Self-Configuration issues. This could eventually take care of the bootstrapping problem.

3.3.1.1 Static overlay node(s)

Every node that wishes to join a P2P network must know at least one of the static nodes that exist. These nodes are hard-coded within the application as fully qualified host names (these names can then be resolved into an IP address using DNS). An obvious question is how this solution scales; it requires some administrative work due to its centralized approach. Dynamic DNS [20] could be used to more efficient update the DNS records. Although there still have to exist some central nodes which handles these updates. A positive effect of using some centrally administered nodes is that it solves the authentication problem in P2P networks, since these central machines could have an authoritative list of allowed users.

3.3.1.2 Broadcast / Multicast / Anycast

A node wishing to join a P2P network sends a multicast message to a well-known multicast group and in reply learns a network address of a node within the overlay. This is better than sending a broadcast message as the broadcast message would be processed by all nodes, even those not participating in the overlay (i.e. multicast group). Multicast could also be used for handling lookups instead of the DHT, although this demands that all nodes listening to this specific multicast address would have to process each request. In the future, IPv6's Anycast [21] might be used rather than multicast, thus offering some of the advantages of central administration while still enabling dynamic discovery.

3.3.2 Centralized systems

First generation peer-to-peer networks consisted of a central server which every node registered their information at and then a requesting node simply asked this central server in order to learn where the content is stored. Once it learned where the information was, a peer-to-peer transmission started to enable the requesting node to access the content from the storing node. There are advantages of this solution, as every node only needs to ask the central server in order to get results. However the load on the central server increases with the number of users. Other advantages are that it is easy to control access to the content stored in the network and as well being easier to perform complex queries – since the central server knows about all the content. An obvious drawback is that there is a single point of failure; thus if the central server does not work properly, then no node is able to successfully make any queries.

3.3.3 Flooding systems

An approach which is opposite from centralized systems are the systems based on flooding when requesting data. In order for a node to find some specific content it simply asks a lot of participating nodes. In this case each node only knows about information which they themselves store. In early networks when a node asked its neighbors for information; if the neighbors did not have the requested information they forwarded the query further. This system avoids central servers and the network consists of participating nodes which is equal. If the requested data is found at a neighbor, then this neighbor responds with it and does not forward the query further; however none of the other neighbors were notified. This approach utilizes flooding; each request contains a time to live field which limits the flooding. These flooding systems generate a lot of traffic due to the inefficient way of searching for information, but they avoid of not having a central server. Kazaa is built this way, but was improved by adding super-nodes that act as smaller central servers or repositories in order to minimize the flooding of messages. In a flooding based system it may be hard for a node to find the requested information as the search scope is limited by the time to live field; thus if

too small time to live field is chosen the query might fail even though there actually is a node that has the requested information.

3.3.4 Distributed Hash Tables

A popular approach in modern P2P solutions is to use a Distributed Hash Table (DHT) [22] [23] which helps a node locate the node that stores a particular data item (i.e. the lookup only gives a node the **address** of the requested data). This type of solution is also called a structured peer-to-peer solution. The idea behind a DHT is that every data item and each node in a P2P network has a unique ID. Nodes and data share the same ID space and are distributed equally which provides load balancing. The unique ID is created by hashing a specific keyword or address. The data is then divided and assigned to particular nodes; each node's id corresponds to a range of data IDs. When later a node wants to access some specific data it performs the same hash and with this hash it can locate the closest node that stores the data, based upon the node with the ID closest to derived hash. If the node does not know exactly which node holds the requested data it may have to pass on the question to a node that is closer to the data which can assist in the search (i.e. a node with a node ID closer to the hashed ID). There are a couple of DHT algorithms to choose from such as Chord [24], Pastry [25], CAN [26], etc. The main difference between them is their overlay topology; which ranges from circular overlays via meshes to d-dimensional Cartesian spaces. These algorithms must also handle (re)composition of network, i.e., when a node joins or leaves, and as well connecting new nodes into the overlay. DHT algorithms are used for looking up data, i.e., as the hash does not necessarily provide the requested data but only provides a pointer to the data.

Figure 5 shows that the central server approach requires least number of messages in order to locate objects, but on the other hand it requires more computing power and storage in order to be able to serve all the connected nodes. In contrast flooding requires a very large number of messages when searching, but each node only needs to store the information it publishes by itself and flooding offer the fastest means of finding the information if it exists anywhere in the network. The third and most appealing solution is distributed hash tables which scale very well and do not require a lot of overhead when searching. For example, each node in the DHT solutions which are logarithmic only needs to know O(log N) other nodes, where N is the size of the ID space in the overlay, in order to be able to search and manage the network. Nodes keep this list of other nodes in some sort of routing table, depending on which solution is chosen. As the IDs of data and nodes are equally spread thanks to the hashing algorithm, then the load between the nodes are also balanced, this also helps when some node leaves the network as no node is more important than others. The only drawback with DHT solutions is if complex queries are needed and not only specific references, then flooding systems are better - as you can search using smaller parts of the search string compared to the DHT where an exact match is the only possibility since the search string is a calculated hash. A hashed value is unique and has no correlation to a similar value. An example is shown below where the popular function Message Digest V5 (MD5) [27] is used.

MD5("123") -> 202cb962ac59075b964b07152d234b70

MD5("1234") -> 81dc9bdb52d04dc20036dbd8313ed055

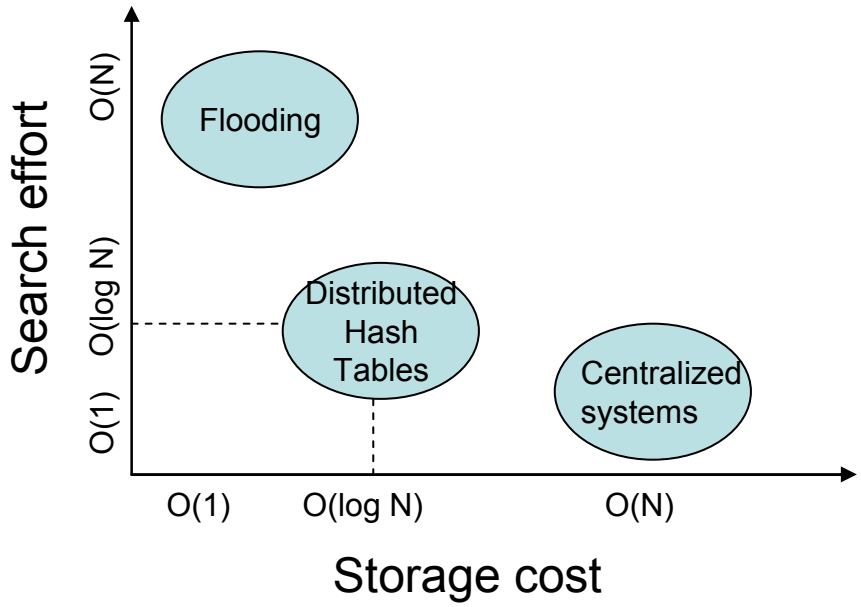


Figure 5: Comparison of different distributed approaches

Common for all DHTs are two primitive operations, put(key, value) and get(key). The put operation is used when a node want to publish some information, the key field is the hashed unique ID of the value that is going to be published. The value field could contain whatever information a node wants to publish, but most often it is a network address of where a larger amount of data could be found. The get command works in the same manner, the requesting node calculates a hash of the unique ID of the information it wants and then send the request. Below is a scenario describing how an ambient network could work which makes use of a DHT algorithm:

> *The context coordinator (described in section 2.1.4) is a logical entity which is in this case distributed over all participating nodes, they could be context sources, sinks, or managers. The scenario is built on one of the described DHT's; in this description there is no significant difference between the DHTs.*
>
> *When a context source has some context information it wants to share, it hashes the UCI and then in pairs (UCI, contact information (i.e. IP address and port of origin)) sending the information to the corresponding node (i.e. the resulting hash of the UCI tells which node to send the information to). When a node later wants to resolve some specific context information it hashes the UCI, accesses the information via the P2P overlay and then retrieves the context information.*

When a DHT network becomes big with nodes spread all over the globe the lookup times might increase together with longer latency times as node IDs are assigned randomly. Typically node IDs are assigned randomly without take into account how the topology is situated. A DHT could have an ID space which either is topology aware or random based, most of the DHTs today are random based. A paper [28] presents a solution to embed locations within the node IDs to make DHTs topology aware without losing scalability and load-balancing. The authors present an idea of structuring the ID space with help of dividing it in regions. This means that the ID space is hierarchically and only the last part of the ID

space is random in order to combine load balance and scalability with reduced lookup and latency times. The sizes of each part together with how many levels the ID should be divided in is related to how the system that uses the method looks like.

Figure 6: Structured ID space

3.3.4.1 Recursive vs. iterative

A DHT algorithm may be either recursive or iterative. The difference between recursive and iterative methods are presented below. The main difference concerns **where** the routing decisions are taken.

In a recursive method (see Figure 7), the requesting node leaves the decision, of where to forward a request, to other parts of a network (i.e. other nodes). This may be compared to a black-box, where the origin sends a request to a black-box and then receives an answer without knowing how the answer was found or from where it was received. When a request has left a requesting node the packet is forwarded from node to node until the requested object is found. The response could then either be sent direct to the source or routed back in reverse along the same path. The difference between the two methods is shown in the figure below, where the direct response method is shown to the left in the figure 7 below. A recursive method limits the load on the requesting node, but demands more responsibility from remote nodes as these have to route messages further to other nodes. When a node in recursive mode sends a request, it does not know whether the request is subsequently routed or if some node fails. The method to the right, where the response is routed to the source along the reverse path, provides some degree of anonymity as each node only knows where it got the request from and how it responded to the request.

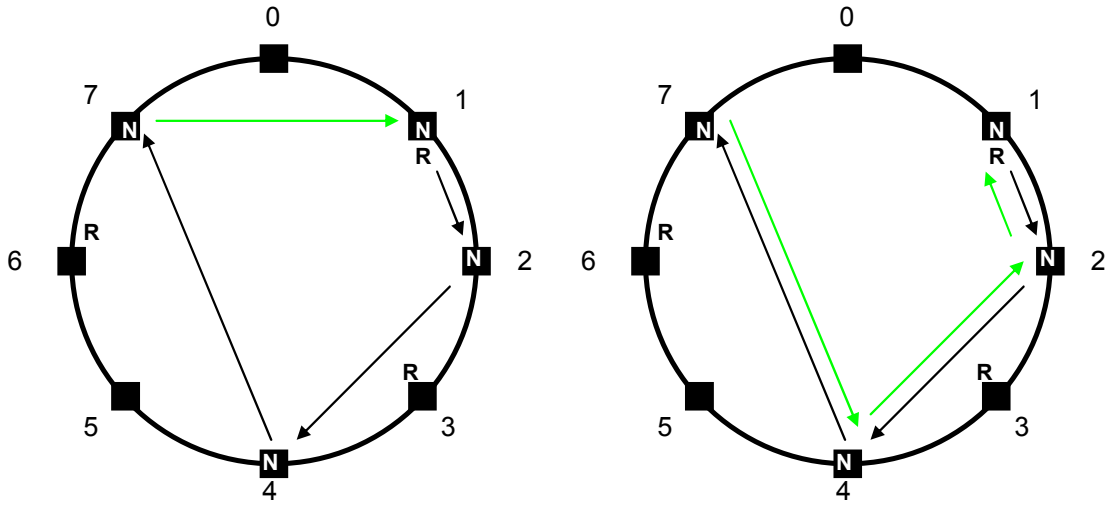


Figure 7: Recursive lookup methods

An iterative method (Figure 8) always sends requests from the origin. Each request always receives a direct reply to (i.e., replies are sent back to the requesting node). The requesting node then has to start over again by sending a new request to another node if the requested object was not found. This method has the advantage of controlling of how the request is routed as the origin sends the requests. This method also provides a means to discover node failure; if a request is not responded to, then the source adapts more quickly to this situation

than in the case the recursive method. Searching is also be better controlled as the requester knows which parts of a network has been searched and need not depend on other nodes. In the figure below the requesting node first tries to request the information from node 2 and then from node 4, but in both cases lead to negative responses. Then when it finally tries to request information from node 7, to which it receives a positive reply.

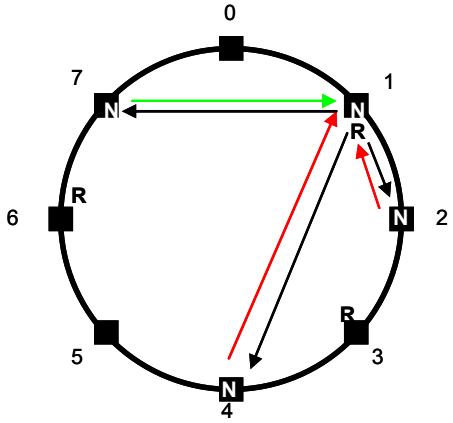


Figure 8: Iterative lookup method

3.3.4.2 Which DHT?

When designing a structured P2P network the hash algorithm is an essential part of the design. Which algorithm to choose and why depends on a lot of factors, such as the potential size of the network, lookup speed, security, replication, etc. Little research has been done comparing different routing algorithms based on distributed hash table solutions, in order to evaluate them. In a document about using P2P techniques in SIP [29] the authors selected Chord because of its simplicity, convergence properties, and its strong reputation in the P2P community. Research that actually evaluated different DHT routing geometries supports the decision of choosing a ring-based structure as Chord [30]. The authors of this later paper point out that the ring geometry has flexibility in order to choose neighbors and next-hop paths and as well achieved the highest performance when resiliency was tested. The conclusion is; "why not using ring geometries?". Similar reasons of choosing Chord are recently made in a dissertation by Ali Ghodsi [31], where he chooses Chord because it being well-known, thus making it attractive for pedagogical purposes. Ghodsi uses an algorithm in his work called Distributed k-ary System which is influenced by Chord, he is studying lookup consistency, group communication, bulk operations, and replication. All of these areas are interesting and might be used to further improve the design in this thesis.

In this thesis a prototype is going to be developed using a local area network and with limited number of devices and in a limited ID space. Anyone of the DHT mentioned in section 3.3.4 could be selected for this prototype as they are similar in how they function. During the development of the prototype effort is put to make it easy to replace the selected DHT with another. This is done as the development of different DHT is a new area and not much research about comparing different DHTs has been done and probably new and more efficient algorithms will be developed within the years to come. The selected DHT for now in this prototype is based on Chord and is presented in more detail in the following section.

3.3.4.3 Chord

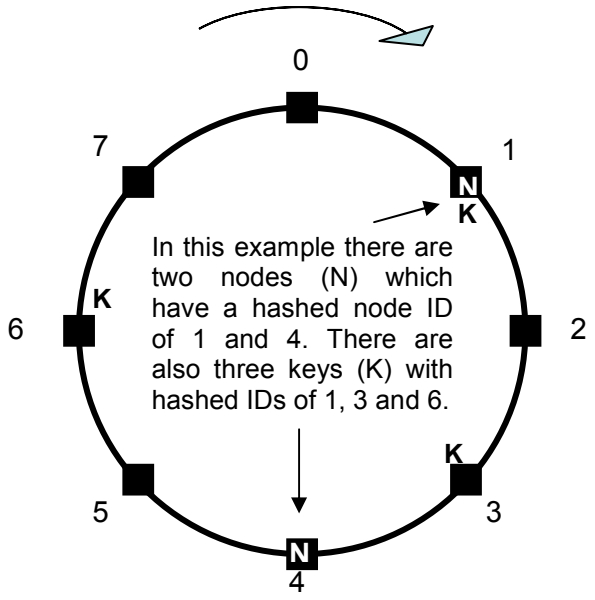
Chord [24][32] was developed in a project at Massachusetts Institute of Technology [33] and is similar to and built on a solution named **consistent hashing** [34]. It is a protocol for

looking up a specific node responsible for a given key. In order for an application to make use of Chord it has to map each key to/from a desired value. The value could be an address, file, document, etc. All nodes and data in the overlay are placed in a ring structure; without any node being more important than another. The keys are distributed equally among the participating nodes (i.e. load balancing). The assigned ID of each node and key is derived using a hash-function, such as SHA-1 [35], typically each node's IP address is used while in the case of a key the key itself or filename is used for hashing. Each key is assigned to the first node whose hashed value is greater than or equal to the key value (i.e. if all nodes and keys are placed in a circle of numbers it is the first node clockwise from the key). The nodes and keys are spread out in the ID space thanks to the hash function's result changing significantly even if the input is only changed a bit. The ID space in a Chord system is from 0 to 2^m-1 where m is the number of bits in the assigned ID of a node or key. Chord is random based which make the node ID assignment random and a node close to another in the ring does not mean that they are close in the topology. Below in figure 9 is the Chord algorithm presented visually with an example.

A node keeps record of its predecessor node together with one or more of its successors in order to maintain the overlay when nodes join and leave. Instead of always asking the next node for a key Chord has introduced a routing table, called a finger table in order to reduce look up times and to accelerate the lookup, thus it is possible to ask a node that it is not the closest neighbor. The size of the finger table is recommended to be maximum the bit length used by the hashing algorithm. This is typically 160 entries in Chord as they recommend using SHA-1 which has a size of 160 bits. However, it is not optimal to host a 160 entry finger table for a smaller network. In smaller network it would be better to keep the finger table small, as this enables a node to be more efficient. The optimal size is often less than 160 bits; it depends on the number of nodes in the overlay. The recommended number of entries are O(log n) [32], where n represents the number of nodes in the overlay.

Each entry in the finger table holds information about a node such as an IP address and port number. The i-th entry in a finger table is the first node that succeeds $m + 2^{(i-1)} \bmod 2^m$. This provides each node with a list of successor nodes starting from its closest neighbor and then gradually becoming more seldom. If the requested node is not directly localized via the finger table, then the asking node has to ask the node with the ID closest the requested node. A node also has to update its predecessor node and as well its successor node.

