Sixth Framework Program

Priority IST-2004-2.4.5

Mobile and Wireless Systems beyond 3G

Project 027662

"Ambient Networks Phase 2"

D10-D.1 Integrated Design for Context, Network and Policy Management

Date of preparation: **06-12-01**
Start date of Project: **06-01-01**
Project Coordinator: **Henrik Abramowicz**
Ericsson AB

Revision: **1.0**
Duration: **07-12-31**

Document Properties[1]:

Document Number[2]:	FP6-CALL4-027662-AN P2/ D10-D.1
Document Title:	**Integrated Design for Context, Network and Policy Management**
Document responsible:	Miklós Aurél Rónai (ETH)
Author(s)/editor(s):	Fatma Belqasmi (CIISE) Marcus Brunner (NEC) Lawrence Cheng (UCL) Simon Csaba (BME) Alex Galis (UCL) Raffaele Giaffreda (BT) Roch Glitho (CIISE) Alberto Gonzalez (KTH) Hamid, Harroud (SITE) Ian Herwono (BT) Kerry Jean (UCL) Theo Kanter (EAB) Ahmed Karmouch (SITE) Péter Kersch (BME) Zoltán Lajos Kis (BME) Johan Nielsen (EAB) Giorgio Nunzi (NEC) Börje Ohlman (EAB) Roel Ocampo (UCL) Rolf Stadler (KTH) Róbert Szabó (BME) Miklós Aurél Rónai (ETH) Nancy Samaan (SITE) Tomasz Szydło (AGH) Per Oddvar Osland (Telenor) Stein Svaet (Telenor) Göran Selander (EAB) Ambrus Wágner (BME)
Target Dissemination Level[3]:	PU
Status of the Document:	Public

[1] Input of Title, Date,Version, Target dissemination level, Status via *"File /Properties/Custom*" in the Word menu

[2] Format: FP6-CALL4-027662-AN P2/<Deliverable number>
Example: FP6-CALL4-027662-AN P2/ D1-A.1

[3] Dissemination level as defined in the EU Contract:

PU = Public
PP = Distribution limited to 6th FP participants
RE = Distribution to a group specified by the consortium
CO = Confidential, only allowed for members of the consortium

Version	1.0

Abstract:

The Ambient Network Management Systems contrasts with traditional management approaches mainly in the area of dynamic heterogeneous management-layer composability and self-manageability. Generally, traditional management systems are homogeneous, hierarchical and highly centralised and there is a clear separation between the roles of management systems and network elements.

The vision for Ambient Networks accounts for heterogeneous networks composing and cooperating, on demand and transparently, without the need for manual (pre or re)-configuration or offline negotiations between network operators. This document provides a collection of solutions that can lead to make this vision a reality.

Keywords:

Integrated Management, Context Management, Policy management, Distributed Management for Ambient Networks

Table of Contents

Executive Summary

The AN Management Systems contrasts with traditional management approaches mainly in the area of dynamic heterogeneous management-layer composability and self-manageability. Generally, traditional management systems are homogeneous, hierarchical and highly centralised and there is a clear separation between the roles of management systems and network elements. SNMP or CMIP are generally used, but they do not scale well in dynamic or large networks, and are not suited in particular for the dynamic (de)composing networks such as Ambient Networks.

The vision for Ambient Networks accounts for heterogeneous networks composing and cooperating, on demand and transparently, without the need for manual (pre or re)-configuration or offline negotiations between network operators. To achieve these goals, ambient network management must become dynamic, distributed, adaptive, self-managing, self-policing and autonomous responsive to the changes in the networking and context.

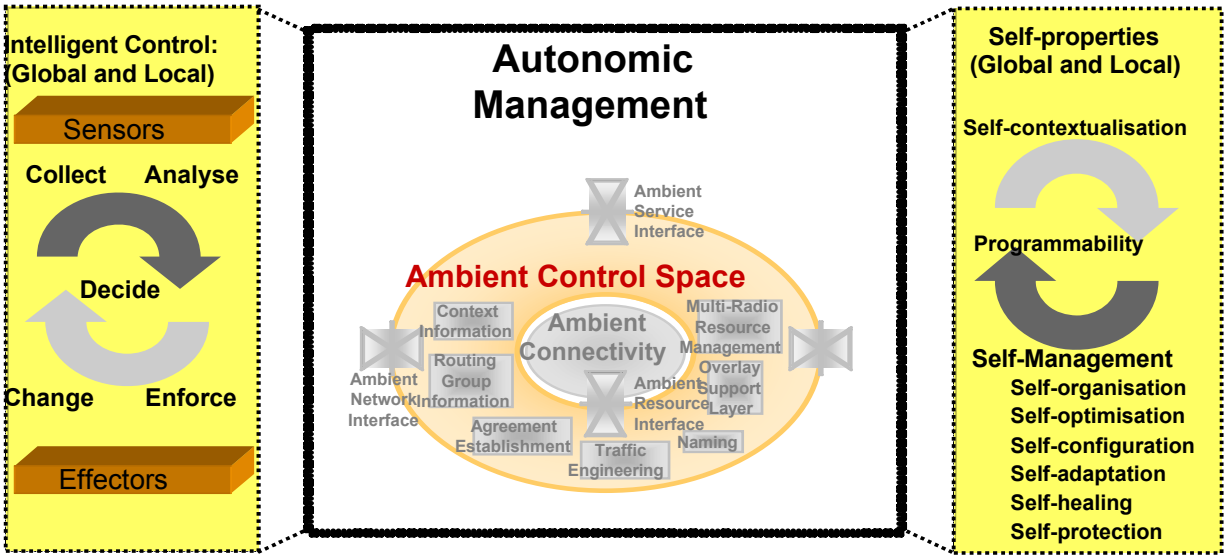


Figure 1 – AN Autonomic Management Systems

The full management functionality in an integrated management for Ambient Networks is not the goal of the WP-D work.