As described above (see section 3.3.4.1) a lookup can be routed in a Chord algorithm in two ways, either recursively or iteratively. The decision about which method to use needs to be taken in the beginning of the design phase - as it is fundamental how the system operates. In a recursive lookup a node asks the corresponding node in its finger table and if that node does not contain the requested key it forwards the request further. In an iterative method the requesting node manages most of the work, a request is sent out to the corresponding node and this node responds with the location of a closer node, if it not possesses the requested information. An iterative solution generates a few more messages than the recursive method, but it reduces the burden upon uninvolved nodes as they only have to handle two messages instead of up to four in the recursive method. This is especially important in ambient networks where devices such as mobile phones and other units with limited processing power appears.



In this example (m=3) the ID space is from 0 to 7. The finger tables:

i	Node ID = 1	Node ID = 4
1	2 → 4	5 → 1
2	3 → 4	6 → 1
3	5 → 1	0 → 1

If node 4 wants to retrieve the resource with ID=6, then it looks into its finger table for the closest match and finds that it should contact node 1.

Figure 9: Picture explaining Chord

3.4 SIP P2P

SIP [14] based communication has so far used a central server in order to lookup a specific user. This central server is a single point of failure and in a draft [29] to IETF this central node is removed and replaced with P2P architecture. The draft covers both the SIP standard and the add-on SIMPLE [5] which handles instant messaging. The urge to move away from central servers could be of many reasons. Not only to move away from the problem a single point of failure case as it is with central servers, but also smaller groups of people might think it is convenient to avoid setting up a server, rely on third parties, or being connected to the Internet.

Their solution is built on a third generation P2P network based on distributed hash tables. In this case they use an algorithm, which is a variant of Chord, along with iterative search, although they state that any DHT algorithm could be used, as no algorithm is standardized yet.

A SIP P2P node must be an active member of a P2P overlay and provide a minimum of server-like functions, compared to an ordinary SIP node which only acts as a client. This means that a node is required to be active by providing functions for communication which it does not participate in. The functions that need to be implemented could be implemented in different places. The most obvious place to implement the server functions is within the node itself, but it is also possible to implement it in a proxy – while ordinary SIP clients interact with a P2P overlay.

The solution is as mentioned earlier based on Chord, but all the messages needed to maintain the DHT are SIP messages. Each node gets its node ID by hashing the node's IP address together with the current port. Every resource, for example a registration, also gets an ID which is a hashed keyword that identifies the resource. Then, as described above, the resource IDs are spread out among the participating nodes according to the Chord algorithm. The node's ID before hashing should be in the form IP-address:port (for example 192.168.0.1:80).

There exists two classes of messages; messages that implement the DHT in order to let nodes join and leave the overlay. The other class of messages is the usual SIP messages which

enable users to register, invite others to a session, etc. The DHT implements the function of a registrar; while communication between two user agents takes place directly as in traditional SIP. The message to manage the DHT overlay is a regular SIP message, the register message.

When a node first wants to join an SIP P2P overlay it needs to find one node to contact, a so called bootstrap node, this could be any node participating in the overlay. The draft has solved this issue by assuming that all user agents know about a limited number of well-known nodes. The joining node calculates its node ID, and then contacts one of the bootstrap nodes with a register message. The bootstrap node responds with a 302 redirect message requesting the joining node to contact a node closer to the node which is responsible for the ID space the joining node occupies. The admitting node, which is the node which is currently responsible for the ID space the joining node occupies, then exchange messages with the joining node in order to divide the ID space between them and to give the joining node information about neighboring nodes. Messages are then sent periodically between these two nodes in order to maintain the DHT and to handle nodes joining and leaving the overlay.

After a node has registered in the overlay it also must register the user agent in the SIP P2P network. This is done much the same way as traditional SIP registering, but the registrar is distributed instead of a central server. The registration creates a link between a user's SIP URI, a resource, and the user agent's current network address. Routing of resource registration messages are done much the same as when the node registers in the overlay. The term resource corresponds to a user name in P2P SIP, but could in other scenarios be any keyword. The resource ID is calculated, and then sent to the node with the closest node ID in the origin node's finger table. If that node is responsible for the specific range then it replies with a 200 ok message, otherwise the resource node gets a 302 redirect message in return and then follows the redirection. Redundancy is possible to implement as the node can register each user several times. In order to get a different hash the user has to add a replica number, starting from 1, to the earlier keyword and calculate a new hash. Then the new hash, which is not related to the earlier, is sent to the corresponding node. A minimum of 2 replicas are suggested.

When a user agent wants to contact another user it has to make a hash of the other user's URI in order to get a resource ID and send it in a register message. Then the source node looks into its finger table in order to find the node with the id closest to the resource ID, which might be the node itself. If the contacted node is not responsible for the ID it issues a 302 redirect message with information about a node that is closer. Then the searching node has to issue a new request to the node which it has been redirected to. This process continues in this iterative way until the responsible node is found. Then the node replies with a 200 ok message containing the contact information of the asked user or a 404 not found message if the user is not found. When the contacting user has received a 200 ok message it tries to contact the other user according to standard SIP procedure with an invite message. The contacting user agent should, to be sure that the received resource is accurate, also ask at least one replica in order to verify the data. The replicas are spread out to nodes that have no relationship and therefore an attacker must have access to a lot of nodes in an overlay in order to corrupt resource registrations. The risk of compromising data in order to change registrations and hijack connections are lesser as the overlay grows and the replicas are more and more spread out among the participating nodes.

In order to distinguish a SIP P2P message from a regular SIP message a new option tag is introduced; dht. All nodes must include the dht tag in all messages that are intended to be

processed within a SIP P2P overlay in order to learn about P2P presence. The authors of the draft recommend the use of SHA-1 algorithm for making hash calculations with a 160 bit output. But in order to use other hash algorithms the authors also propose that the name of the algorithm is specified in a separate header. This sort of information leads to two new headers; DHT-NodeID and DHT-Link. The first is used for general information such as node ID, hashing algorithm, overlay name, etc. While the latter contains information about predecessor and successor nodes together with finger table.

Open issues in the draft which the authors point out are naming issues; for example how names are allocated and protected. NAT-boxes cause troubles as a lot of people behind NAT-boxes have local IP-addresses, thus they have the same node ID since the node ID is a hashed version of the IP address of the node.

Information that is sent to a user who is offline, such as voicemail or instant messages needs to be handled. The information must somehow be stored until the user is online again and when it is possible to deliver it. However, the use of SIP P2P requires that the DHT overlay only handle resource mapping, while sessions use the standard protocols and methods. An interesting solution to this problem is presented in the paper called "Providing Secure Services in Peer-to-Peer Communications Networks with Central Security Servers" [36]. The solution is similar to Skype in using a central server for handling authentication, then using super nodes together with ordinary nodes. As the only time a user needs to contact the central server is when logging in and when they want to retrieve or store offline material the load on these central servers are low. Another benefit of using a central server to handle authentication is the possibility to use a public-key system which makes it possible to easily encrypt media traffic, verify user address, and prevent spoofing of identities. The paper presents a solution for sending voicemail to an offline user. This is done with help of cryptographic keys. The sending user starts to create a session key with help from the authentication server, then the user provides the server with the session key together with the receiving user's username and a resource ID. The actual voicemail is encrypted with the session key and sent to a storage location based on the resource ID. Later on when the receiving party logs into the authentication server they are notified about having a new voicemail and the session key together with the resource ID is sent. Then the receiving party lookups the node responsible for the resource ID and fetches the voicemail and with help of the session key is able to decrypt it. This approach solves a lot of the problems that are mentioned above and more described in the draft. The obvious problem with the naming space are solved as a central authentication server handles all logins and authenticate only trusted users.

4 Distributed ContextWare Architecture

In chapter 2 the background of context information and specifically Ambient Networks was introduced. As mentioned in the problem statement in chapter 1 a distributed architecture of ContextWare is going to be designed and implemented. This chapter explains the proposed architecture which combines Ambient Networks with distributed system together with why several decisions were taken. It begins with giving details of the design and how the proposed architecture fits into the Ambient Networks project.

The Ambient Networks project WP-D has defined the entities that comprise the ContextWare architecture. The WP-D defined entities are:

- Context Coordinator (CC)
- Context Manager (CM)
- Context Information Base (CIB)
- Context Source
- Context Sink

Each node or set of nodes comprising an ambient network must be able to host a minimum of server-like functionality (CC, CM, and CIB functionality) in order to provide context aware functionality for communication it is **not** involved in. The requesting node (or nodes) in an ambient network hosting these functional entities have to, with help from the UCI, resolve an address of where the Context Information Object (CIO) is located and then send a fetch request. The entities which were described above in section 2.1.4 are compatible with the architecture and entities which were defined in WP-D. In our distributed approach to ContextWare, compatibility with Ambient Networks WP-D is preserved by the following entities:

- Context Coordinator

  The context coordinator is distributed within the DHT overlay and handles registrations and resolving of UCIs.
- Context User Agent (CUA)

  The Context User Agent (CUA) is a re-interpretation of Context Sources and Sinks. The context user agent runs on every node that wants to be a part of ContextWare and provide or consume context. It provides means to interact with the DHT overlay either on behalf of an application or service outside an ACS, or implemented within an ACS in a source and/ or sink.
- Context Manager + CIB

  A context manager uses a CUA in order to register and resolve context information in the DHT overlay. A context manager aggregates and process context information either in a static way or dynamically. A source can also delegate the handling of subscription and notification to a context manager.

4.1 Design principles of the ContextWare

The Ambient Networks project proposes a distributed ContextWare to be used for aggregating and disseminating context information to functional entities and management functions within Ambient Control Spaces (ACSs). This choice is based on the capabilities and characteristics of a distributed ContextWare, i.e. distributed storage capabilities, as well as its ability to deal with churn (rapid changes in network composition), storage redundancy, and scalability. Distribution of ContextWare also helps to avoid the problems of a single point-of-failure solution, which Vidal [4] has presented. An ambient network contains a large number of wireless and handheld devices with limited power, processing power, and bandwidth, thus a distributed ContextWare must limit the signaling required and use few resources least applications on participating devices suffer.

Although Vidal [4] bases his design of CXP on a central server, CXP can be transformed to operate in a distributed architecture which require a number of changes. The CXP message architecture is of primary interest and is further explained in chapter 5. The message architecture of CXP was used as inspiration when designing the distributed ContextWare and a new distributed protocol. The distributed ContextWare extends CXP by defining new messages to provide the means for a CUA to manage a DHT overlay. The DHT overlay acts as an overlay on top of a transport protocol which a CUA uses in order to collect and disseminate context information.

Moreover, a distributed approach to ContextWare moves the context information and especially the lookup of context information closer to the users as compared to a central server solution as the user devices are part of the overlay.

The design of distributed ContextWare allows all participating nodes to collect and disseminate context information, but they also have to assist other nodes in storing and fetching UCIs representing context information.

4.2 Ambient Interfaces

A context user agent can act as both a producer or as a consumer of context information, most likely it acts as a consumer and producer simultaneously. An application or service that uses the Ambient Service Interface (ASI) communicates with the CUA when resolving a UCI (in the case of a consumer) into a location of a Context Information Object. The CUA when it receives the resolve request uses the DHT overlay in order to locate the address to the source which holds the CIO. The Ambient Resource Interface (ARI) is used by an application or service (in the case of a source) when it wants to register a UCI with the Context Coordinator. This is done with help of a CUA that provide means to register the UCI in the overlay (i.e. distributed Context Coordinator).

The context user agent also acts as a node in a DHT overlay; logically a CUA is treated as the rendezvous point for CIOs by the applications and services requesting or registering context information (i.e. an application or service uses the messages of the ContextWare protocol (section 2.1.4) to interact with the CUA which then forwards the requests to the DHT overlay.

In the scenario above the context consumer and source are assumed to be in different ambient networks. Thus the Ambient Network Interface (ANI) is used in order to exchange information and also negotiate and perform context network compositions. This is described in more detail in following sections.

Figure 10 depicts the relations between the ContextWare, DHT Overlay and Ambient Networks interfaces that were discussed above:

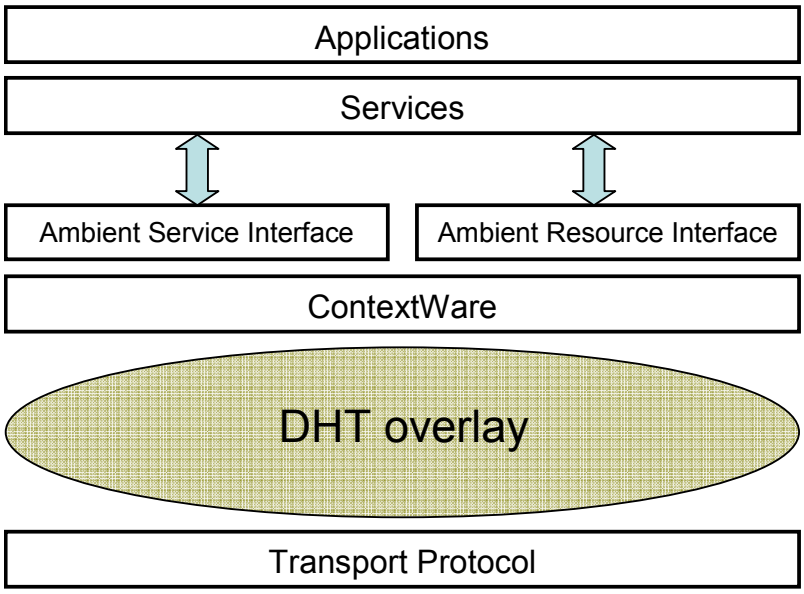


Figure 10: ContextWare viewpoint of Ambient Interfaces

4.3 Design Principles CUA participating in a DHT Overlay

The following sections explain the basic principles of the overlay. The DHT overlay is based upon the Chord algorithm [32]. The detailed reasons for choosing Chord is described in section 3.3.4.2. In this thesis the main interest is not which DHT is used, but how a ContextWare can benefit from using a DHT. The prototype is going to implement a variant of Chord adjusted to the limited number of devices. The prototype could have used any of the DHTs mentioned in section 3.3.4 as inspiration as they are similar in how they function.

A ContextWare uses a DHT overlay on top of a transport protocol to perform the tasks of resolving and fetching context information. Each participating node must run a CUA in order to be able to join a DHT overlay; this enables consumers or sources to use the underlying ContextWare (via the DHT overlay). A CUA provides two methods for applications or services on top of the DHT overlay: REGISTER and RESOLVE. These two functions are further explained below and are also referred to as "PUT" and "GET" in some literature [37][38].

4.3.1 Iterative decision

Ambient networks have a high ratio of wireless devices which might have higher latencies depending upon their environment and network, limited processing power, and as well entities might disappear from time to time. An iterative solution serves entities better as they instantly find out if an entity has disappeared compared to a recursive method where the origin node has to wait for a response from the black-box (i.e. a single latency time compared to multiple latency times). In an iterative solution the origin node can take action instantly and issue a new request to another node when one link is down or has high latency. Each query is handled by the requester without burdening the rest of the network. This also an advantage of an iterative method as the requester has most effort in the lookup and not put more burden than necessary on the rest of the network.

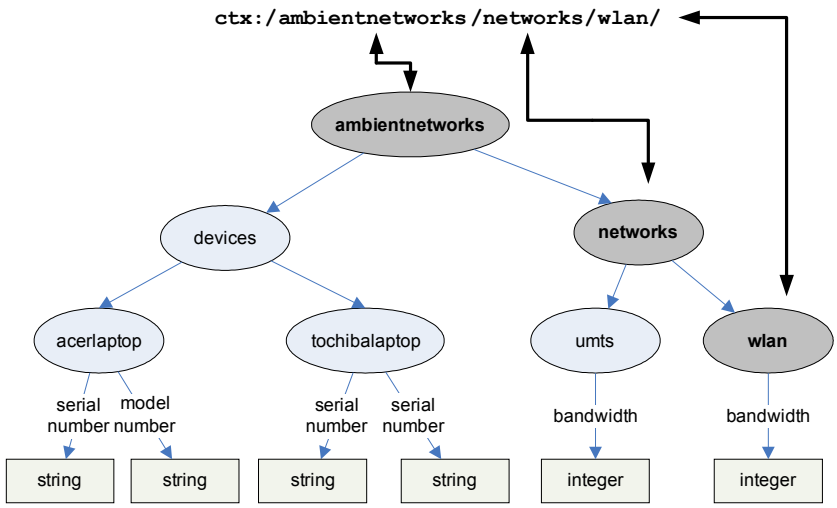
There might be scenarios where a requesting node in a recursive solution suspects that a message is lost, due to high latency time, and issue a new request. This puts even more loads to uninvolved nodes compared to the iterative solution where the sources have full control of latency times and node failures for each request being sent. Therefore is an iterative solution better suited to the requirements from distributing ContextWare within an ambient network.

4.3.2 Bootstrapping decision

The most suitable method for the protocol is to use a broadcast or multicast based discovery mechanism; if a node is within the DHT overlay it replies to this message with its network address and then at least one of the participating nodes in the DHT overlay known. This avoids having a central authority but still be dynamic. This mechanism could later be replaced by a new self-configuration functional entity which could provide means for a new node to get in contact with the DHT overlay.

4.4 Addressing

A UCI is well-known and acts as a name for a data object containing context information at a source or a manager. The path-component must be standardized in order to allow access to well-known properties. Naming conventions of context information (described in section 2.1.5) together with methods of how to assign a UCI or how to disseminate available UCIs are still an open issue within the AN project.



Picture courtesy of the Ambient Networks project [9].

Figure 11: An UCI address tree

Each of the branches in the figure must be resolved. If for instance a resolve message is issued with the UCI of ctx://ambientnetworks/networks the response would be both UMTS and WLAN.

The UCIs that are used are based on the UCIs that are developed within the AN project. To make it easier when composing, the UCIs are divided into two parts, the domain part and the path part:

- ctx://domain.org

  This part represents the domain part of the UCI. This should be equal to the name of the Ambient Network.

– /path

This part represents the path part of the UCI. This should represent logically where in the network the UCI points to (i.e. the physical location is not revealed).