As such this document provides a partial management functionality picture of the overall integrated ManagementWare solution for Ambient Networks. The document presents various results on management solutions that support cooperating networks in highly dynamic situations where the need for autonomic behaviour is paramount to reduce complexity and operating costs. These results allow a level of network auto-configuration, so called “self-functions” (see Section 1.2 for more information about self-functions), self-functions, including self-organisation and reconfiguration.

WP-D Objectives

The main WP-D objective is to develop solutions and tools that enable an AN to manage itself in an autonomic, scalable and secure manner, through the use of context information, network monitoring and policies. The WP-D specific objectives for year 1 of the AN 2 project are:

- Specification and development of a context management system·
- Development of management functions for real-time monitoring, resource discovery and auto-configuration

- Creation of a scalable policy management framework in support of the operations of the Ambient Control Space·

Provision of an integrated management coupling context, policy and network management in support of ACS bootstrapping and reconfiguration, including design of a common distributed registry for AN, analysis of cross-layers management approaches, design of self-organised management structures, development of dynamic adaptation and instantiation of management functions and description of the ASI management primitives.

Being part of a wider project another objective for the WP results was to integrate them within the overall AN system description (WP-A) and to develop self-contained software modules that would plug into the wider AN prototype (WP-H) to provide some automated management features.

Work Approach

The WP was set-up as a merger of two major activities that took place during Phase 1 of the project, namely combining a context management infrastructure and at a number of distributed network management solutions with a framework for policy management for AN. The results inherited from Phase 1 were therefore a network-context management framework called ContextWare (*Deliverable D6-3 of the AN1 project [125]*) and new solutions for distributed network management (*Deliverable D8-3 of AN1 project [126]*).

The approach taken in WP-D includes identification of AN management requirements and interactions between different management systems in achieving common goals as well as interactions between the ManagementWare FE and other FEs of the Ambient Control Space (ACS). These interactions were captured in an initial model for an AN Management representing the building blocks needed to fulfil that vision for Ambient Networks. This model was created as an initial attempt to bring together the infrastructural features of different management sub-systems: a network-context management, those of a policy management system and the ones of a network management system.

Subsequently the approach involved detailed features description of the different management sub-systems and some of the possible interactions with systems and functional entities being developed in other work-packages.

Progress on WP-D results were capture in two early WP-D internal reports “Interim WP-D Report on Integrated design for context, network and policy management”, July 2006 [127] – and “WP-D/WP-E/WP-F Milestone Report on Cross Layers Interactions, Optimisations and Enhancements”, October 2006 [128].

Results

The progress towards the objectives stated above has been continuous and substantial; a considerable number of high-quality publications have been made. The integration of some proposed solutions has also been achieved within the system description. In spite of delays caused by a licensing issue with regards to the development and use of software components, an initial contribution of code was also made towards the overall AN prototype. First prototypes of the Context Management system and of the Composition Management system has been built and integrated into the common Ambient Network prototype.

Figure 2 depicts the AN ManagementWare model, which underpins the WP-D design and implementation work.

Below the Ambient Service Interface (ASI), we find the execution environments (EEs) for ContextWare (to the left) and ManagementWare (to the right), respectively. The ContextWare EEs operates on the ContextWare FE, while the ManagementWare EEs operates on the Composition Management FE that has an Ambient Network Interface

(ANI). Common to both is the distributed Registries Management FE that also holds the ContextWare and ManagementWare Information Bases (CIB/MIB).