Dividing each UCI into two parts are helpful when composing, as the path part is the same even though the domain is changed into a new domain (representing the composed AN). This enables a more efficient search routine as a node can quickly find out in which DHT overlay the search should be executed. Re-registration could potentially be avoided, as a new domain name replaces the existing domain name before the UCI is divided into two parts.

5 Distributed Context eXchange Protocol

This chapter presents the Distributed Context eXchange Protocol (DCXP) that constitutes a ContextWare. It studies how a DHT overlay is managed and how a CUA uses to manage a DHT overlay. The chapter also gives details about the relation between different node entities, transport protocol, and message format. This in turn is followed by descriptions of how the overlay operates in different situations and how an end node may use the protocol.

5.1 Transport protocol

ContextWare employs the Distributed Context eXchange Protocol (DCXP) to resolve, store, and register Context User Agents (CUAs). Messages are carried by a DHT overlay which is based on top of UDP. Messages in ContextWare are mostly a single notify message or subscribe message. There are no longer sequences of messages which might justify setting up a TCP connection. The main advantage of using UDP over TCP is to minimize the overhead of TCP session. TCP creates a session for each connection and maintains the session until the end-points disconnect. In a network with wireless entities use of TCP would be disadvantageous. That is if an entity disappears, TCP would try to retransmit packets that are not acknowledged until timeout expired. TCP also causes significantly reduced transmission rates due to adverse interaction between latency in the radio interface and the TCP transmission window that causes TCP to believe that packets are lost.

Therefore a connectionless protocol is used, UDP. UDP provides the minimal service needed, while TCP would provide a lot of unnecessary features which will not be used and would burden the entities and network with more work and messages. However, the protocol has to include a mechanism to acknowledge messages and retransmit lost messages. Therefore, DCXP is designed as a reliable protocol on top of a DHT overlay on top of a connectionless protocol, UDP.

5.1.1 Message format

The format of the data in messages is XML [39] which is popular and provides a means of describing data which both computers and humans can interpret. XML gives the developer an easy way of structuring the data, especially when the structure is complex. A parser can easily interpret the information in a XML message to make use of the information on a computer. XML and existing parsers enable a developer to rapidly create and interpret correctly formatted documents.

In addition, Vidal [4] used XML in his protocol and prototype which is used as a template when developing and implementing the prototype for my thesis.

5.1.2 Node entities

In a DHT overlay as described here which is based on Chord [24] all participating nodes are logically placed in a ring structure, depending on the node ID which is a calculated hash of a node's IP address. In order for a node not to have full knowledge of all nodes in the overlay

some helpful lists are introduced. These lists must be updated periodically as nodes join and leave the overlay. This is explained further in the following sections. Below is a node's entities described:

- Successor list
  A list which contains a number of a CUA's immediate successor(s). This list helps a CUA when immediate nodes leave as then can a CUA establish contact with the next CUA on the list.
- Predecessor list
  A list which contains a number of a CUA's immediate predecessor(s). This list works well even if it contains only one node, this is because each node only check if the successor list is updated with working CUAs.
- Finger table
  This list is not crucial in order for the overlay to work, but it enhances routing of messages. Finger tables are described more specifically in section 3.3.4.3.
- Registered UCIs
  A database of all the UCIs that are registered at this node. Each element contains UCI, contact information and for how long each registration lasts.

5.2 Management of an overlay

When a new node tries to join a DHT overlay it sends a discover message. If the node does not get reply within a predefined time, then it is assumed to be the first node in creates a new DHT overlay. In order to avoid problems with long latency times and lost packets a node which has not received a response to a previously sent discover message should try again for a number of times. The second node that joins should receive a discover message and then be able to join the existing DHT overlay.

If the node receives a reply, then it joins this existing overlay, by sending a request to join to the responding node. The responding node grants, redirects, or refuses the joining node based upon its preferences and policies. If a node is refused to join an overlay it is not able to register and resolve any context information.

The discover response message contains information about which DHT overlay it represents (i.e. the name of the ambient network). If multiple responses are received from the same network the requesting node should send a join message to one of these nodes. If the multiple responses are from different networks, then the requesting node joins the network which equals the name of the ambient network the node intends to join. The discover message is also used in order to find nodes of other networks when a composition is going to take place.

5.2.1 Transferring registrations

A node is responsible for a part of the ID space, the range between its own ID down to the ID before the predecessor, i.e. in figure 12, below, node with ID 1 is responsible for the range from 5 to 1. If a new node joins, then the ID space must be divided between the joining node and admitting node. The admitting node is the node which is responsible for the ID space range which corresponds to the ID of the joining node. Therefore all the registrations that lie within the joining node's ID space must be transferred to the joining node. If a new node joins with ID 7 in the figure below, then the node with ID 1 has to transfer the registration of key with ID 6 to the joining node.

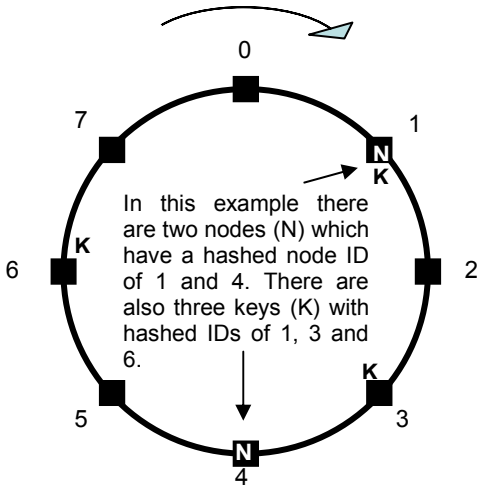


Figure 12: Picture how ID space is divided in a Chord [24] ring

5.2.2 Departing Nodes

If a node knows that it will leave the network, then it could leave gracefully by sending un-register messages to its successor. This helps them to quickly adapt to the new environment; as they may update their finger tables together with their successor and predecessor lists. The other scenario is when a node leaves without telling its neighbors. The DHT overlay still works but needs to fix its tables at the next periodic update.

5.2.3 Periodic Update

A node can fail, thus leaving the DHT overlay without telling other nodes. In order to detect this; there must be periodic updates of finger tables, the successor list, and the predecessor list. This update (stabilization) is needed in order to maintain the DHT overlay, but it generates overhead due to this protocol. The number of additional messages depends upon the number of nodes that have failed since the last update procedure.

A large number of nodes joining and leaving a network generates overhead traffic, as nodes must update their lists more often than if the overlay is stable. Therefore it is important that all nodes leaving the DHT overlay send a message to its surrounding nodes that it is leaving. This help the remaining nodes as they then get updates to their successor list, predecessor list and finger table when a node leaves and can instantly update its surrounding nodes.

5.3 DCXP management of a DHT overlay

This section describes how the Distributed Context Exchange Protocol (DCXP) manages a DHT overlay and provides a means for sources, sinks, and managers to exchange context information. This protocol is inspired by a draft of a Peer-To-Peer Session Initiation Protocol (P2P SIP) [29] and by the Context Exchange Protocol (CXP) [4].

5.3.1 Registering in a DHT overlay

Before a Context User Agent (CUA) can accept registrations and resolve Universal Context Identifiers (UCIs) it must find the DHT overlay in the network and register. This is typically done as soon as a device powers on and registers with the network.

A typical registration process looks like the following; see in Figure 13. In this scenario there exists an overlay which nodes 1, 4, and 6 participates in, node 2 is the joining node which starts the discover procedure.

- Discover of overlay

  A discover procedure starts when the new CUA broadcasts a DISCOVER message. The CUA receives responses from other CUAs consisting of a DISCOVER_RESPONSE message which contains information about the name of the overlay, hashing algorithm, DHT algorithm, and contact information. Multiple DISCOVER_RESPONSE messages may be received. The joining node sends its REGISTER message to the node which represents the overlay that the new node wants to join.

  NOTE: The name of the overlay should be equal to the ambient network name, such that the name of the overlay may be retrieved from other Functional Entities when AN composition takes place. The Management of Network Composition Functional Entity is responsible for posting a composition initiative to the ContextWare when a composition takes place.

- Registration in overlay

  The local CUA try to registers with the DHT overlay by sending a register message to a node within the overlay (i.e. a node which has responded to the DISCOVER message). This node might redirect the joining node to another node based upon the DHT algorithm (i.e. how the ID space is defined and shared among the nodes). Figure 13 shows a scenario where the first node contacted is also the correct node. If node 2 had contacted one of the other nodes in the overlay it would have been redirected to try to register with another node. The REGISTER message tells which overlay the joining node should join together with its calculated nodeID.

- Registration response

  The contacted node replies with a REGISTER_RESPONSE confirming the registration or redirecting the node. If it is a successful registration the joining node also receivee information about its neighbors (i.e. predecessor and successor lists).

- Updating the overlay

  When a joining node receives a REGISTER_RESPONSE message confirming the registration it has to tell its new neighbor about its existence. This is done by sending an UPDATE message to its new

predecessor. Periodically an UPDATE_REQUEST is sent by a node to its successor(s) to check if they are alive and also get an update of this successor's view.

The registration process is completed after the UPDATE message is sent to the predecessor. The periodic UPDATE_REQUEST messages are sent while a node is in an overlay. After a node has completed a registration in a DHT overlay it is able to register its context identifier (UCI), store information concerning other nodes, assist when new nodes join, or assist in lookups. Figure 13 shows the complete procedure of a node discovering an overlay and registering within it.

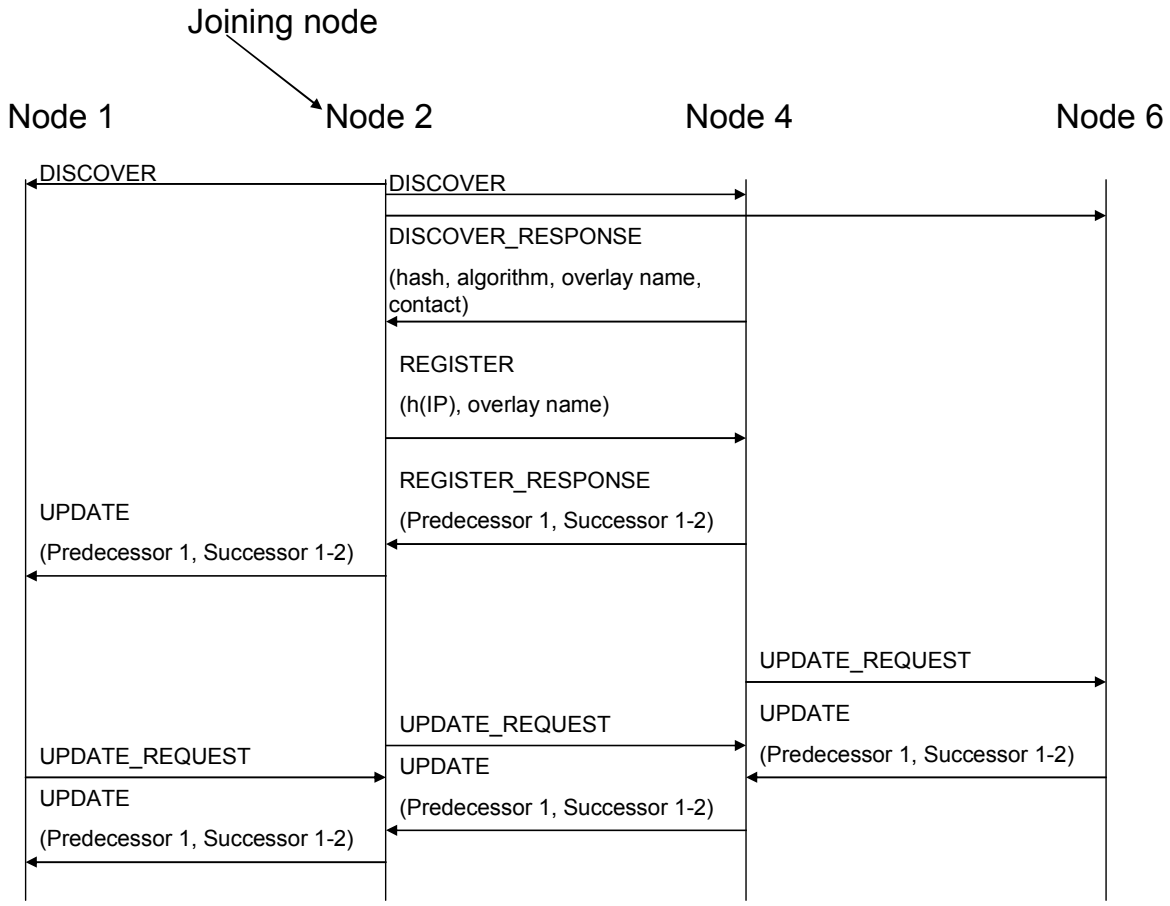


Figure 13: A CUA registering in a DHT overlay

5.3.2 Registering the existence of a context source via the DHT overlay

Once a node is registered in the DHT overlay, then the local context source is able to register its context UCIs within the overlay via a local CUA. The local CUA receives a register request from a source, then it registers the UCI with the source's data information at the corresponding node in the DHT overlay (i.e. this message is sent on behalf of a request from an application or service). This process is shown below in figure 14.

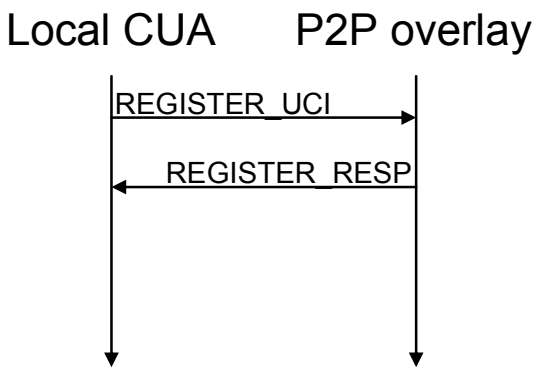


Figure 14: A CUA registering a UCI

The node to which the REGISTER_UCI message is sent is the node which is responsible for the ID space corresponding to the ID of the UCI (i.e. the ID is a hash of the UCI). If the first contacted node is not the correct node within the ID space, then the registering node is redirected to another more appropriate node. The local CUA in addition to registering the UCI within the corresponding node also concatenates an integer (starting from one (1)) to the end of the UCI and calculates a new hash in order to register a replica to avoid losing information if the first node fails.

5.3.3 Resolving a UCI in DHT overlay

The DHT overlay behaves similar to a lookup service for the different resources. In this case the DHT overlay stores UCIs and information corresponding to them (specifically the address of the node which registered this UCI). When an application or network service initiates a lookup of a UCI, the local CUA sends a message to resolve the UCI, see figure 15. This message is routed as described above using a DHT algorithm. If a UCI can not be found, then the local CUA tries to identify one of the replica nodes which does have this information (the manner of generating the names of this replica was described in the previous section) and resolve this new UCI. As all UCIs registered in the overlay are stored with at least two replicas this should be tolerant to at least a single DHT node failure. A local CUA can also verify the resolved UCI by resolving the UCI to one of the other registered replicas and comparing the data, this prevents a single node from tampering with the registered data.

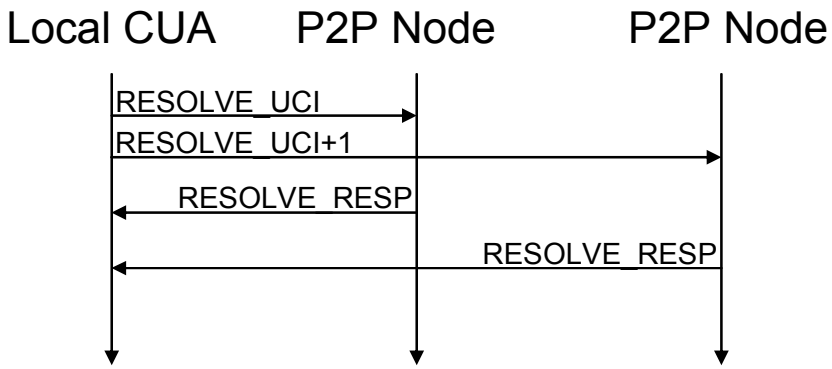


Figure 15: A CUA resolving a UCI together with a replica

5.4 DCXP signaling for end nodes

This section describes the signaling an end node must perform in order to register, resolve, and fetch context information with a Distributed Context eXchange Protocol (DCXP) enabled CUA, the basic operation of which is presented in subsection 5.3.

A context user agent acts as middleware between context clients and sources with the help of a DHT overlay in order to both register and resolve UCIs. A DHT overlay helps a CUA to distribute the essential functional entities (specifically Context Coordinator, Context Manager, and Context Information Base) within a P2P network.

5.4.1 Registering context information

When a source wants to publish context information it registers its context information in the DHT overlay, see figure 16. This is done by sending a REGISTER message containing the UCIs of the context information it wants to register to the local CUA. Then the local CUA registers the UCI within the DHT overlay as described in section 5.3.2. If the source wants to update its context it sends a new register message. Note that a register message must also be updated periodically to prevent it from being aged out.

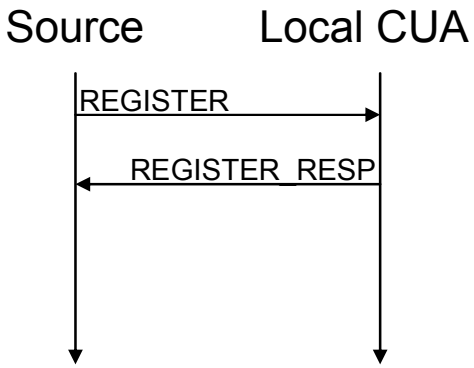


Figure 16: A source registers its UCIs

Below are two different scenarios of handling context information presented:

- Registering context without delegating

  This is the typical scenario: a source registers its UCI and then on its own handles all the fetch requests from consumers.

- Registering context when delegating

  A source can delegate context handling to a context manager. This might happen if the context is of interest for many nodes or if the source has limited processing power or bandwidth. The scenario is similar to that above, but instead of simply registering the UCI, it also has to send the context information to the delegate. The context information is then stored within the DHT overlay together with the UCI.

5.4.2 Resolving a context information request

A consumer of context information knows which UCI it wants to fetch, but not where it is located. This is solved by sending a RESOLVE message to the local CUA which does the lookup in the DHT overlay on behalf of the requesting context client. The context client receives either the contact information of where the context is stored or the full context information depending on whether the source has delegated the context handling or not. This scenario is presented in figure 17 below.

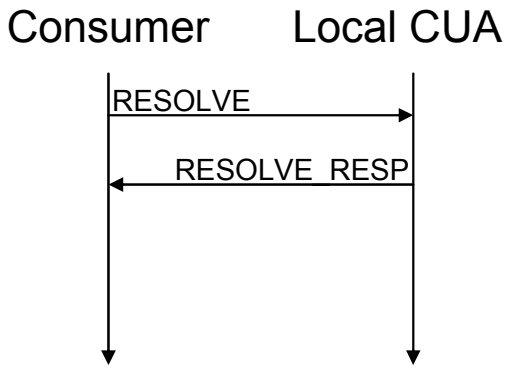


Figure 17: A context client resolves a UCI

5.4.3 Get/Subscribe to specific context information

When a CUA has received the contact information from a lookup operation the CUA returns this information to the requester which can fetch the context information directly from the handler (i.e. a source or context manager depending upon whether the source has delegated the task or not). The requested information might be requested only once or with help of a subscription method enable notifications of updated versions of the context information to be automatically sent for as long as the subscription is valid. The fetch message must contain information about which UCI or UCIs the consumer is interested in.

In the first scenario, figure 18, a consumer only wants to retrieve the context once. Thus a get message is send and the sources replies with a notify message containing the requested context.

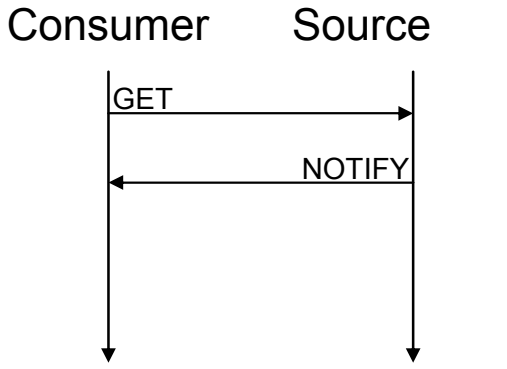


Figure 18: A context client retrieves context information once

The other scenario is used when a consumer wants to retrieve the context information when the information is updated. Figure 19 shows the case of a subscribe message being sent to the source and the source replying with notify messages whenever the context information changes. This solution minimizes the traffic as the requesting node is not periodically polling

the source for updates, instead the source notifies the subscribing consumer when context information is updated.

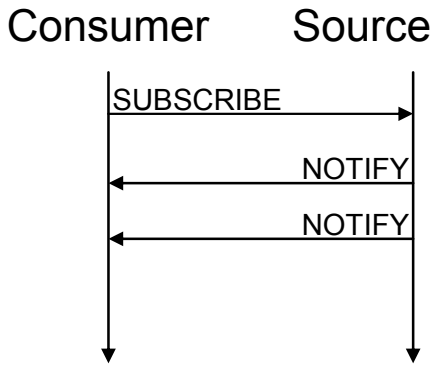


Figure 19: A context client subscribe to context information

To unsubscribe to a subscription the consumer only has to send a new subscribe message to the source with an expiration timer set to zero. If the expiration timer is set to another value the consumer must issue a new subscribe message before the time expires, otherwise the subscription ends.

5.5 Composition

Composition is one of the key features ambient networks provide. A composition of two networks provides the entities services and resources of both networks, if they decide to compose. The decision to compose or not compose is based upon the policies the network owners have established. Two networks can compose by many different reasons and the time spent composed can vary from minutes and seconds to years and in the extreme case, forever. Composition is handled by the WP-G working group within the Ambient Networks project. They have specified a functional entity which provides composition management. This enables other entities the ability to subscribe to context information derived following composition.

When two networks are close and able to discover each other, they can decide to compose. The decision to compose is handled by the Management of Network Composition Functional Entity within each of the two networks' ACS. The ContextWare, specifically the Context Coordinator, receives a request from the Composition Management Functional Entity to compose the two ACSs with unconnected ContextWare [9]. The Context Coordinator (i.e. one of the nodes within the overlay as all nodes together represent a CC) sends out discover messages containing the name of the overlay it represents and then thereafter try to negotiate and compose the two unconnected spaces of ContextWare.

In prior research [4] a central server approach was used together with context managers acting as gateways between two ambient networks. Context clients used context managers to request context information from other ambient networks, in the same way as they would have requested it from within their local network. A gateway approach is also used to solve the composition problem within a DHT overlay. The other method which is used is absorption or full compose; this method demands the creation of a common namespace in a DHT that is spread over nodes in both of the two composing networks. These two methods are further explained below.

A problem when composing two different ambient networks with two different ContextWares could be that they use different DHT algorithms, hashing algorithms, etc. Therefore when these networks compose they must negotiate a common set of settings.

5.5.1 Gateway (de-)composition

A gateway acts as middleware between the two DHT overlays. This approach is used when two ambient networks only partially compose and little information needs to be exchanged: for example, when two Personal Area Networks are composing (as it is likely that they decompose back to the two original networks). Using a gateway would create a single-point-of failure and the load on the gateway could be high. However this is avoided by using a distributed network, as shown below. A gateway solution minimizes the number of messages that must be exchanged when two networks (de-)compose. All the registrations are still in their original place, as there was only routing of messages between the nodes within the two domains that formed the combined DHT. The extra messages needed when a request is made are the number of messages needed to find a node which is part of the requested domain. Initially there are some additional messages, as the newly composed overlay forms and messages are to be exchanged in order to populate the successor lists and predecessor lists. In the start up phase when the combined DHT only consists of two nodes the successor and predecessor simply are the other and visa versa.

Figure 20 shows the concept of a gateway solution where two ambient networks have composed into a third ambient network. In this scenario the two networks have different DHT algorithms so the nodes inside the composed DHT must be able to handle two different kinds of algorithms and be part of two overlays as well. A node participating in the combined overlay acts as a gateway node for nodes in the opposite network. This is because if a node from A would have acted as a gateway node for the nodes in A it must also join B in order to be able to do lookups in that overlay. This should be avoided as it complicates when the two networks decompose, thus this node must then unregister when the two networks decompose and it causes unnecessary traffic in B's overlay.

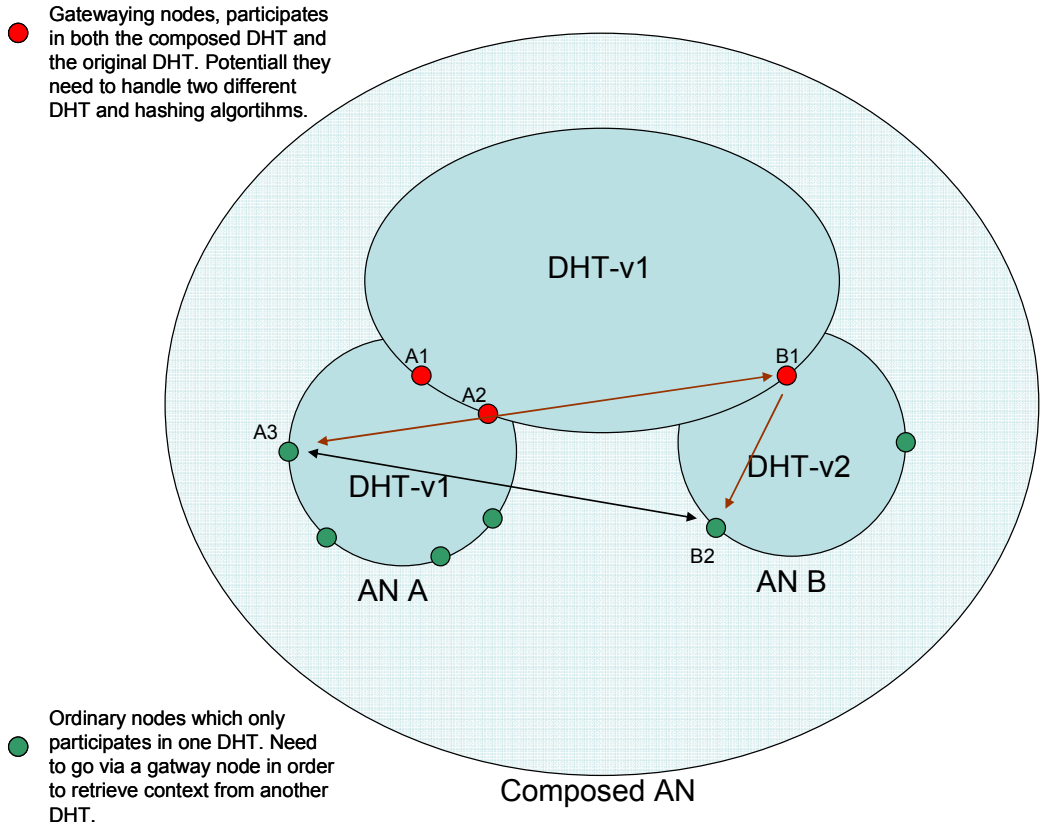


Figure 20: Abstract gateway scenario

When gatewaying, a first contact is established between two nodes within each of the two networks. In the scenario below two networks, A and B, want to compose. This is done with a DISCOVER message and a DISCOVER_RESPONSE message. Later they exchange information about each DHT overlay (such as algorithm and hashing method used) and they agree to compose.

- Node B1 receives a DISCOVER message and replies with its capabilities (i.e. sends a DISCOVER_RESPONSE message). Node A2 sends a COMP_REQ in order to attempt to fulfill B1's request. If B1 can accept A2's proposal it confirms the set up with a COMP_REQ_RESPONSE message saying OK.
- After the two initial nodes have agreed to compose they register the opposite domain as a UCI in their DHT (i.e. node A2 register a UCI in network A containing B's domain name and visa versa).
- Node A2 and B1 now create a new common DHT overlay with a joint name together with N nodes which have the best performance (or some other desirable characteristics as decided by network owners) in each of the two networks (i.e. a COMP_NOTIFY is sent to the selected nodes). This information could be static or retrieved with help of context information. This new overlay only acts as a distributed gateway between the two networks; the overlay could vary in size from only one node (which creates a single node of failure) from each domain up to a large number of nodes. In this scenario a COMP_NOTIFY is sent to node A1 and B2.
- When new nodes join the combined overlay they have to update the registered UCI with the new nodes.

Below is the above scenario illustrated with sequence diagrams:

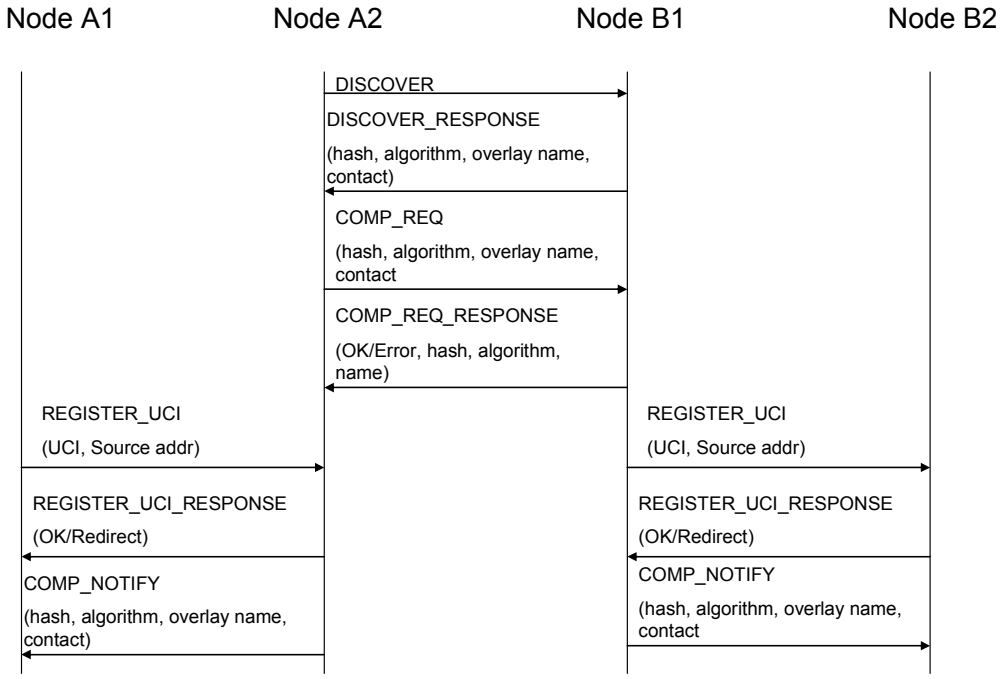


Figure 21: Gateway composition scenario

- If a node within AN A needs something from AN B it contacts one of the nodes on A's side which can redirect the query to a node in AN B. The node in AN B can do the lookup and then responds with the address to the source. This is illustrated in Figure 22 with node A2 which first resolves the UCI of B's domain name in its own overlay. Then send a resolve request to B1 which responds with the address of the source representing the UCI. Finally A2 fetches the context information at the source.
- The combined overlay only handles keys with UCIs of the composed network. All the original keys are still stored in the original two DHT overlays.

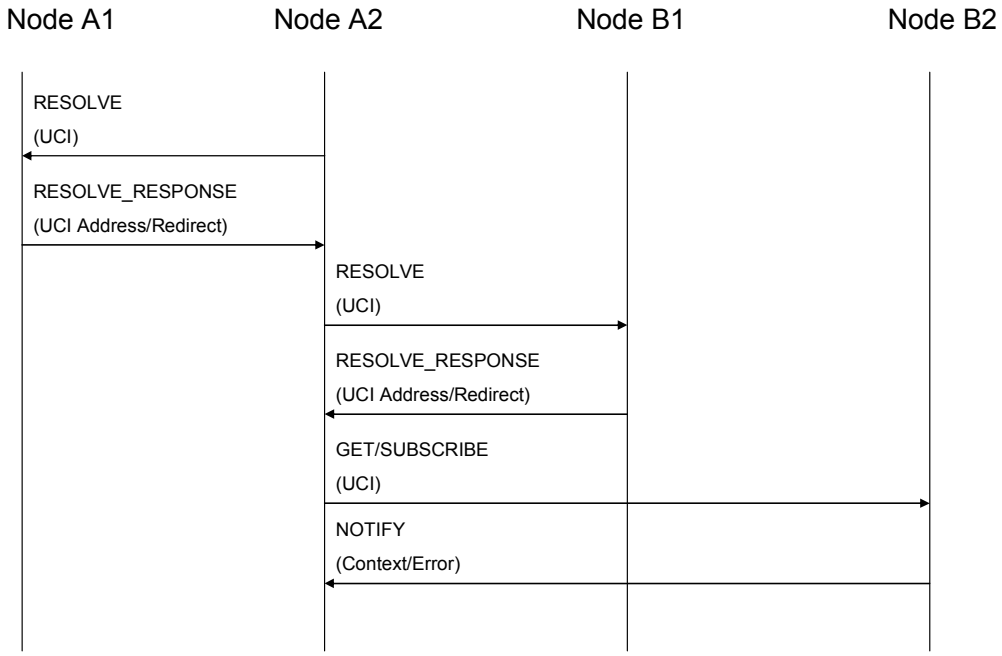


Figure 22: Resolve scenario of a UCI via gateway composition

When a node resolves a domain name for a network the reply could contain a list of potential gateways. The requesting node has to select one of the nodes by random before it sends the resolve request to one of the gateway nodes in the other network. In order for a node to make a faster lookup it should store the list of gateway nodes, the list could then be used next time a cross-network lookup is executed. The list of gateway nodes must then be updated periodically.

In order for a smooth decomposition the original DHTs still must be managed even though the two networks are composed. When then they decompose the joint DHT ceases to exist and the two original DHTs should work without knowledge of each other (see figure 23).

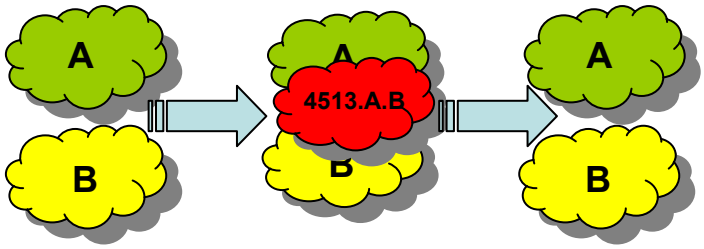


Figure 23: Gateway (de-)composition of two ContextWares

5.5.2 Absorption (de-)composition

In order for a full compose, or absorption, there is need for another more efficient method than gateway composition. This method is used when handling a large number of nodes and large amounts of context information. In those cases the utilization of a gateway would be counter to the advantages of a P2P network. The initial message exchange is the same as in gateway scenario; thus the discover procedure and negotiation are performed in the same manner. Below is a scenario describing a full composition scenario, **after** the initial discover and negotiation procedure.

- Two networks, A and B, want to compose. In this case the networks are large and the network owners have the ambition to merge these networks for a long time period. This is established with help of policies set by the network owners.
- After initial discover and negotiation the ContextWares have agreed on the preferences for the common overlay (i.e. the hash algorithm, DHT algorithm, etc). Then is the overlay set up as described in the gateway scenario above. Initially, it is set up only between the two nodes of first contact, but then the overlay grows as more and more of the selected nodes join.
- The new overlay initially only functions as a distributed gateway between the two networks. If a node within AN A needs something from AN B it contacts one of the nodes on A's side which redirects the query to a node in AN B.
- One of the two overlays which compose is selected as target overlay, thus all nodes from the other network has to join this. Sequentially nodes from the overlay **not** selected as target overlay joins. Once a node has joined the target overlay, then all new registrations and re-registrations are made in this new overlay as well.
- All nodes can not join target overlay immediately as that creates an enormous amount of extra traffic and pressure on the target overlay. Therefore some select algorithm is required. This algorithm has to in the beginning select and let a limited number of nodes join the selected overlay. While the total number of nodes in the selected overlay grew a larger number of new nodes can join since the traffic caused by they joining nodes are more and more spread out.