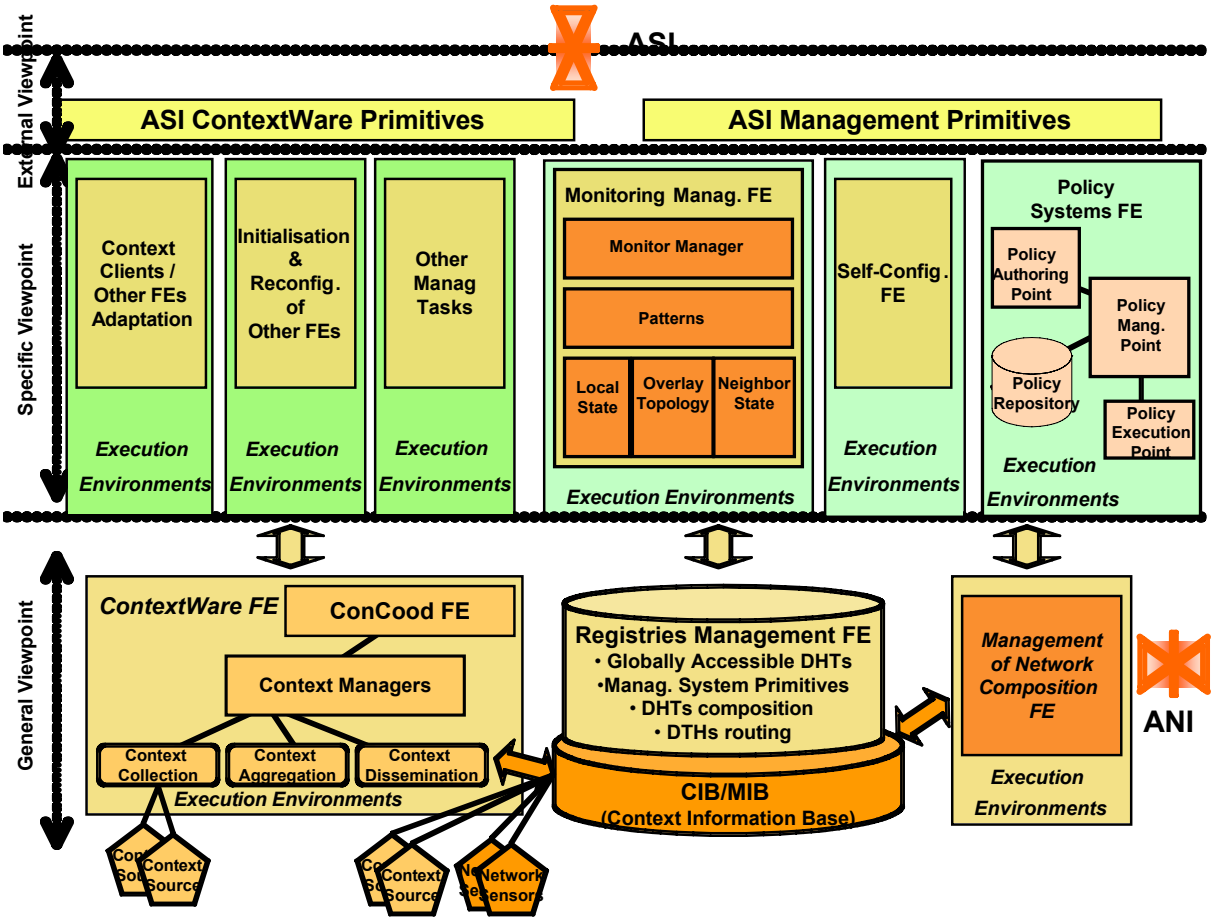


Figure 2 – AN ManagementWare Model

The main WP-D results in the first year of phase 2 are:

- *Context Management System design and implementation (D1)* – Full design and development of a novel network-level context management system. It has the role of managing the context information in the Ambient Control Space, including its distribution to *context clients/consumers*. Context clients are context-aware services, either user-facing applications/services or network services, which make use of or/and adapt themselves to context information. Network services in the Ambient Networks project are described as the services provided by a number of *Functional Entities (FE)*. The Context Management functionality helps to make the interactions between the different *context sources* and *context clients* simpler and more efficient. It acts as a mediating unit and reduces the numbers of interactions and the overhead control traffic. A model based evaluation of the performance for context distribution was developed.
- *ConCoord System design and implementation (D1*) – Full design and development of the ConCoord, which corresponds to a distributed registry that maps Universal Context Identifiers (UCIs) to the location of context information objects. The ConCoord maintains this registry by receiving REGISTER requests from context sources (entities providing one or more context objects). The ConCoord FE is the first point-of contact for context clients. When Context clients want to access context information, these clients send RESOLVE requests, which contain one or more UCIs to the ConCoord. The ConCoord then responds by returning the

locations of the corresponding context objects. Universal Context Identifiers (UCIs) are a new type of Uniform Resource Identifiers (URI). UCIs uniquely identify a given context object, but not its location within the network. Then clients contact the located context sources directly to GET the information, or to SUSBSCRIBE to context change events by receiving NOTIFICATIONs. The REGISTER and RESOLVE primitives with which the ConCoord communicates with the context sources and clients is a SIMPLE-inspired protocol. The ConCoord is implemented using Bamboo DHTs.

- *Context-aware Service Management design (D1)* – In cooperation with WP-F the network underlay ContextWare work has being extended and integrated with the service overlay (SATOs). By overlay context-awareness, we refer to the capability of a SATO to use network context information for self-adaptation, or in the provision of services. The provision of network context-awareness is needed in SATO in order to aid in distributed service management in heterogeneous wireless environments, in response to the ever-changing network context. As a summary, the context-aware functionality of SATO service management includes: i. support efficient deployment, activation, and (re)configuration of (new) Context Sensors in a scalable and distributed manner on (potentially) large numbers of AN nodes; in order to monitor and collect specific pieces of network context that are needed to support service management; ii. Provision of a scalable, distributed mechanism for locating (the collected) distributed network context within an AN; in order to support subsequent network context retrieval by AN service managers for service management.
- *Access Control and Authorisation in ContextWare (D1)* – Security domains will be created during AN bootstrapping and (re-)configuration phases, and nodes (or resources) will be enrolled into these domains in accordance with their security policies. The implication of security domain memberships of CW nodes on the mechanisms for accessing/retrieving context information from context sources (or managers) has been investigated. Our approach is to perform coarse-grained access control (at ConCoord) through security domain management and to run further (private) access policies at the context sources, if necessary.
- *Adaptive Decentralized Real time Monitoring: design, evaluation and implementation (D2)*- The *decentralized real time monitoring FE* provides other FEs and network administrators with network-wide state variables in real time. The client FE (or network administrator) specifies: (i) the network-wide variable it is interested in (e.g., the number of VoIP flows in the domain), (ii) the monitoring technique (e.g., periodic sampling or continuous) and (iii) the required accuracy. The monitoring FE provides the client with an estimation of the variable with the required accuracy and minimal overhead. (Monitoring distributed systems involves the fundamental trade-off between accurate estimation of a variable and the generated overhead). The monitoring FE creates a self-organizing layer that interconnects management processes on the network devices. This self-organizing layer is scalable, robust and adaptive to networking condition changes. The monitoring FE includes a number of protocols we have developed (e.g., Echo, GAP, A-GAP, TCA-GAP). Echo provides distributed polling. A-GAP provides continuous monitoring and can reduce the generated overhead by almost two orders of magnitude for allowing small accuracy errors. A prototype has been developed and is currently under evaluation. It will be integrated with the ACS framework in 2007.
- *Self-configuring management design (D2)* – Self-management is the dominant approach in Ambient Networks and different configuration tasks are performed independently by different FEs without the control of a central component. On the other hand, decentralized configuration requires in general some mechanisms to

accomplish correctly their configuration tasks. Stability and oscillations avoidance are the major issues that must be tackled in a distributed scenario. An architecture for plug-and-play base stations has been designed and evaluations have been conducted to investigate the behaviour of some of the control algorithm underneath. So far, stability has been enforced with a simple mechanism that blocks contradictory actions. With this experience, a specific FE has been proposed to support more advanced services to guarantee stability of distributed configuration tasks. This new self-configuring FE basically relies on instruments of control theory to mitigate the enforcement of configuration actions over time.

- *Management of Composition – design and implementation (D2/D4)* – A set of P2P hierarchical self-organization models have been developed along with a formal description and evaluation methodology to analyse maintenance of a logical network structure on top of a dynamically changing physical network topology. The Composition Management FE has been designed based on experience with the above self-organization models. It cooperates with the ACS framework, the Composition FE, and the Policy Management System to manage the logical structure of the ACS as well as FE instances in this structure. A first prototype was developed and is currently under integration with the ACS framework.
- *Policy Framework design (D3)*- Policies are used in a number of different areas within AN. Still all FEs in the ACS need to use the common AN policy framework to ensure consistency between the different policy sets. When composing there is a need to check that the policies remain consistent also after the composition. The policy framework developed by D3 was used for self-management and security domains in AN.

A number of management interoperability and integration techniques and enables were investigated and developed including:

- *Registry Management Design* (D2/D4) – The goal of a common distributed registry management FE is to exploit synergies and provide a unified distributed registry functionality for all functional entities. As such the registry management enable interworking and integration between all ACS FEs and in particular the management FEs. The design of this novel registry is based on DHTs, which are applicable to AN (i.e. they follow (de)composition of the ANs and efficient stochastic routing maintenance mechanisms guarantee low maintenance overhead in dynamic AN environments).
- *Dynamic contextualisation and adaptation; instantiation of management functionality* (D4) – a modular and scalable system facilities, called DINA, that enables the dynamic deployment, control and management of functionality (i.e. programmable sessions over network entities) was redesigned for AN project, including porting to FreeBSD. These facilities are aimed at use in year 2 work for on-demand dynamic instantiation and activation of new execution environments, management functionality, new network sensors, or for dynamic adaptation and self- contextualisation. DINA was used for the ContextWare prototype of WP-DF and it also used in the WP-F work.
- *Cross-layers Management approaches (D4)*- Cross-layer design enhances the traditional design, where each layer of the protocol stack operates independently by facilitating information exchange between layers and by optimising end-to-end performance of different problems. We are assuming that there is certain cooperation possible between AN underlay and Service overlay layers. Information and /or policies at the overlay could be provided at underlay and vice versa in order to better solve an overlay/underlay problem, including cross-layers optimisation. Joint work WP-F/ WP-E/WP-D has focussed on identifying the main principles for

cross-layers design and their used in solving different types of cross-layers interactions or optimisation problems, including: i. Interactions between AN bearers; ii. Underlay transport protocol performance optimisation; iii. Re-routing in the overlay based on congestions status in the underlay; iv. Underlay/overlay QoS interactions; v. Collecting and storing distributed network context information, to support service management in SATOs; VI. vii. Underlay/overlay policies' interactions. Cross-layer approaches are one of the most promising design models to serve as a blueprint for dynamic network architectures.

- *ASI Management Primitives Design (D4).* The ASI framework includes the specification and design of the ASI primitives ASI primitives, related to particular FE: ContextWare, Network Management, Policy Management have being developed in cooperation with WP-F.
- *Self-organisation Management Design (D4)* – a self-organising management system is needed to automatically and dynamically determine (or to select) a set of nodes in the underlay and overlay, to be responsible for different context and management aggregation processes. Existing approaches rely on real-time negotiation(s) for node selection. However; real-time negotiations are expensive for bandwidth-limited environments such as ANs. A novel AN Virtual Management Backbones (VMBs) was developed. Each VMB has a dedicated (distributed) context aggregation task, and consist of a set of dynamically selected VMB nodes, where aggregation is carried out. The selection of VMB nodes is conducted without heavy-weighted real-time negotiations. Our solution achieves this goal by utilising the scalability and dynamicity advantages of Distributed Hash Tales (DHTs).
- *Dynamic ACS Orchestration Engines (D4)* – ACS orchestration describes: i. required patterns of interaction between FEs, and templates for sequences of interactions (i.e. orchestration scripts /policies); ii. control and data-flow using processing steps with activities, sequencing rules/policies, flows with data and/or control links; iii. execution of the order of the interactions between FEs. iv. execution of task-driven orchestrations (e.g. management tasks, mobility tasks, security tasks, connectivity tasks, etc.). Dynamic Orchestration Engine is an intelligent intermediary between ACS FEs that alters the flow of orchestration based on run time information and it executes task-driven orchestration scripts. ACS may require multiple types of orchestration engines, which are task -optimised. We envisage orchestration engines, which are management-task optimised. The management tasks include: initialisation, dynamic reconfiguration, adaptation and contextualisation, FCAPS functionality, dynamic service deployment support, optimisation, organisation of ACF FEs. The management orchestration is for further study in year 2.

The interaction with other work packages (WPs) has been driven by a number of related topics being addressed separately in different WPs. Links with some WPs have been stronger than with others and more details about it can be found in the Common milestone report WP-D/E/F on cross layer design, interactions, optimisations and enhancements. Worth mentioning here the collaboration with WP-B leading to a number of publications and to a standard contribution on a Network Information Service Infrastructure, as well as the collaboration with WP-F on making the SATOs context-aware. By overlay context-awareness, we refer to the capability of a SATO to use network context information for self-adaptation, or in the provision of services.

Interactions with WP-A and contribution to the AN System Description (SD) report were performed for all quarterly SD releases.

Interactions with WP-H for the AN prototype requirements and S/W licence were made. The following first prototypes of the ContexWare & ConCoord systems and the

Management of Composition have been built and integrated into the common Ambient Network prototype. In addition DINA platform (i.e. dynamic management adaptation and dynamic deployment, control and management of functionality) and XCAM Policy Wrapper have being developed for FreeBSD and made available for WPs use.