- The two overlays work meanwhile all new nodes join, context information can be reached from the two networks with help of gateway function. Although when a node has joined the selected overlay it has to unregister in the old overlay and all new registrations are made in the new overlay.

Decomposition of two networks which are fully composed could happen in two ways. The two networks decompose and take the same form and size as prior to their composition, or two completely new networks could be created. In the light of a full composition, the two original domains do not exist any longer, therefore two completely new domains have to be created.

5.6 DCXP Messages managing DHT overlay

The message structure is inspired by earlier work performed by Vidal [4]. A modification of the messages are required according to recent work done in the Ambient Networks project together with the distributed approach of this thesis compared to Vidal's central-server approach.

Each message contains the same fields apart from a specific body which contains message specific information. The following fields are included in every message:

- Method
  This specifies the method of this message. It is represented by a string value containing SUBSCRIBE, REGISTER, etc.
- Sequence number
  A sequence number in order to check if the messages received are in correct order.
- Acknowledge
  This field contains the sequence number of the message which is acknowledged. This is an optional field.
- FromNodeID
  This contains the nodeID of the sender of the message.
- From
  This contains the IP-address of the sender of the message.
- To
  This contains the IP-address of the receiver of the message.
- Body
  Specific information depending which type of message it is. This section is further explained in the text below specifically for each message.

5.6.1 DISCOVER

This message is sent from a node that wants to participate in a P2P overlay or to find another overlay to compose with. The <FromNodeID> and <To> field are empty as a hashing algorithm is not decided together with that the message is intended for anyone, not a specific node.

- **Overlay** – This field is optional. It is the name of the overlay this node represents. If a node is not registered with any overlay this field is left out.

5.6.2 DISCOVER_RESPONSE

A response sent from a node within a DHT overlay to a node outside this overlay contains the address information of this node. The following fields are included in the body:

- **Address** – The address to which a REGISTER message or COMP_REQ message can be sent. This element contains an attribute to specify what type of address is used.
- **Network_info** – Contains specific information about this overlay. For now this only contains three fields:
    - **Name** – The name of the DHT overlay.
    - **Hash** – The hashing algorithm used, ex. SHA-1 or MD5.
    - **Algorithm** – The DHT algorithm used, ex. Chord.

5.6.3 REGISTER

This message is sent from a node that wants to be a part of a DHT overlay. Usually it is sent immediately after a DISCOVER_RESPONSE message is received. It is sent to the node that sent the DISCOVER_RESPONSE message.

- **NodeID** – The nodeID of the sender of the message.
- **Address** – The address of the sender. This element contains an attribute to specify what type of address is used.
- **Name** – The name of the overlay the node wants to join. This should be the name specified in the received DISCOVERY_RESPONSE message.

5.6.4 REGISTER_RESPONSE

This message contains information indicating about if the registration was sent to the DHT overlay responsible for the correct area of the ID space within the P2P overlay. If it was not, then the node sends a reference to the closest node to the correct node which it is aware off.

- **Success** – This field specifies “True” if the registration was accepted. If the node request is redirected the field will be “Redirect”. If the node can not join, then the field will be “False”.
- **NodeID** – The nodeID of the sender if the field <Success> contains “True” or “False”. If the field <Success> contains “Redirect”, then this is the nodeID of the node to which the joining node is redirected to.
- **Address** – The address of the sender if the field <Success> contains “True” or “False”. If the field <Success> contains “Redirect”, then this is the address of the node to which the joining node is redirected to.
- **SuccessorList** – This list contains a number of a node’s successor nodes. This list is currently limited to 2 nodes.
    - **S0**
        - NodeID – The nodeID of the first successor.
        - IPAddress – The address of the first successor.
    - **S1**
        - NodeID – The nodeID of the second successor.
        - IPAddress – The address of the second successor.
- **PredecessorList** – This list contains a number of a node’s predecessor nodes. This list is currently limited to its immediate predecessor node.
    - **P0**
        - NodeID – The nodeID of the first predecessor.
        - IPAddress – The address of the first predecessor.

- **ContextAvailable** – This list contains the UCIs a node wants to transfer to the new node in the overlay. This is an optional field.
    - **UCI –** The UCI.
        - Domain – The hashed domain-part of the UCI.
        - Path – The hashed path-part of the UCI.
        - IPAddress – The address of where the context the UCI is representing can be found.
        - Exp – Expiration timer, for how long in seconds is a registration valid.

5.6.5 UNREGISTER

This message is sent from a node to its immediate successor when it wants to leave the overlay gracefully. This helps the overlay to fix the configuration lists and also able to transfer UCI registered in the overlay to another node.

- **ContextAvailable** – This is list contains the UCIs a node wants to transfer to another node in the overlay. This is an optional field.
    - **UCI –** The UCI.
        - Domain – The hashed domain-part of the UCI.
        - Path – The hashed path-part of the UCI.
        - IPAddress – The address of where the context the UCI is representing can be found.
        - Exp – Expiration timer, for how long in seconds is a registration valid.
- **PredecessorList** – This is list contains a number of a node's predecessor nodes. This list is currently limited to its immediate predecessor node.
    - **P0**
        - NodeID – The nodeID of the first predecessor.
        - IPAddress – The address of the first predecessor.

5.6.6 UPDATE

This message contains information about a node's predecessor and successor. This message is always sent to a nodes closest predecessor. This message is either sent in response to a received UPDATE_REQUEST message or to notify its predecessor about its existence when it has recently joined the overlay.

- **NodeID** – The nodeID of the sender of the message.
- **Address** – The address of the sender. This element contains an attribute to specify what type of address is used.
- **SuccessorList** – This is list contains a number of a node's successor nodes. This list is for now limited to a size of 2 nodes.
    - **S0**
        - NodeID – The nodeID of the first successor.
        - IPAddress – The address of the first successor.
    - **S1**
        - NodeID – The nodeID of the second successor.
        - IPAddress – The address of the second successor.
- **PredecessorList** – This is list contains a number of a node's predecessor nodes. This list is for now limited to its immediate predecessor node.
    - **P0**
        - NodeID – The nodeID of the first predecessor.
        - IPAddress – The address of the first predecessor.

5.6.7 UPDATE_REQUEST

This message is sent periodically to a node's immediate successor in order to check if the successor is still alive and to get an update if successors move away.

- **NodeID** – The nodeID of the sender of the message.
- **Address** – The address of the sender. This element contains an attribute to specify what type of address is used.

5.6.8 ACKNOWLEDGEMENT

This message is sent when a packet should be acknowledged and no other packet is to be sent back to the sender. The message must contain the <ack> field, but the body is empty.

5.6.9 REGISTER_UCI

This message is sent from a CUA registering one (or more) UCIs with another node in the overlay.

- **ContextAvailable** – This is list contains the UCIs a node wants to register in the overlay.
    - **UCI –** The UCI.
        - Domain – The hashed domain-part of the UCI.
        - Path – The hashed path-part of the UCI.
        - IPAddress – The address of where the context the UCI is representing can be found.
        - Exp – Expiration timer, for how long in seconds is a registration valid.

5.6.10 REGISTER_UCI_RESPONSE

This message is sent as a reply to a REGISTER_UCI message. It either confirms a successful registration or redirects the registering CUA to another node, which is closer to the node responsible for the ID space of the hashed UCI.

- **Success** – This field specifies "True" if the registration was accepted. If the node request was redirected or any other error the field is "False".
- **NodeID** – The nodeID of the sender if the field <Success> contains "True". If the field <Success> contains "False", then this is the nodeID of the node to which the joining node is redirected to.
- **Address** – The address of the sender if the field <Success> contains "True". If the field <Success> contains "False", then this is the address of the node to which the joining node is redirected to.

5.6.11 RESOLVE_UCI

This message is sent from a CUA wanting to resolve a UCI into an IP address.

- **Uci** – This field specifies the UCI that a node wants to resolve.

5.6.12 RESOLVE_UCI_RESPONSE

This message is sent as a reply to a RESOLVE message. It either confirms a successful resolve or redirects the resolving CUA to another node which is closer to the node corresponding to the ID space of the hashed UCI.

- **Success** – This field specifies "True" if the registration was accepted. If the node request was redirected or any other error the field is "False".
- **Hash** – The hash of the requested UCI.

- **Address** – The address of where the context information represented by the UCI, if the field <Success> contains "True". If the field <Success> contains "False", then this is the address of the node to which the requesting node is redirected to.

5.6.13 COMP_REQ

This message is sent after a node has received a DISCOVER_RESPONSE message from another network.

- **Address** – The address to the gateway node in the overlay this node represents. **Network_info** – Contains specific information about this overlay. For now this only contains three fields:
  - **Name** – The name of the DHT overlay.
  - **Hash** – The hashing algorithm used, ex. SHA-1 or MD5.
  - **Algorithm** – The DHT algorithm used, ex. Chord or DCXP.

5.6.14 COMP_REQ_RESPONSE

This message is sent as a confirmation message back to the network which initiated the discover procedure.

- **Success** – This tells if the composition were successful or not.
- **Network_info** – Contains specific information about this overlay. For now this only contains three fields:
  - **Name** – The name of the DHT overlay.
  - **Hash** – The hashing algorithm used, ex. SHA-1 or MD5.
  - **Algorithm** – The DHT algorithm used, ex. Chord or DCXP.

5.6.15 COMP_NOTIFY

This message is sent from each of the nodes which have established contact between the two networks to the selected nodes (in each network). The nodes which receive this message should send a REGISTER message to the node indicated in this message.

- **Contact** – Information about which node a REGISTER message should be sent to.
- **Subscription** – Contains specific information about this overlay. For now this only contains three fields:
  - **Name** – The name of the DHT overlay.
  - **Hash** – The hashing algorithm used, ex. SHA-1 or MD5.
  - **Algorithm** – The DHT algorithm used, ex. Chord or DCXP.

5.7 DCXP messages end node signaling

The messages used by an end node to communicate in order to collect, manage and disseminate context information could be implemented in different ways. Either as regular messages sent from the end node to the local CUA which then manage the DHT overlay or as an API if the CUA is implemented into an application, the latter case then also has to implement the subscribe, get and notify messages into the CUA.

The messages are developed from the same ground as described in the previous section. Each message contains the same fields apart from a specific body which contains message specific information. The following fields are included in every message:

- Method

  This specifies the method of this message. It is represented by a string value containing SUBSCRIBE, GET, etc.

- Sequence number
  A sequence number in order to check if the messages received are in correct order.
- Acknowledge
  This field contains the sequence number of the message which is acknowledged. This is an optional field.
- FromNodeID
  This contains the nodeID of the sender of the message.
- From
  This contains the IP-address of the sender of the message.
- To
  This contains the IP-address of the receiver of the message.
- Body
  Specific information depending which type of message it is. This section is further explained in the text below specifically for each message.

5.7.1 REGISTER

This message is sent from a source or manager to a local CUA in order to register the existence of context information together with UCI.

- **Uci** – The name of the requested UCI.
- **Expires** – Expiration timer, for how long in seconds is a registration valid.
- **Value** – The value of the context information represented by the UCI.

5.7.2 RESOLVE

This message is sent to a local CUA from a manager or sink in order to get contact information of the requested UCI.

- **Uci** – The name of the requested UCI.

5.7.3 SUBSCRIBE

This message is sent from a requesting node to a source after it has received information about where the Context Information Object is located. This message forces the source to periodically send updates of the requested context information until the message has expired.

- **Subscription** – Contains specific information about this overlay. For now this only contains two fields:
  - **Uci** – The name of the requested UCI.
  - **Expires** – Expiration timer, for how long in seconds is a registration valid.

5.7.4 GET

This message is sent from a requesting node to a source after it has received information about where the Context Information Object is located. This message forces the source to send one immediate update of the requested context information.

- **Subscription** – Contains specific information about this overlay. For now this only contains one field:
  - **Uci** – The name of the requested UCI.

5.7.5 NOTIFY

This message is sent from the source after it has received a SUBSCRIBE or GET message. If a SUBSCRIBE message is received the source periodically sends NOTIFY messages when the Context Information Object is updated. This message contains the requested value of the Context Information Object.

- **Context element** – Contains specific information about this context element. For now this only contains three fields:
  - **Uci** – The name of the requested UCI.
  - **Seq** – The sequence number of the requested context information. Each time a context information is updated this number is increased by one.
  - **Value** – The value of the context information represented by the UCI.

6 Prototype implementation

This chapter describes the implementation phase, how and which applications used, problems etc. It focuses on how the prototype in overall works and in which environment. It also describes how the Distributed Context Exchange Protocol were implemented and adapted to match this specific implementation.

6.1 Environment

6.1.1 Hardware

The hardware used to run the implemented prototype was handheld devices in the iPAQ series by HP. Two different models were used which used the same operating system *Windows Mobile 2003 for Pocket PC Premium Edition* powered by Windows CE 4.2. The models are:

- HP iPAQ HX4700 [40]
- HP iPAQ h5500 [41]

The main difference between the two devices are the processors where they both use Intel processors but with different speeds. Both of the models are equipped with an 802.11b network card which supports WEP and WPA encryption.

6.1.2 Software

The prototype was implemented in Java Micro Edition (Java ME). In order to be able to run a Java application on the handheld devices a Java Virtual Machine (JVM) was installed called J9 by IBM. In this case the JVM is based on the CDC 1.0 and Personal Profile 1.0 which is only a part of the original full Java SE. A comparison chart can be found at: *http://java.sun.com/products/cdc/reference/cdc_packages.pdf*.

The integrated development environment used was IBM WebSphere Studio Device Developer (WSDD) [42]. The WSDD is used for developing applications on handheld devices using Java ME and includes J9 and provides good support for the Windows Mobile platform.

6.1.3 Network

An exclusive network is set up for the prototype, this is a wireless LAN which uses 802.11b with WPA encryption technique. The handheld devices connected to this network were all in the same subnet which let them communicate with each other directly using IP.

Picture 24 shows the logical presentation of the prototype when it was divided into two different ambient networks. Each node (PDA) is represented by a CUA which is a member of the distributed overlay that makes up the function of Context Coordinator. The prototype was sometimes also used as one ambient networks, where all five nodes were in the made up the same overlay.

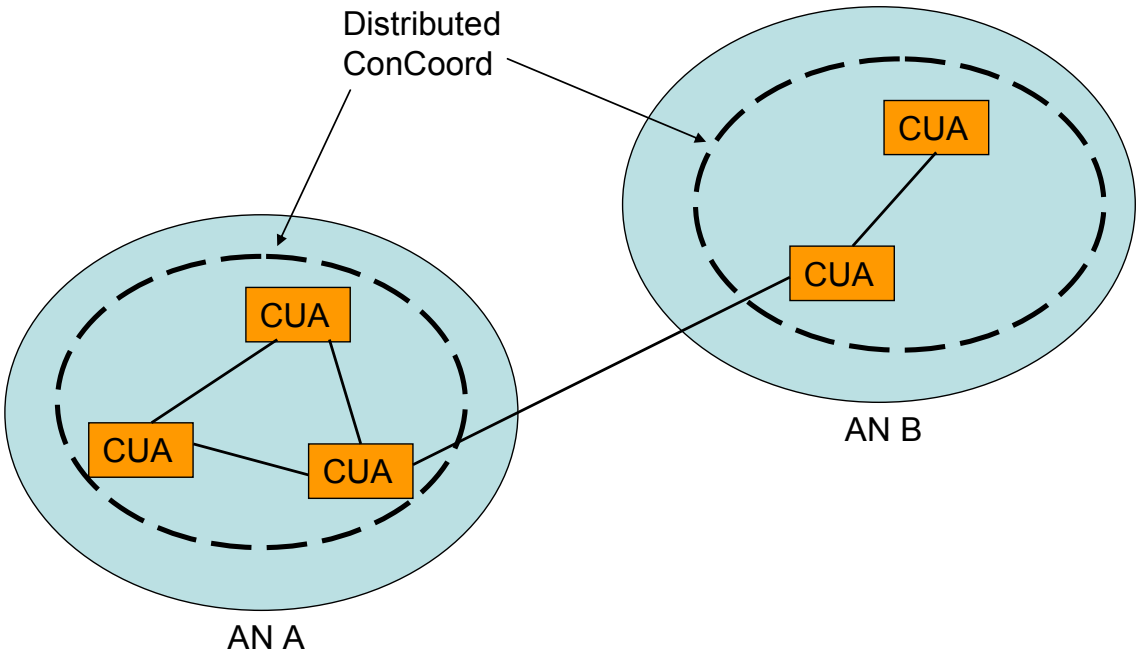


Figure 24: Logical presentation of prototype

6.2 Prototype description

This subsection presents an overview of how the prototype has implemented the previously described Distributed Context eXchange Protocol (DCXP) protocol (see chapter 5).

6.2.1 Interpret XML

The protocol is using XML formatted messages to communicate with other entities. In order for a device to interpret a XML message a XML parser is needed. There exist multiple parsers for Java ME but the kXML [43] was selected. The main reason for selecting kXML is that it was one of the first XML parser explicitly for Java ME. kXML has now been used by the open source community for a couple of years and during this time it has been modified and upgraded to become more reliable and stable.

A second reason for using kXML is that Vidal [4] used it in his prototype and it proved there it provides the function needed to fulfill the XML parsing on a handheld device with limited resources.

6.2.2 ContextWare Entities

The prototype implements a CUA acting as a CC node in a DHT overlay helping applications and services using the CUA to register and resolve UCIs referring to context information objects. A CUA in the prototype acts on behalf of an application using the CUA when it wants to interact with the ContextWare.

The CM functionality was not implemented thus a CM uses the CC functionality each CUA implements regarding registering and resolving UCIs. A CM's main task is to aggregate context information and a CM is able to do that with help of a CUA, thus a CM needs to have logic implemented of how and which context information should be aggregated and how.