Collaboration with the other WWI projects SPICE, E2R2, WINNER, MOBILLIFE on the integration of the main context-aware components part of an overall WWI System architecture resulted in joint paper presented at the World Wireless Research Forum in Nov 2006.

Papers

Detailed WP-D results were presented in the following 19 published/accepted for publication papers and 1 standard contribution (i.e. 7 additional submitted papers are currently under review):

Title	Forum and document id	Date	Contributing partners
Presentation in the IEEE 802.21 meeting in Melbourne	http://www.ieee802.org/21/doctree/2006-09_meeting_docs/21-06-0762-00-0000-Ambient-Networks.ppt	Sep 06	RMR, BT

Title	Where published	Date	Contributing partners
"An Information Service Infrastructure for Ambient Network"	IASTED International Conference on Parallel and Distributed Computing and Networks – PDCN 2007, Innsbruck, Austria, http://www.iasted.org/conferences/home-551.html	Feb 07	BT, VTT, UC, SIEMENC, UCL
PACMAN: A Policy-Based Architecture for Context Management in Ambient Networks	IEEE CONSUMER COMMUNICATIONS And NETWORKING CONFERENCE (IEEE CCNC'2007) Las Vegas, USA,	Jan 07	SITE
Context Provisioning in the WWI System Architecture	Wireless World Research Forum-WWRF17; Heidelberg, Germany; www.wireless-world-research.org/meetings/WWRF17/WWRF17-default.php	Nov. 06	NOKIA, KCL, INFO, UCL
"Context Dissemination and Aggregation for Ambient Networks. Jini Based Prototype"	EuroSSC 2006- 1st European Conference on Smart Sensing and Context; Enschede, The Netherlands http://wwwes.cs.utwente.nl/smartsurroundings/eurossc/	Oct. 06	AGH
Towards a Context Monitoring System for Ambient Networks	IEEE CHINACOM 2006 International Conference on Communications and Networking in China , Beijing; China www.chinacom.org	Oct 06	UCL, KTH
Adaptive Continuous Monitoring in Large-scale Networks with Accuracy Objectives	Fourth Swedish National Computer Networking Workshop (SNCNW 2006), Luleå, Sweden; http://www.sncnw.se/	Oct 06	KTH
Distributed Hash Tables Composition in Ambient Networks: A Synthesis Study	IEEE DSOM'06, 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management – Large Scale Management; Dublin, Ireland	Oct 06	UCL, BME, BT

Title	Where published	Date	Contributing partners
	www.manweek2006.org/dsom/dsom.php		
Adaptive Distributed Monitoring with Accuracy Objectives	ACM SIGCOMM workshop on Internet Network Management (INM 06), Pisa, Italy http://research.microsoft.com/%7Edmaltz/inm06/	Sept 06	KTH
Towards Flexible Service-aware Adaptation Policy Management in Ambient Networks	IEEE International Conference on Networks 2005, Singapore, http://www.sp.edu.sg/icon2006/	Sept 06	UCL
A Policy Management System for Ambient Networks	London Communications Symposium (LCS) 2006 – London, United Kingdom; http://www.ee.ucl.ac.uk/lcs/	Sept 06	UCL
Contextualisation of Management Overlays in Ambient Networks	IEEE International Multi-Conference on Computing in the Global Information Technology, ICCGI'06, Bucharest, Romania	August 06	UCL
Towards Ambient Networks	.IEEE International Multi-Conference on Computing in the Global Information Technology, ICCGI'06, Bucharest, Romania	August 06	UCL
Towards Context-Based Flow Classification	International Conference on Autonomic and Autonomous Systems (IEEE ICAS'06)- http://www.iaria.org/conferences/ICSA06.html	July 06	UCL
Peer-to-Peer Management in Ambient Networks	IST Mobile and Wireless Communications Summit, Mykonos, Greece, http://mobilesummit2006.org/ms2006/servlet/org.nkpap.visualizer.Main	June 06	BME, UCL
Self-Management of Wireless Base Stations.	IST Mobile and Wireless Communications Summit, Mykonos, Greece, http://mobilesummit2006.org/ms2006/servlet/org.nkpap.visualizer.Main	June 06	NEC
Controlling and Managing Compositions in Ambient Networks	IST Mobile and Wireless Communications Summit, Mykonos, Greece, http://mobilesummit2006.org/ms2006/servlet/org.nkpap.visualizer.Main	June 06	BME, NOKIA, ERICSSON
Pattern-based Network Management within Ambient Networks	IST Mobile and Wireless Communications Summit, Mykonos, Greece, http://mobilesummit2006.org/ms2006/servlet/org.nkpap.visualizer.Main	June 06	ERICSSON, BME, NEC, KTH
"Distributed Real-time Monitoring with Accuracy Objectives"	IFIP Networking 2006, Coimbra, Portugal; http://networking2006.dei.uc.pt/	May 06	KTH

Title	Where published	Date	Contributing partners
Requirements for Policy Framework for Ambient Networks	Wireless World Research Forum-WWRF16, Shanhai; China; http://www.wireless-world-research.org/meetings/WWRF16/	April 06	ERICSSON, UCL, BT

Submitted papers under review -Title	Submitted to	Submission Date	Contributing partners
"A-GAP: An Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives"	IEEE eTransactions on Network and Service Management (eTNSM)	Sept 06	KTH
Virtual Structures for Self-Organisation in Ambient Networks	IEEE e-TNSM (eTNSM) Special Issue of Self-Managed Networks, Systems and Services	Sept 06	UCL, FT
Design and Evaluation of Distributed Selfconfiguring Load-Balancing	10th Symposium on Integrated Network Management, Munich 2007	Oct 06	NEC
Virtual Private Ambient Networks	IEEE Communications Magazine Featue Issue on "Virtual Private Networks"	Oct 06	UCL, FT, NEC, ERICSSON
Towards Management of Service-aware Adaptive Transport Overlays	IEEE Communications Magazine – Network & Service Management Series	Oct 06	UCL, FT,
Stochastic maintenance of routing in structured P2P systems	Elsevier Journal of Computer Communications on Foundations of Peer-to-Peer Computing	Oct 06	BME, UCL
An Information Service Infrastructure for Dynamic Networks	IEEE ICCGI 2007 International Multi-Conference on Computing in the Global Information Technology	Nov 07	VTT, BT, UC, SIEMENS, UCL

1 Introduction

The future wireless world will be filled by a multitude of user devices interconnected with each other through networks like HANs, BANs, PANs, etc., and wireless technologies. Simple-to-use, anytime-anywhere network access affordable for everyone is an undisputed must for the eSociety where the user will expect a rich set of communication services independently of the access used. Devising technologies for true Plug-and-Play networking, efficient use of all infrastructure investments and access competition are the key technical challenges to achieve this vision.