6.2.3 Context User Agent API

In the prototype a CUA holds an API for applications and services to use when handling context information. This API corresponds to Figure 3 and section 5.4 together with the message specification in section 5.7. Primarily are the methods REGISTER and RESOLVE (i.e. the methods that interact with the CC). These methods are in detail:

- REGISTER
  This method is used when an application wants to register a UCI in the distributed CC.
- RESOLVE
  This method is used when an application wants to resolve a UCI into the location of a CIO.

The following messages in the prototype are implemented as a part of the CUA. But it is not necessary to do that as these messages are primarily sent between a source and a client.

- GET
  This method is used when an application wants to retrieve a CIO only once. This method is called after RESOLVE thus an IP address in needed in order to know where to send the GET message.
- SUBSCRIBE
  This method is used when an application wants to periodically retrieve a CIO upon a change. This method is called after RESOLVE thus an IP address in needed in order to know where to send the SUBSCRIBE message.
- NOTIFY
  This method is called when an application wants to notify a change in a registered CIO.

An application that wants to use the ContextWare in order to receive or distribute context information has to execute the prototype. The application using the prototype must host methods for a CUA to report back the responses from the above described methods.
A logical overview of the application stack looks like this:

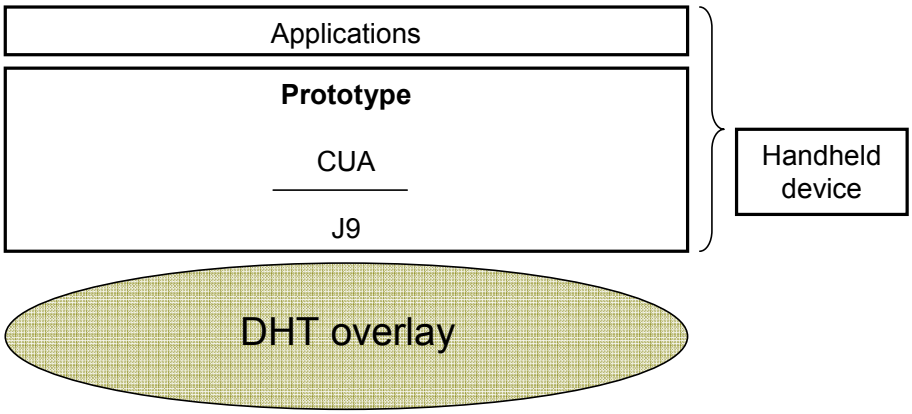


Figure 25: Overview of prototype's application stack

6.3 Limitation

The prototype had to be adjusted according to the environment and the available resources compared to the protocol described in previous chapter. In the following text these limitations are explained and why they were not implemented as described in chapter 5.

6.3.1 Bootstrapping

The discovery mechanism of the DHT overlay could be implemented in many different ways. For the implementation of the prototype a simple discover mechanism is the most suitable way. A broadcast mechanism could simply flood the test network with discover messages and still not benefit the prototype as it might be replaced by a new functional entity handling the lookup.

The prototype is implemented in a network that is limited in size and all participating nodes are located within the same subnet and all are able to directly reach each other, although all nodes are not in the same AN. Therefore is the bootstrapping mechanism a simulated functional entity giving an IP address of a node already in the DHT overlay to a node waiting to send a DISCOVER message. Each node running the prototype has from start up a list of IP addresses, containing at least one, to which it could send DISCOVER messages.

6.3.2 Finger table

A finger table is a help to a node in a DHT overlay to make routing more efficient as the finger table gives a node a wider view of the overlay, not only the closest succeeding nodes in the successor list but only nodes more far away (i.e. far away from the perspective of ID space).

The prototype is only evaluated in a smaller network which consists of less than 10 handheld devices. Therefore are finger tables not be implemented as this has no impact of the prototype's performance. A successor list containing each node's two immediate successors and a predecessor list containing each node's immediate predecessor are sufficient. This gives each node the ability to avoid a failure in the overlay due to one collapsed node at its immediate successor. It also gives each node a possibility to make routing more effective as it could redirect a message not only to its immediate successor but also to a node one node away.

7 Evaluation

This chapter presents the evaluation of the protocol, how the evaluation was done along with the results. In order to evaluate the proposed solution measurements of some key functions are needed together with theoretical analysis. The result of these measurements indicates if the P2P structure is efficient in distributing context information, how resilient the overlay is to node failure and how much of added computational power is needed. All of this together shows how appropriate a P2P solution is to ContextWare in Ambient Networks.

7.1 Test setup

The tests in evaluating the prototype is carried out in a WLAN only used by the prototype on channel 1 and in an environment where there exist two other WLANs on channel 2 and 11. The entities running the prototype are two different models in the HP iPAQ Pocket PC series, specifically h5550 and HX4700. All five entities are connected to the wireless network with their 802.11b network cards. The PC is a HP in the nc6000 series with an Intel Pentium M 1.4 GHz processor and 512 Mb RAM memory. The wireless network is hosted by an access point called 3Com OfficeConnect Wireless 11g/b. The encryption method WPA is used but has no impact of the results of the tests. More information about the logical set up of the network is presented in section 6.1.

The measurements are made with help of using an application called Airscanner® Mobile Sniffer for Pocket PC [44]. This application stores all the packets sent and received on the wireless network card within an iPAQ. The information is stored in a format which can be used by Wireshark [45] (formerly known as Ethereal). Therefore the results from the measurements can be analyzed on a regular computer.

7.2 Evaluation model

These tests enable us to evaluate the strengths and the weakness of the proposed solution. The latency is measured in different scenarios; where each scenario shows some specific key features of a P2P-system. These are the tests that are made:

Scenario	Section	Key feature
Network latency	7.3.1	Network delay
XML interpretation latency	7.3.2	Processing effort
Registration latency	7.3.3	DHT Efficiency
UCI Registration latency	7.3.4	DHT Efficiency
Context fetching latency	7.3.5	DHT Efficiency
Overhead signaling	7.3.6	Scalability/Churn
Composition latency	7.3.7	Scalability
Recovery	7.3.8	Redundancy

Table 1: Evaluation scenarios

The Distributed Context Exchange Protocol (DCXP) enables the distribution of ContextWare by providing a means for a node to participate in a DHT overlay, thus it can resolve and register UCIs in order to collect and disseminate context information. The measurements are performed to calculate the latency of a registration within the overlay and to learn the correct node as compared to being redirected multiple times. Another scenario concerns collecting context information, here the UCI first has to be resolved with help of the overlay into an address to the source and then the source has to be contacted in order to fetch the context information. Moreover, the overhead due to DCXP, in order to have the overlay functional and stable, without either register or resolve is calculated. This is compared to the increased number of messages as the packet loss ratio increases.

The overlay consists at most of five different nodes each running one instance of a CUA with an application of top controlling the CUA. In all the figures below that present each scenario the name have no correlation with how the hash is calculated; the figures are only there to show each step visually. Each node has a successor list containing its two closest successors and a predecessor list containing its immediate predecessor.

The prototype was configured with the following configuration during the evaluation:

- Time-out

  During development of the prototype a randomly behavior of the iPAQs were recognized. This behavior occurred sometimes when a node was supposed to reply with a response to a received message, the reply was delayed sometimes between 1 to 2 seconds. It could occur sometimes twice in a row and there was no correlation to which type of message that was being sent or received. This behavior was also recognized by Vidal [4]. In order to minimize the disturbance with resent packets and duplicates, a time out was introduced. After initial testing a time-out of 2.5 seconds was found suitable for the prototype. A 2.5 second time-out could be seen as very long, but there is a big difference between handheld devices and regular computers. These handheld devices seem more unreliable compared to a fixed computer with regard to networking, as they sometimes delay transmission of messages.

  A longer time-out than 2.5 seconds would end up in very long latency times since retransmission of messages are delayed more than needed. A shorter time-out could have been selected but that would have caused more retransmissions and duplicate messages.

- Number of retries

  Retransmission of a message can not be extended into eternity, there has to be a maximum count. This is needed since packets can get lost during the transport from sender to receiver and then a new packet needs to be sent in order to get the message to the receiver. The maximum number of retries in the prototype is selected to be 10. The only situations that demanded such a high number of retires where when simulation of lost packets were introduced into the prototype. In general a smaller number could have been selected, but when the prototype was developed and the packet loss was simulated a large number as 10 was needed in

order for the prototype to work especially during the tests were the packet loss rate were 40% and 50%.

- Time between UPDATE_REQUESTs

  UPDATE_REQUESTs are sent periodically to each node's immediate successor in order to notify about its existence and also retrieve a current view of the overlay further away. In the prototype UPDATE_REQUESTs are sent 2 times each minute (i.e. each 30 seconds). The time span is dependent on the characteristics of the entities in the network. If entities are joining and leaving frequently a shorter time span is selected, but this creates more overhead compared to a longer time span. In order to help the overlay, nodes should leave the network with sending a UNREGISTER message in order to notify the surrounding entities about it is leaving. This would help the nodes in updating their successor list and predecessor list, thus update requests are not necessary to transmit as often as if no UNREGISTER messages are sent.

- Hash algorithm

  The selected hash algorithm in the prototype is SHA-1 due to that Java natively supports it and that it is a widespread and known algorithm.

7.3 Test results

7.3.1 Network latency

Network latency, or the time a packet needs to travel from a sender to a receiver is important when starting evaluating the prototype as this gives information about how fast the network reacts. This is measured with two devices running the Airscanner application and registering the flow of messages. The latency is calculated as the figure below shows. The colored area marks the time which is of interest, thus the latency time in the network.

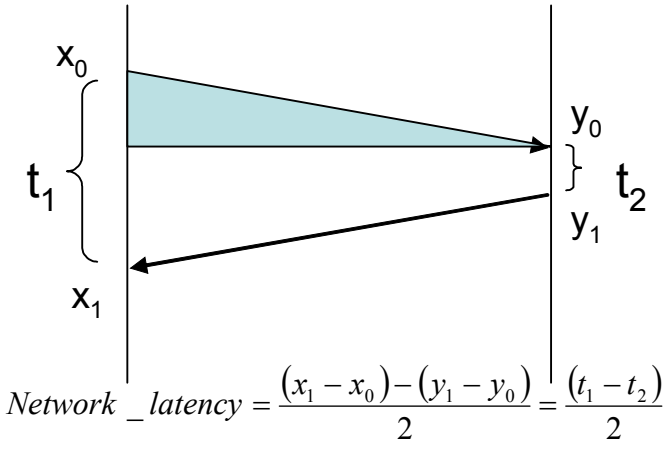


Figure 26: Network latency calulcation

The formula above gave the following result:

	Shortest	Longest	Average
Network latency	5.72 ms	9.01 ms	**7.22 ms**

Table 2: Network latency

As can be seen from the above, the network latency of this single link is approximately 7 milliseconds. Thus in the measurements reported in the next section, the network latency is insignificant.

7.3.2 XML interpretation latency

In order to look into how efficient kXML is compared to the overall performance a test was set up in order to see the XML interpretation latency. This latency is measured on the two different models of the PDAs and then compared to the time a regular PC needs to interpret the same XML message using the same code. The message that is interpreted is the REGISTER_RESPONSE message as this is the most complex of all the messages.

(Average)	hx4700	h5550	PC
XML interpretation	30 ms	37 ms	10 ms

Table 3:XML interpretation latency

The results show that XML interpretation is not the most significant part of the overall latency. The total time of processing a message is between 0.2 – 0.4 s which is shown in the following tests, this means that the XML interpretation is in the worst case 18.5% of the time a PDA spends when it receives a single XML message.

7.3.3 Registration latency

The registration process can be divided into a discovery part and the actual registration procedure. The discovery part requires roughly equal time as when each node joins it sends a DISCOVER message and will get a DISCOVER_RESPONSE message. The registration on the other hand could be different for different nodes as their registration might be redirected to another node. The total registration latency is the combined time of the two parts, i.e. the time from when the first discovery message is sent until registration is completed and the CUA is registered and able to resolve context information. Below three different scenarios are presented and illustrated in figure 27:

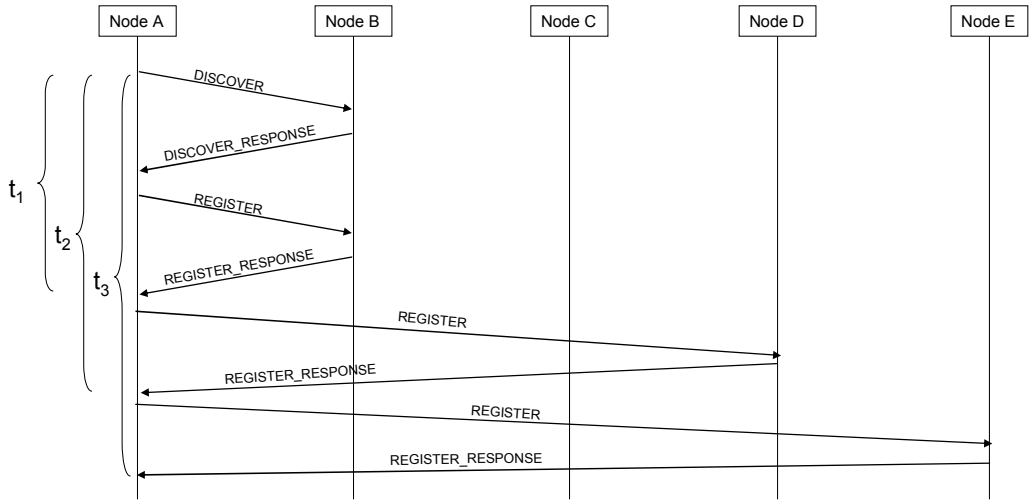


Figure 27: Three different registration scenarios

– t1, node A register directly at the correct node

	Shortest	Longest	Average
t1	1.29 s	1.68 s	**1.48 s**

– t2, node A is redirected once

	Shortest	Longest	Average
t2	3.16 s	5.96 s	**3.89 s**

– t3, node A is redirected twice

	Shortest	Longest	Average
t3	5.53 s	8.45 s	**6.19 s**

Table 4: Node registration latency

7.3.4 UCI Registration latency

The UCI registration is similar to the previous registration process. A UCI registration is typically routed the same way as a node registration, i.e. based on the hash value.

In this evaluation the UCI registration process tries to show what input a wider view of the network can give (i.e. each node knows not only about its immediate successor but also node(s) further away). In the figure below four different UCI registrations are done, $t_1 – t_4$. In the first scenario the responsible node (i.e. the node which should accept the registration of the UCI) for the UCI being registered is node B, in the second scenario the responsible node is node C, etc.

In the first two scenarios node A knows about both node B and node C due to its successor list. In the latter two scenarios node A needs help from node C in order to get in touch with node D and E.

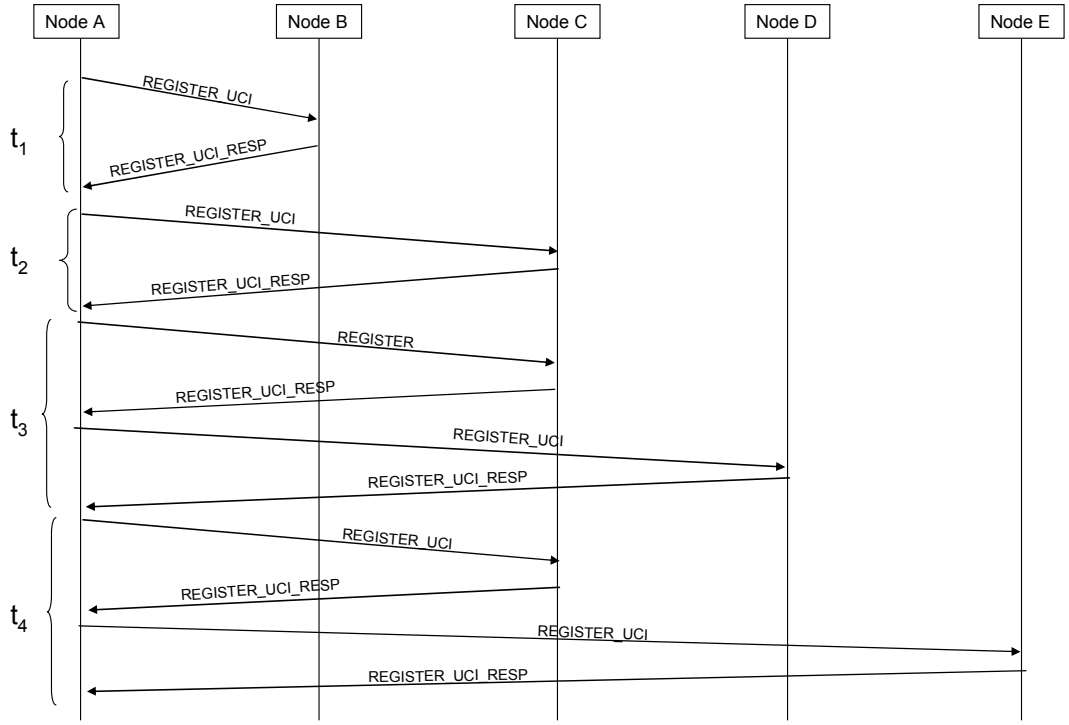


Figure 28: Four different UCI Registrations

- t_1, node A register a UCI directly at the correct node.

	Shortest	Longest	Average
t1	0.20 s	0.43 s	**0.35 s**

- t_2, node A uses its successor list to find out a node not its immediate successor.

	Shortest	Longest	Average
t2	0.23 s	0.53 s	**0.36 s**

- t_3, node A uses its successor list to find out a node not its immediate successor but is still being redirected.

	Shortest	Longest	Average
t3	1.09 s	1.32 s	**1.19 s**

- t_4, node A uses its successor list to find out a node not its immediate successor but is still being redirected.

	Shortest	Longest	Average
t4	1.22 s	2.12 s	**1.60 s**

Table 5: UCI registration latency

7.3.5 Context fetching latency

When an application at a CUA wants to access some specific CIO it first has to send a request to the local CUA to resolve the UCI into an IP address to the source. When the CUA has resolved the contact information and provided information to the requesting application the application has to fetch the CIO by either a subscription or a single get.

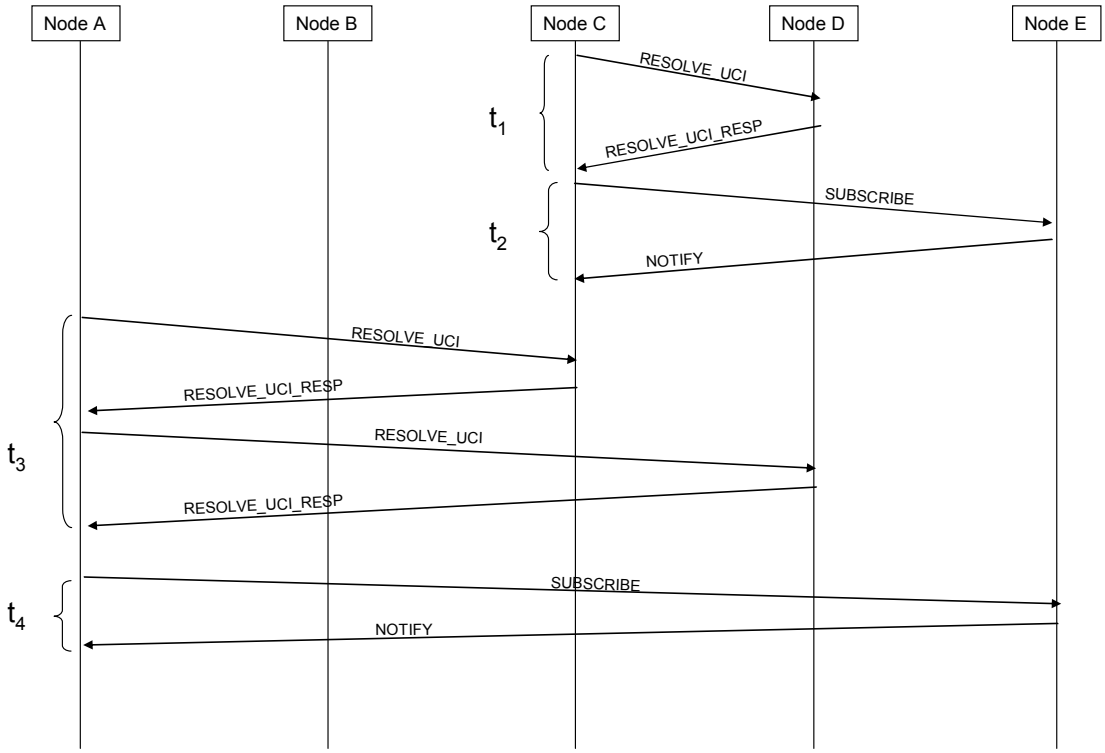


Figure 29: Subscribe and notify evaluation

Above in the figure two scenarios are presented. The first one is a scenario where the requesting node without redirects can resolve the UCI into an IP address. The second scenario involves a redirect before the actual SUBSCRIBE message can be sent. In these two scenarios the fetch message is of the SUBSCRIBE type, a GET message also causes the source to send a NOTIFY message.

– Node C resolves the UCI direct at correct node and then send a SUBSCRIBE message to the source. The source sends a NOTIFY message immediately.

	Shortest	Longest	Average
t1	0.18 s	0.36 s	**0.28 s**
t2	0.39 s	0.54 s	**0.45 s**
Total	1.10 s	2.33 s	**1.68 s**

Table 6: Subscribe scenario at correct node

– Node A uses its successor list to find out a node not its immediate successor but is still being redirected to node D. Then node A sends a SUBSCRIBE message to the source which replies with a NOTIFY message.

	Shortest	**Longest**	**Average**
t3	0.93 s	1.16 s	**1.03 s**
t4	0.42 s	0.60 s	**0.48 s**
Total	1.99 s	2.21 s	**2.10 s**

Table 7: Redirected subscribe scenario

7.3.6 Overhead signaling

The DHT overlay is going to be installed on all nodes in an ambient network. Therefore it is important to study how much resources it consumes from a regular node when the overlay is stable, i.e. overhead. This is measured in terms of bandwidth. How much of extra resources are needed when the DHT overlay is stable compared to how much of overhead (i.e. messages required to keep the overlay stable) when packets are lost.

Failure ratio	**Packets**	**Increased packet ratio**	**Avg. Packets/s**	**Bytes**	**Avg. Bytes/s**
0%	79		0.27	44328	148.7
10%	91	15%	0.30	51155	175.0
20%	93	18%	0.31	53393	178.0
30%	105	33%	0.35	61149	203.8
40%	114	44%	0.38	67003	227.4
50%	130	64%	0.43	78050	258.5

Table 8: Overhead signaling during increased packet loss

7.3.7 Composition latency

Composition is one of the key features of ambient networks and therefore it is important to evaluate what the effects and cost are. In the prototype only one of the two proposed composition scenarios are implemented, specifically the gateway method. Two tests were performed, first a test of how long time it takes for two networks to compose and be ready to gateway traffic between the two networks. The latter test shows the difference between a lookup of a UCI in its own overlay compared to using a gateway in order to do a lookup of a UCI.

When two networks have decided to compose ContextWare gets a request from the Management of Network Composition Functional Entity to start looking for other network's ContextWare overlay. This is done by sending a DISCOVER message telling about its existence. The process finishes when the other domain's name is registered as a UCI within the overlay, i.e. the latency representing t_3 in figure 30.

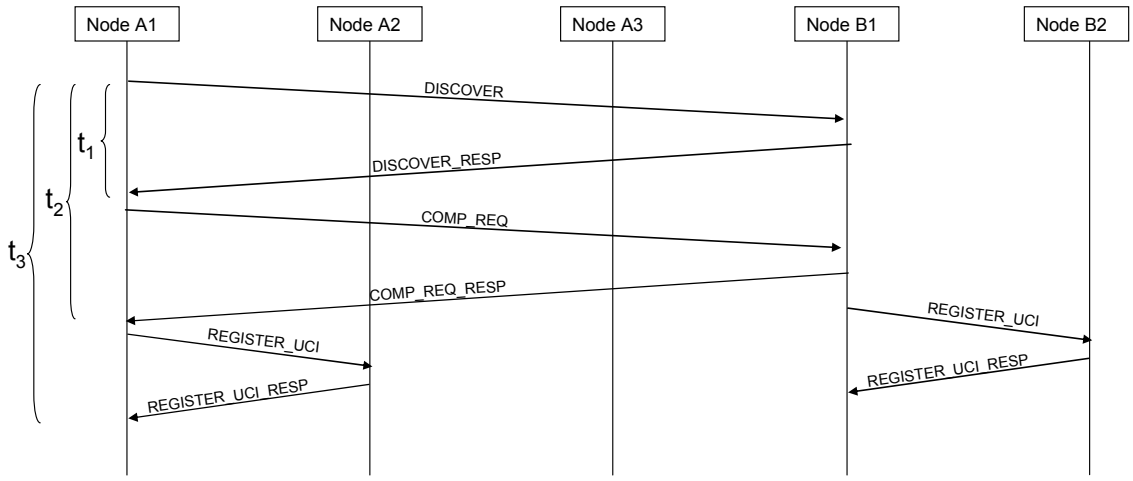


Figure 30: Gateway composition latency

	Shortest	Longest	Average
t1	0.27 s	0.40 s	**0.33 s**
t2	1.15 s	1.42 s	**1.26 s**
t3	2.14 s	2.44 s	**2.34 s**

Table 9: Composition latency

These results show that the time spent composing two networks is small compared to the time two networks are likely to be composed.

The second test, figure 31, begins after the discover process is completed and when a node from one ambient network is looking for context information from another. The process is similar to a lookup within its own overlay, but first the node must first find an address of a gateway to the other network. When a minimum of one address representing a gateway is found the resolve process is analogous to a lookup in its own domain. A RESOLVE_UCI message is sent to the gateway which then can either respond with an address to the source or redirect the requesting node to another node closer in the ID space. Finally when the source is found a SUBSCRIBE or GET message is sent and the source sends the context information back in a NOTIFY message.

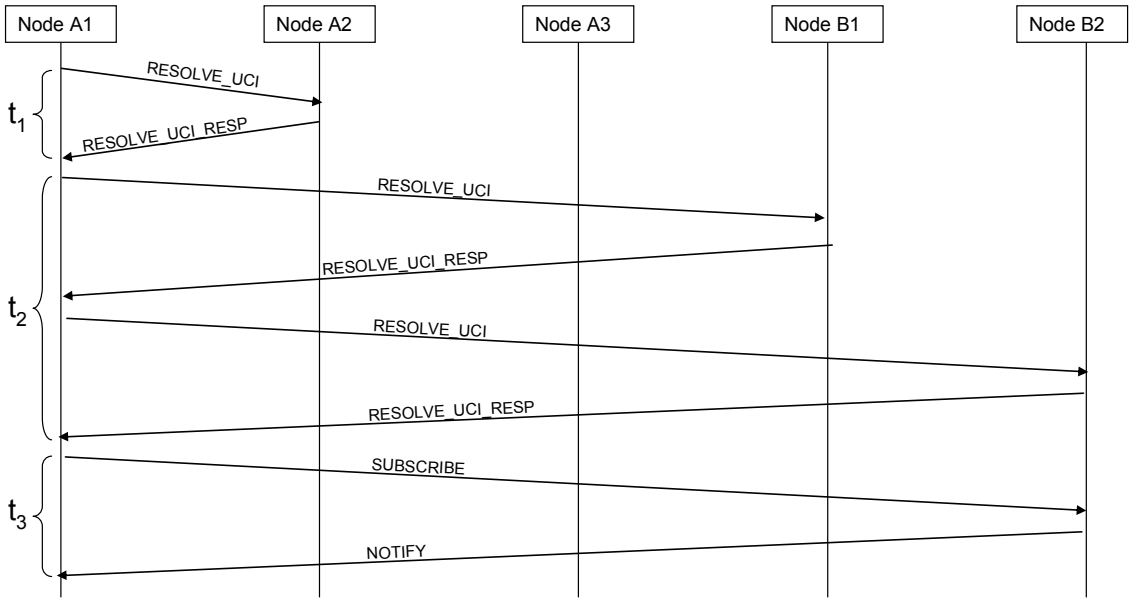


Figure 31: Resolve of UCI using a gateway

	Shortest	Longest	Average
t1	0.30 s	0.45 s	**0.37 s**
t2	1.27 s	1.89 s	**1.43 s**
t3	0.50 s	0.62 s	**0.59 s**
Total	3.48 s	4.88 s	**4.02 s**

Table 10: Resolve of a UCI during composition

7.3.8 Recovery

After a node has left the overlay without notice it is important that the DHT overlay quickly rearranges its nodes in order to fully operational. Therefore the UPDATE_REQUEST message is very important as it notifies a node whether its successor has left the overlay or not. If a node leaves the overlay without sending an UNREGISTER message the time it takes to notify the surrounding nodes range from 0 up to 30 seconds in the prototype. The full process includes the time between when a node leaves and the other nodes are notified about the absent node. The process is divided into three phases, t_0–t_2.

- t_0

  t_0 depends on the value of the UPDATE_REQUEST timer at the predecessor. The value varies from 0 up to 30 seconds in the prototype.

- t_1

  t_1 depends on the timeout value and the number of retries. In the prototype the value of t1 is the number of retries x timeout value, i.e. 10 x 2.5 = 25 seconds.

- t_2

  This value is the time it takes to send a message to its next successor and receive a reply. From measurements above this value is expected to be around 0.4 s.

– total

The total value is the sum of t_0 to t_2. This value varies between 25.4 and 55.4 seconds depending on where in the UPDATE_REQUEST cycle a node leaves the overlay without sending an UNREGISTER message.

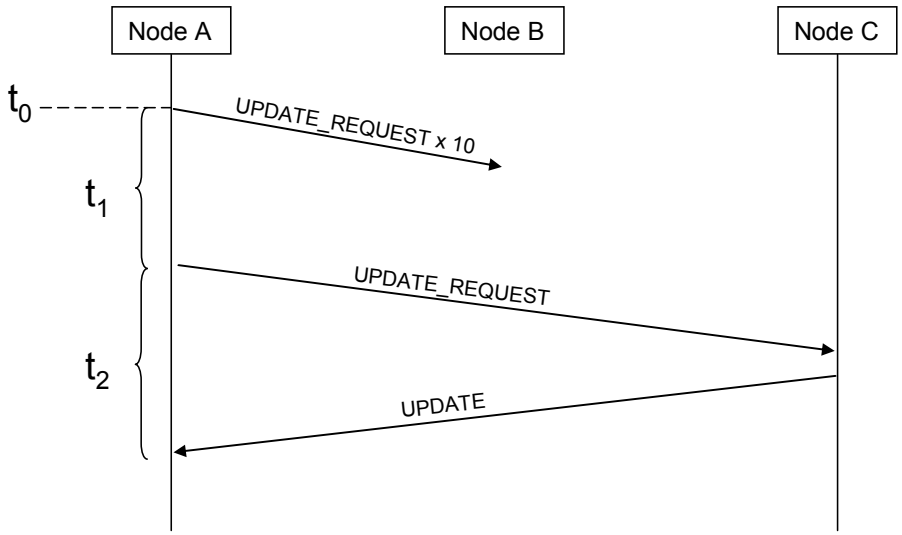


Figure 32: Update scenario when a node fails

If a node instead sends a UNREGISTER message to its immediate successor before leaving, then the procedure above is avoided and a lot of time is saved. The time consumption, t_1, is lowered to the same time represented by t_2 in the scenario without a UNREGISTER message.

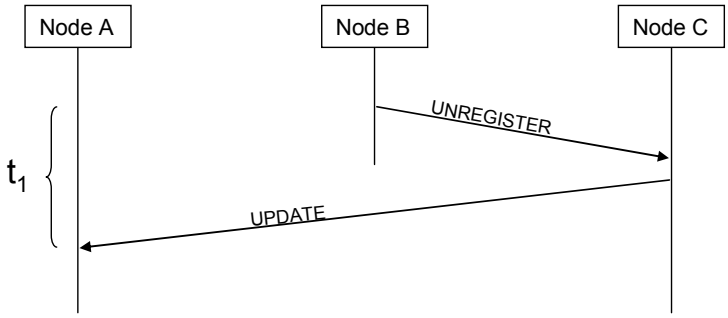


Figure 33: Update scenario when node sends unregister message

7.4 Discussion of test results

The evaluation showed the strengths and weakness of a peer-to-peer inspired ContextWare in ambient networks. These tests have given answers to questions and also indications of how to proceed further.

During the tests we found that the iPAQ's networking was not as reliable as a typical computer. They sometimes froze and did not respond promptly when a message was received. As a result of this their limited processing power is that processing a message takes significant time, around 0.2 – 0.4 seconds. This behavior is especially visible when more and more packets are lost. Thus, the handheld devices must process more information than in the other scenarios when some packets were not received.

The main impression from these tests with the prototype is the need to minimize the number of redirects in every phase, as this is the most time consuming phase. Furthermore, the handheld device's processing time of each message was a major part of all of the measured phases.

The tests show that latency increases linearly with the number of redirects a node is faced with when it makes a registration or lookup. Both registration and lookups are time consuming and this time depends on how many hops are required in order to find the correct node. As the lookup time should be as low as possible this process should be improved with help of finger tables, especially in larger networks. In smaller networks such as the evaluated network a finger table only causes **more** overhead. As a result a finger table is only useful when the size of the successor list grows too big and a node can not make efficient lookups in it. The size of the successor list is reduced when a finger table is introduced as it will assist in lookups that are farther away than those that the successor list holds.

A big issue in order to minimize overhead and reduce lookup times is that each node should unregistering when leaving instead of just leaving and relying on stabilization methods. This is even more important in networks where nodes join and leave regularly, i.e. mobile networks. The tests show that the time the stabilization method takes in order to find an absent node varies. During this time routing of messages could be affected by the missing node and messages could get lost. Another aspect is that when a node tries to send a message to a node that does not respond it tries again for a number of times, this keeps the node busy, until the node can consider the non-responding node as an absent node (i.e., after a predefined number of attempts. This procedure is avoided if a unregister message is sent.

In order to limit the stress on the overlay when nodes join and leave regularly super-nodes could be implemented. Instead of having all nodes that want to exchange context information join the DHT overlay only some nodes do. These nodes, so called super-nodes, then create the overlay - they are assumed to have reliable connections, enough processing power, etc. An ordinary node contacts one of these super-nodes in order to exchange context information, this could be achieved with help of the messages described in section 5.7. This also limits the overhead needed in order to keep the overlay functional.

In a network where nodes are mobile, long latencies and lost packets can occur. A DHT still works under those conditions, but the overlay creates more overhead and demands more work from the participating nodes (i.e. retransmissions, duplicate packets, etc.). The simulations show that in a network where all nodes suffer from a packet loss the overhead is increased by 64% when one out of every two packets is lost. Under more realistic circumstances of 10-20% of packet loss the additional overhead is below 20%.

Once a UCI is registered it is possible for nodes to get in contact with the source. The source does not need to update the UCI registration every time the CIO is changed, only before the registration times out. When a CIO is changed all subscribing entities are notified with help of a notify message. In the prototype all nodes are responsible for updating and managing subscriptions of the UCI they registered. Registration is a small fraction of all the messages, as a unit only registers once and then is registered until it leaves, compared to the number of messages handling dissemination of context information (i.e. subscribe, get, and notify messages). The tests show that a lookup in the overlay is the most time consuming part, not the actual fetching of context information. A node in the prototype can make a lookup once

and then get the requested context information multiple times as it knows where the source is located. This reduces the work for the overlay.

When two networks are composed the ContextWare could be composed in two different ways, either using a gateway or creating a new combined overlay, in the implemented prototype and during simulations gateway composition is used. Thus two regular lookups have to be performed before the actual context information could be requested, hence the lookup time is doubled. First the requesting node has to do a lookup in its own domain to learn at least one of the gateways representing the requested domain; then the actual lookup in the other overlay can occur with the help of a gateway node. The first part of the process could be accelerated if nodes after composition are done resolve the other domain and store the list of gateways. Then when a request of a UCI from another domain is issued it could send the request directly to a gateway node.