The Ambient Networks project is addressing these challenges by developing innovative mobile network solutions for increased competition and cooperation in an environment with a multitude of access technologies, network operators and business actors. It offers a complete, coherent wireless network solution based on dynamic composition of networks that provide access to any network through the instant establishment of inter-network agreements. The concept offers common control functions to a wide range of different applications and access technologies, enabling the integrated, scalable and transparent control of network capabilities.

The project is currently in its second phase. Phase 1 has established the overall approach and developed innovative technical concepts. Phase 2 will prove their viability through implementation, integration, measurements and performance evaluation, enabling concurrent standardisation.

The vision for Ambient Networks accounts for heterogeneous networks composing and cooperating, on demand and transparently, without the need for manual configuration or offline negotiations between network operators. To achieve these goals, ambient network management must become dynamic, distributed, self-managing, self-policing and autonomously responsive to the network and its ambience.

1.1 AN ManagementWare Model

Some of the characteristics of Ambient Networks are the high level of heterogeneity and dynamicity in composition, in combination with the introduction of multitude at all levels: we will have multiple different link technologies to choose from, we will have multiple different network technologies, we will have multiple different networks that will compose and decompose, we will have multiple different operators and service providers that will cooperate and compete in a dynamic fashion, we will have a multitude of different applications that will run on a multitude of different terminals with different capabilities. All this provides the user with the best connectivity possible based on the users' requirements at that time and the current capabilities of the networks available.

In order to manage networks and network nodes in such an environment, significant changes have to be made to the management paradigms of today. This will not be feasible using a centralised, hierarchical solution with a high degree of hands-on control of networks and nodes. Instead new network management systems are required. These must be distributed over the different networks, highly autonomous and self-managing, and only require interaction from operators and specialists if the system itself cannot resolve conflicts.

Furthermore, this interaction should be as intuitive, user-friendly and unambiguous as possible to minimise possible errors and time spent on the problem.

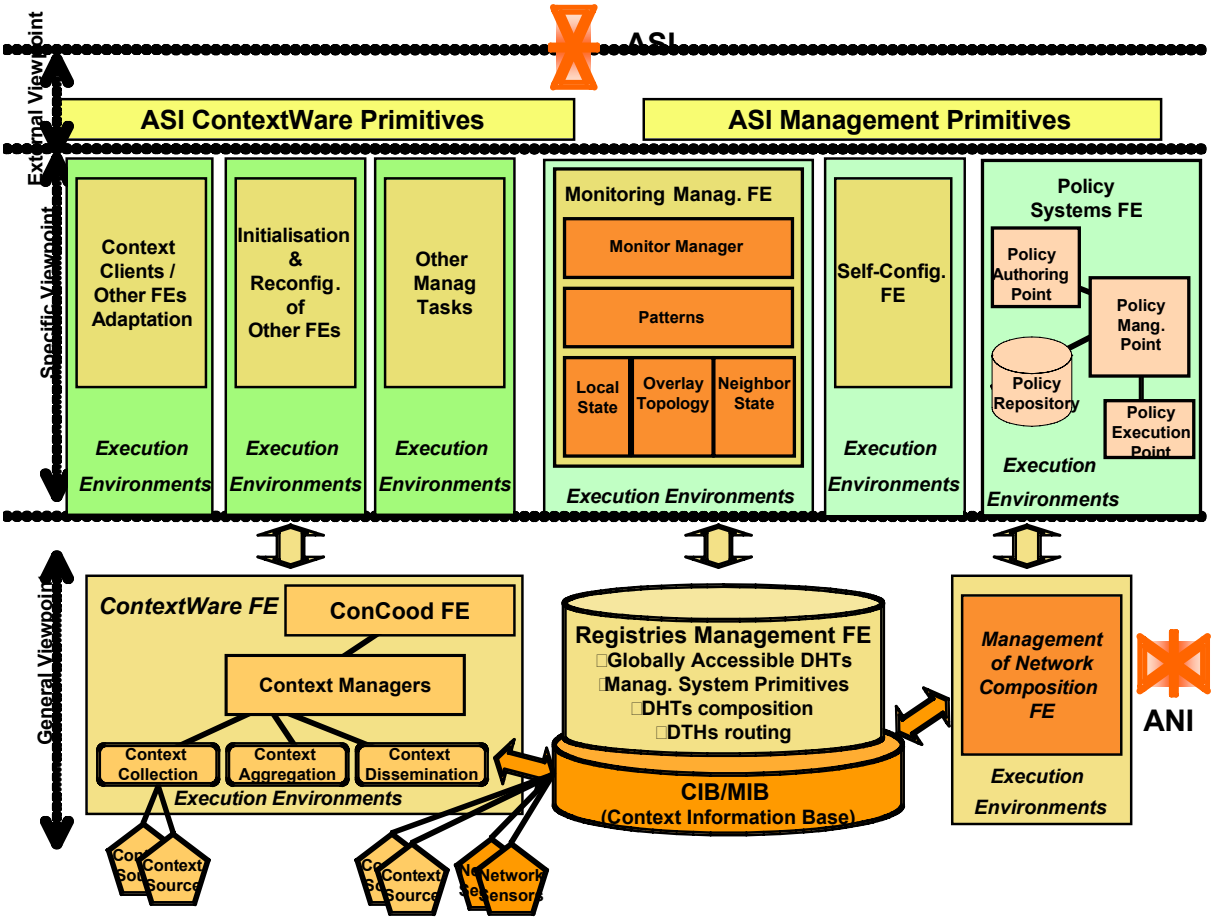


Figure 3 – AN ManagementWare Model

We have identified and analysed a number of AN Management systems, which would enable the integrated management for Ambient Networks. We have also identified the network management sub-systems and their interactions as presented in chapters 2.8 and 4, The proposed integrated solutions must be tested and verified as valid building blocks in an AN management system. The Figure 3 depicts the AN ManagementWare model, which underpins the WP-D design and implementation work.

Management application functionality is depicted at multiple viewpoints (see Figure 3):

- AN Management General Viewpoint
- AN Management Specific Viewpoint and
- AN Management External Use Viewpoint.

At the *AN Management General Viewpoint* we envisage the management aspects, which are used by all specific AN management sub- systems and their users. It includes ContextWare FE, Composition Management FE and Registries Management FE.

At the *AN Management Specific Viewpoint* we envisage the management systems which exhibit FCAPS (Fault, Configuration, Accounting, Performance and Security Management) functionality for the AN. It includes Decentralized Real-time Monitoring FE and Policy Managements Systems FE.

At the *AN Management External Use Viewpoint* we envisage the ASI management primitives allowing a service client to request reservation and configuration of AN management resources.

An other viewpoint of the Management FEs is described in the following figure.

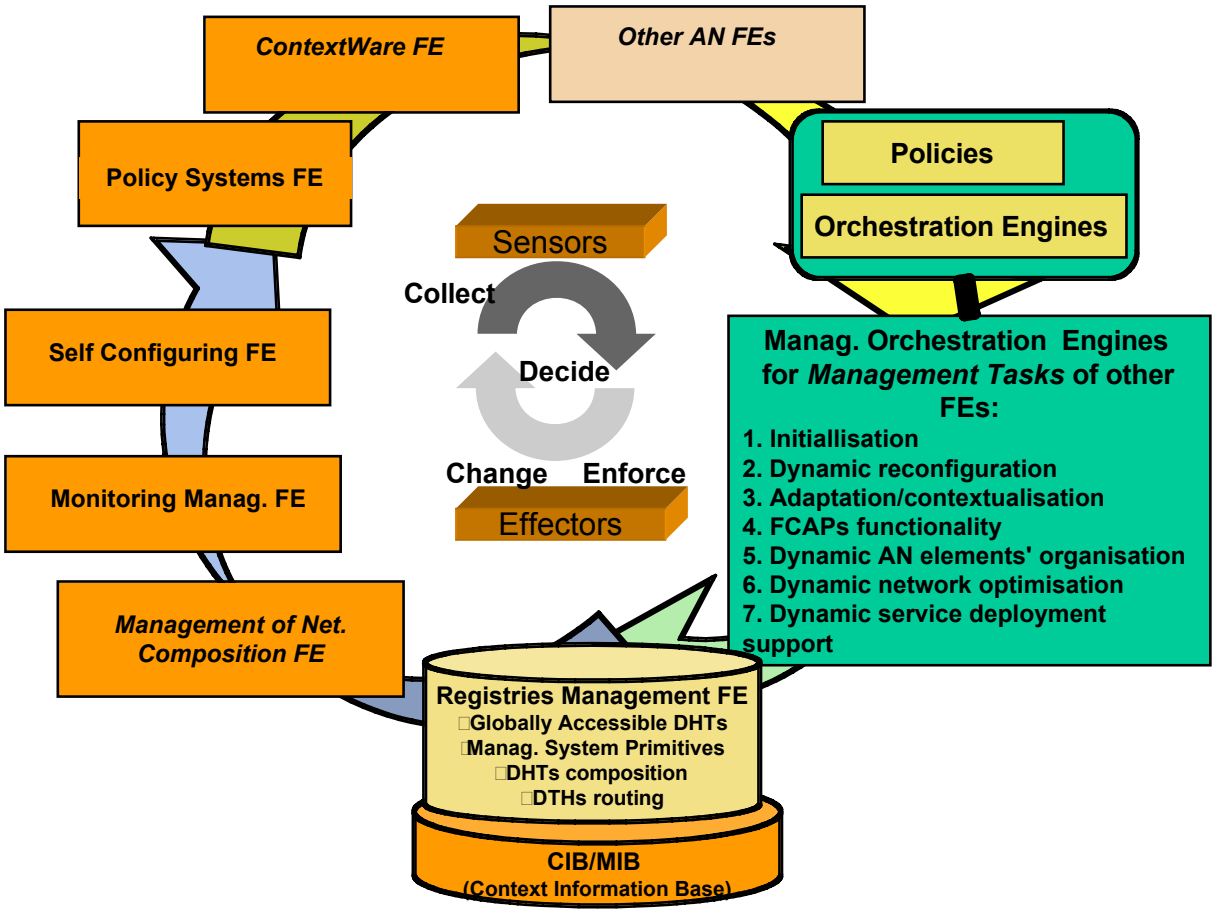


Figure 4 – Orchestration view point

ACS orchestration describes:

- required patterns of interaction between FEs, and templates for sequences of interactions (i.e. orchestration scripts /policies)
- control and data-flow using processing steps with activities, sequencing rules/policies, flows with data and/or control links.
- execution of the order of the interactions between FEs.
- execution of task-driven orchestrations (e.g. management tasks, mobility tasks, security tasks, connectivity tasks, etc.)