8 Conclusions and future work

This, the final chapter states what has been accomplished together with conclusions and suggestions for future work.

8.1 Conclusions

The objective of this thesis was to design and implement a distributed ContextWare providing means for entities in an ambient network to collect and disseminate context information. It should also allow networks to compose in order to allow entities from different networks to exchange context information.

This thesis proposed a distributed ContextWare solution fitting the Ambient Networks project ideas. The solution has been designed, implemented, and evaluated. There is research [4][13] about dissemination of context information in the sense of ambient networks and as well about distributed systems, although the combined research about distributed ContextWare is missing. I did not focus on which algorithm to use, but rather I focused how a distributed ContextWare works from an overall perspective. To select another and perhaps more appropriate algorithm would be a thesis itself. The first part of the project provided a good background and overview of the problem area. This gave me good insights into DHTs and how the Ambient Networks project sees the future of networking in a world with heterogeneous access technologies.

The second part was to design the distributed ContextWare and specifically the protocol behind it. The protocol needed to interoperate with prior research within the Ambient Networks project, such as protocol definitions and ContextWare entities. The proposed solution allows an entity to join an ambient network and to collect and to disseminate context information. It also provides a means for networks to compose and decompose in order to exchange context information between two different networks. The design also allows changing to another type of DHT.

Further more, a distributed ContextWare system was implemented on handheld devices in order to prove the feasibility of the solution in a suitable environment. This allowed applications running on the handheld devices, where the prototype was installed, to exchange context information with other entities within a network and between different networks.

Moreover, an evaluation of the implemented protocol was conducted in order to test the proposed solution and draw conclusions about to proceed further. This was the most challenging part as this shows how well the proposed solution actually works in a real-life scenario. The tests show how a distributed ContextWare performed in different situations and in different network scenarios.

Finally, this work has shown that a distributed ContextWare system is feasible. Although more tests is needed, especially in larger scale implementations. The latencies in section 7.3,

specifically sections 7.3.3 and 7.3.5, are too long for a real-life system where the demands are an almost instant response to a query. In order to limit the latency the look up procedure should be improved by minimizing the redirects. Other ways of improving the DHT algorithm are suggested below. The main impressions are although that the two major obstacles of the prototype are the Chord algorithm and the implementation issues.

If and when a distributed ContextWare is introduced into an ambient network the ContextWare could take advantage of a more integrated solution. New entities with better processing power, more bandwidth, etc. are another benefit. The solution could also be implemented natively into entities and networks, unlike the prototype where an application was running within another application (i.e. the prototype was an application within a JVM on top of Windows Mobile). This should lead to reduced latencies and more efficient networking. A user of the prototype that has to wait for almost a minute before the node reacts to a partition is probably not a happy user, although the prototype has shown that a ContextWare can benefit from a distributed solution; it has also shown that not all nodes should be members of the overlay.

A distributed ContextWare must be able to support different kinds of ambient networks and different network situations as ambient networks ranging from smaller networks such as PANs up to large enterprise networks such as WANs. These scenarios are below presented in more detail.

- PAN

  In a Personal Area Network, which consists of a few numbers of nodes, a distributed overlay is of minor use as an overlay creates more overhead compared to a solution where only one node hosts the ConCoord functionality (i.e. a distributed overlay has to maintain the overlay). This is further explained in the sections below. In a PAN a more client-server function is fulfilling the needs without demand commitment from all nodes. In a PAN some nodes might be low-powered and only got a low-bandwidth connection to the other nodes in the network.

- LAN

  A Local Area Network could be in the range from smaller networks in home to large enterprise networks. Generally in a home LAN the demands on a centralized ContextWare is low and the ideas in PAN scenario are used. In a larger LAN, enterprise, there are more needs for a distributed overlay as more units forms the overlay and the load on a central server could be high. A LAN is usually built up by nodes that have reliable links and good processing power, therefore are most of the nodes able to host the overlay functionality and help in the overlay structure.

- WAN

  In a Wide Area Network which most often consists of a majority of mobile nodes a distributed overlay is still functional, although as mentioned in the following section super-nodes should be implemented in order to limit the overhead caused by nodes joining and leaving frequently. Another aspect is that mobile nodes have higher latencies and could as well suffer from temporary network loss due to the nature of wireless networks. This creates more

overhead as nodes have to stabilize the overlay frequently which process more overhead.

In the following scenario the aspects of the descriptions above are considered. The result is that a node could be involved in two different types of networks and acting differently in the two networks at the same time.

- A network 'A' is formed by a person's devices in a PAN where one node, his laptop, is selected as the super-node and then acting as Context Coordinator for the network. Then can other nodes like cell-phone contact the node in order to use be able to disseminate or receive context information.
- Person 'A' enables his laptop for communicating over a 3G network, i.e. a WAN. When the laptop connects to the 3G network it becomes a member of that ambient network and starts to act as a gateway for the other nodes in the PAN when they want to reach context information from the WAN network. From the WAN perspective the laptop could be selected as a super-node, regular distributed node, or only join the ContextWare of the WAN using another node, this is up to the selection algorithm.

8.1.1 Scalability issues

An important aspect in all the three scenarios above is the scalability of a distributed ContextWare and how much of traffic a solution like this will generate. A distributed ContextWare is most optimal when the nodes and network are not suffering from extremely bad conditions such as packet loss, nodes joining and leaving frequently due to wireless nature, etc. In such extreme cases nodes that are reliable should be selected as super-nodes with the other nodes using these super-nodes as gateways into the ContextWare instead.

A distributed approach is lacking a central set of servers which are reliable and can provide users with a good and continuous service. But there exist distributed applications such as BitTorrent that employs a large number of widespread nodes using Internet to connect to each other. This, together with links that are fault tolerant and/or tolerates latencies within reasonable limits shows that in order to have a distributed ContextWare that scales even in larger implementation more research is needed.

ContextWare and the stored context information are of interest for many different purposes. The main reason is to provide users with the best network connection possible based on available networks, and the requirements and capabilities of each user. Managing networks (especially wireless networks), composition and decomposition, user application demands, and network operators' requirements are among the task that can use ContextWare in order to provide the best connection possible.

A major concern for the ContextWare is that it should fit the smallest PAN but also the largest WAN. The demands that those two extremities have are very different as described above. A set of replicated servers are not necessary in a smaller networks but larger enterprise networks could benefit of that.

To summarize, a distributed ContextWare is usable in a lot of scenarios, but in order to build a complete ContextWare solution that suits all scenarios the DCXP should be combined with the capability to use a central-server solution. This could be combined with the implementation of super-nodes as suggested below. A dynamically distributed ContextWare which is able to adapt to different network situations is a goal of future work.

8.1.2 Open issues

In order to improve and develop DCXP the following improvements should be investigated:

- Neighbor selection algorithm

  Improve network performance by selecting a specific neighbor instead of a random neighbor. Use a structured ID space which represents where each node is logically located. This would decrease latency especially when the networks are larger and nodes are more widely spread.

- Super-nodes

  In order to enhance the overall performance in an overlay not all nodes are suitable for joining as a regular node, this is due to limitations in processing power, bandwidth, asymmetric connections, etc. Therefore some nodes should be selected as super-nodes, these acts as gateways to the overlay for other nodes. A node then contacts a super-node and can subsequently interact with the distributed Context Coordinator and collect and disseminate context information.

- DHT Algorithm

  If a distributed system is going to be similar to the prototype using a DHT algorithm, then more research is needed in order to study suitable algorithms and select the most appropriate one(s). Different algorithms could suit different networks. A lot of study has been made and a recent dissertation [31] has focusing on improving a variant of Chord called Distributed k-ary System. Other research has shown that the Chord algorithm can be improved when exploiting proximity [46] of the underlying network topology.

8.2 Future work

A different approach than a distributed ContextWare should be explored in order to find the best solution that fits the Ambient Networks project. Other possibility architectures than the proposed solution are presented below, specifically in the following areas.

- Distributed approach with super-nodes used as central servers

  A distributed approach as mentioned before should further investigate how to implement super-nodes. When super-nodes are implemented the obvious question is which nodes should be selected as super-nodes. This should be explored and the selection algorithm could eventually select one node as super-node (i.e. this super-node acts as a single Context Coordinator). This selected super-node is then used as a central server in the PAN case and when more and more nodes join the selected super-node(s) are also increased. This combines the two architectures of distributed ContextWare and client-server ContextWare.

  The most important issue when introducing super-nodes is how the selection algorithm should work. This is where most development effort should be put, as this determines how and where the data is stored and routed. The selection algorithm must take into account what type of network it is, what this node is capable of, etc. in

order to make the selection of a more distributed approach instead of a more centralized approach or vice versa. The selection algorithm should also be dynamic which helps when a network or a node changes characteristics.

- Replicated servers

  Instead of having a distributed ContextWare where the participating users cooperatively form an overlay the ContextWare could be replicated on a set of servers. These replicated servers should then serve users when they want to store or fetch context information. If they are in a large widespread network the servers should not be together but placed in strategic parts of the network in order to serve users faster. A potential inspiration to a system like this could be how the Domain Name Systems DNS works and the extension with Dynamic DNS which supports real time updates of records.

- Security

  Security must be introduced. Not only to protect the actual data that is transmitted between entities, but also prevents entities from impersonating another entity. In an upcoming master thesis report by Nupur Bhatia [8] policies regarding ContextWare are investigated. This is an area which is very important as access to context information needs to be restricted, especially as sensitive information could violate the integrity of a user. Policies are used to set up rules about who is allowed to access specific context information. Her work should be introduced into a distributed ContextWare.

  Security is not only about protecting context information, but also how to prevent a malicious user from impersonating another. Much more work is needed in order to solve these sorts of issues regarding a distributed ContextWare.

- Storing policies and context information in the same distributed overlay

  The distributed overlay which is used in the ContextWare is also able to store policies. In order to store and find a policy each policy must have a unique identifier. An ambient network could have one distributed overlay which could serve multiple purposes such as storing UCIs, policies, etc.

References

[1] Telefonaktiebolaget LM Ericsson, *Press release: 3 and Ericsson sign major managed services agreement,* 2005-12-06, http://www.ericsson.com/ericsson/press/20051206-100037.html (accessed 2006-06-26)

[2] *Ambient Networks Project*, http://www.ambient-networks.org/ (accessed 2006-06-26)

[3] Ambient Networks Project WP6, *Ambient Networks ContextWare - First Paper on Context-aware Networks*, Technical report, IST–2002-507134-AN/WP6/D61, 2005-01-07

[4] Sergio Quintanilla Vidal, *Context-Aware Networks: Design, Implementation and Evaluation of an Architecture and a Protocol for the Ambient Networks Project*, Final Thesis, Linköpings University, 2006-03-13

[5] A. B. Roach, *Session Initiation Protocol (SIP) – Specific Event Notification*, IETF RFC 3265, June 2002

[6] Ambient Networks Project WP1, *AN Framework Architecture*, Technical Report, IST-2002-507134-AN/WP1-D05, 2005-12-30

[7] Teemu Rinta-aho, *Introduction to Ambient Networks*, Presentation, NETS 1st seminar, April 2005

[8] Nupur Bhatia, *Context-Aware Networks: Policy Management*, Upcoming master thesis report, Royal Institute of Technology, Stockholm, January 2007

[9] Ambient Networks Project WP-D, *Integrated Design for Context, Network and Policy Management*, Technical report, FP6-CALL4-027662-AN P2/D10-R1, 2006-12-21

[10] T. Berners-Lee, R. Fielding and L. Masinter , *Uniform Resource Identifier (URI): Generic Syntax*, IETF RFC 3986, January 2005

[11] A. K. Dey, *Understanding and Using Context*, Personal and Ubiquitous Computing Journal, Volume 5, 2001, pp. 4-7.

[12] Ambient Networks Project WP6, *Ambient Networks ContextWare - Second Paper on Context-aware Networks*, Technical report, IST–2002-507134-AN/WP6/D6-3, 2005-12-16

[13] C.G. Jansson, et al., *Mobile Middleware for Adaptive Personal Communicatio*n, chapter VII.f in Corradi and Bellavista, The handbook of mobile middlewares, CRC Press, to be published in September 2006

[14] G. Camarillo, M. Handley, A. Johnston, J. Peterson, J. Rosenberg, E. Schooler, H. Schulzrinne and R. Sparks, *Session Initiation Protocol (SIP)*, IETF RFC 3261, June 2002

[15] *SETI@home HomePage*, http://setiathome.berkeley.edu/ (accessed 2006-08-15)

[16] Gerard Tel, *Introduction to Distributed Algorithms*, Cambridge University Press, 2000, ISBN 0521794838

[17] *BitTorrent Homepage*, http://www.bittorent.com/ (accessed 2006-07-10)

[18] *Skype Homepage*, http://www.skype.com/ (accessed 2006-07-04)

[19] *P2P Telephony Explained*, http://skype.com/products/explained.html/ (accessed 2006-07-04)

[20] J. Bound, Y. Rekhter, S. Thomson, P. Vixie, *Dynamic Updates in the Domain Name System (DNS UPDATE)*, IETF RFC 2136, April 1997

[21] R.Hinden and S. Deering, *IP Version 6 Addressing Architecture*, IETF RFC 4291, February 2006

[22] Ken W. Ross and Dan Rubenstein, *Peer-to-Peer Systems,* Infocom, Tutorial T6, 31 March 2003 http://cis.poly.edu/~ross/papers/P2PtutorialInfocom.pdf (accessed 2006-07-03)

[23] Klaus Wehrle, Stefan Götz and Simon Rieche, *Distributed Hash-Tables*, Lecture Notes in Computer Science Volume 3485, Springer-Verlag, September 2005, p. 79-93

[24] H Balakrishnan, M. F. Kaashoek, D Karger, R Morris and I Stoica, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, ACM SIGCOMM'01, August 2001, pp. 149-160

[25] Antony Rowstron and Peter Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, 18th IFIP/ACM International Conference on Distributed Systems Platforms, 2001

[26] Paul Francis, Mark Handley, Richard Karp, Sylvia Ratnasamy and Scott Shenker , *A Scalable Content-Addressable Network*, ACM SIGCOMM'01, August 2001

[27] Ron Rivest, *The MD5 Message-Digest Algorithm*, IETF RFC1321, April 1992

[28] Gregory R. Ganger, Shuheng Zhou and Peter Steenkiste, *Balancing Locality and Randomness in DHTs*, Technical Report, CMU-CS-03-203, Carnegie Mellon University School of Computer Science, November 2003

[29] D Bryan, C Jennings and B Lowekamp, *A P2P Approach to SIP Registration and Resource Location*, IETF Internet Draft, draft-bryan-sipping-p2p-02, March 2006

[30] K. Gummadi, R. Gummadiy, S. Gribblez, S. Ratnasamyx, S. Shenker and I. Stoica, *The Impact of DHT Routing Geometry on Resilience and Proximity*, ACM SIGCOMM'03, August 2003

[31] Ali Ghodsi, *Distributed k-ary System: Algorithms for Distributed Hash Tables*, Disseration, Royal Institute of Technology, Stockholm, December 2006

[32] H Balakrishnan, F Dabek, M. F. Kaashoek, D Karger, D Liben-Nowll, R Morris and I Stoica, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, IEEE/ACM Transactions On. Networking, vol. 11, pp. 17-32, February 2003

[33] *The Chord Project*, http://pdos.csail.mit.edu/chord/ (accessed 2006-07-13)

[34] D Karger, E Lehman, F Leightin, M Levine, D Lewin, and R Panigrahy, *Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web*, In Proceedings of the 29th Annual ACM Symposium on Theory of Computing (May 1997), pp. 654–663

[35] NIST, *Secure Hash Standard*, U.S. Department of Commerce/NIST, National Technical Information Service, FIPS 180-1, 17 April 1995 (this superseded FIPS PUB 180 of 11 May 1993), http://www.itl.nist.gov/fipspubs/fip180-1.htm

[36] Feng Cao, David A. Bryan and Bruce B. Lowekamp, *Providing Secure Services in Peer-to-Peer Communications Networks with Central Security Servers*, Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW'06), 2006, p. 105

[37] Sameh El-Ansary, *Designs and Analyses in Structured Peer-to-Peer Systems*, Doctorate of Philosophy Dissertation, Royal Institute of Technology, June 2005 (ISRN KTH/IMIT/LECS/AVH-04/03–SE)

[38] *The Bamboo DHT Homepage*, http://bamboo-dht.org/ (accessed 2006-11-21)

[39] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler and François Yergeau, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation, 16 August 2006

[40] *HP iPAQ hx4700 Series Pocket PC – Specifications*, http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?lang=en&cc=us&taskId=125&prodSeriesId=420534&prodTypeId=215348&objectID=c00251237 (accessed 2006-11-15)

[41] *HP iPAQ Pocket PC h5500 Series – Overview*, http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?lang=en&cc=us&taskId=125&prodSeriesId=322916&prodTypeId=215348&objectID=c00102129 (accessed 2006-11-15)

[42] IBM *WebSphere Studio Device Developer Homepage*, http://www-306.ibm.com/software/wireless/wsdd/ (accessed 2006-11-15)

[43] *kXML Homepage*, http://kxml.sourceforge.net/ (accessed 2006-09-22)

[44] *Airscanner Mobile Sniffer for Pocket PC Homepage*, http://airscanner.com/downloads/sniffer/sniffer.html (accessed 2006-09-07)

[45] *Wireshark Homepage*, http://www.wireshark.org/ (accessed 2006-09-07)

[46] Feng Hong, Minglu Li, Jiadi Yu and Yi Wang, *PChord: Improvement on Chord to Achieve Better Routing Efficiency by Exploiting Proximity*, Proceedings of the First International Workshop on Mobility in Peer-to-Peer Systems (ICDCSW'05), 2005, p 806-811

COS/CCS 2007-03

www.kth.se