Orchestration constructs are: sequence, repetition, conditional, fork, and-join, or-join concurrent execution, multiple instances, composite tasks, removal of tokens, directly connected transitions, signalling fault(s), doing nothing, templates for sequences of interactions, scripts (i.e. aggregation of orchestration constructs).

Dynamic Orchestration Engine is an intelligent intermediary between ACS FEs that alters the flow of orchestration based on run time information and it executes task-driven orchestration scripts. ACS may require multiple types of orchestration engines, which are task -optimised. We envisage orchestration engines, which are management-task optimised. The management tasks include: initialisation, dynamic reconfiguration, adaptation and contextualisation, FCAPS functionality, dynamic service deployment support, optimisation, organisation of ACF FEs. The management orchestration is for further study.

Different AN nodes may include different management functionality. As such the deployment of minimum or optimum management functionality per AN node is a subject for further study.

1.2 AN Autonomic Management Characteristics

This section describes the expected autonomic features of the AN Management.

AN Management Systems first main characteristic is that they manages complexity, continuously tune themselves, adapt to unpredictable conditions, prevent and recover from failures and provide a safe environment.

The autonomic nervous system frees our conscious mind from self-management and is the fundamental point of AN Management Systems thus 'freeing' up system administrators and normal users from the details of system operation and maintenance. If a program can deal with these aspects during normal operation, it is a lot closer to providing users with a machine what runs 24x7 and its optimal performance. The autonomic management system will change anything necessary so as to keep running at optimum performance, in the face of changing workloads, demands and any other external conditions it faces. It should be able to cope with software and or hardware failures whether they are due to an unforeseen incident or malicious acts.

Installing and configuring systems can be extremely time consuming, complex and can be open to human error no matter how qualified the administrator is. AN Management Systems could configure themselves automatically by incorporate new components seamlessly.

Current networking systems may contain large amounts of different variables/options/ parameters which a user is age to change to optimize performance. Few people however know how to use these and even fewer know how to get them exactly right to get 100% performance. An autonomic management system could continually monitor and seek ways of improving the operation efficiency of the systems in both performance and/or cost. It is faster at this than a person and is able to dedicate more time to finding ways of improving performance.

The characteristics stated above all come to together to help a system run more efficiently while reducing costs due to less human input.

The second main characteristic of AN Management Systems is that they exhibit self-knowledge and self-awareness properties, in particular self-contextualisation, self-programmability and self-management (i.e. self - optimisation; - organisation; - configuration; -adaptation; - contextualisation, -healing; - protection).

Self-Contextualisation - Context is any information that can be used to characterise the situation of an entity (a person or object) that is considered relevant to the interaction between a user and an application. A context-aware system is capable of using context information ensuring it successfully performs its expected role, and also maximises the perceived benefits of its use. Nevertheless, this is a user-centric view and reflects the fact that most research on context and context-awareness up to now has been focused on "user context". In contrast a new generation of system give context a much broader scope and renders it universally accessible as a basic commodity provided and used by the network. In this way context becomes a decisive factor in the success of future autonomous systems adaptive to changing conditions. As such contextualisation is a service/software property. Self-contextualization is the ability of a system to describe, use and adapt its behaviours to its context meanwhile it does not have to be aware of any other form of context knowledge. However, a context aware system is a system that acts based on knowledge of a certain context. Self-contextualisation means that a service/software component autonomously becomes context-aware. Context for supporting system/software

components should be made available, so that multiple service/software components may take advantage of the available network and application context. Once a service/software component becomes context-aware it can make use of context information for other self-management tasks that depend on context information.

Self-Programmability - Recent research on distributed systems technologies has focused on making service networks programmable. The objectives of programmable system networks are to take advantage of network processing resources and to promote new service models allowing new business models to be supported. The resulting service models do not, however, target development of services, but, rather, their deployment. Dynamic system programming applies to executable service code that is injected into the autonomic systems elements to create the new functionality at run-time. The basic idea is to enable third parties (users, operators, and service providers) to inject application-specific services into the Autonomic Systems platform. Applications and services are thus able to utilize required network support in terms of optimised network resources and as such they can be said to be network-aware i.e. a service-driven network. Self-programmability means that network programmability is following autonomous flows of control triggered and moderated by network events or changes in network context. The service network is self-organised in the sense that it autonomically monitors available context and provides the required context and any other necessary network service support to the requested services, and self-adapt when context changes.

Self-Management - Management is an essential topic when dealing with utilisation of network and application context information for supporting services and contextualised services. Managing network context from the perspectives of the context information provider means dealing with a number of processes related to manipulation of network context information. Self- management could be characterised by a set of interrelated and orchestrated capabilities, which are exhibited at a component and system levels. Theses capabilities include self - organisation -optimisation, -, -configuration, -adaptation, -healing, - protection.

- Self-organisation – Components, context information and resources are distributed across heterogeneous networking systems. In order for services to make use of these distributed information and resources, they must be structured or referenced in an easy-to-access-and-retrieve structure in an automatic fashion. All these context information and resources must be autonomously organised and reserved through a service layer. The autonomous and dynamic structuring of components, context information and resources is the essential work of self-organisation.
- Self-optimisation - In the large and heterogeneous network context information and resources and their availability are rapidly changing. There is a need for an autonomous mechanism for consistent use and monitoring of components, control of context information and resources, so that networking functions and services may be executed or activated in the most optimised fashion. Autonomic systems must seek to improve their operation every time. They must identify opportunities to make themselves more efficient from the point of view of strategic policies (performance, cost, etc).
- Self-configuration/Self-adaptation - It enables autonomous structuring of component information and resources, making them available to clients. Autonomic systems must configure themselves in accordance with high-level policies representing service agreements or business objectives, rules and events. When a component or a service is introduced, the system will incorporate it seamlessly, and the rest of the system will adapt to its presence. In the case of components, they will register themselves and other components will be able of use it or modify their behaviour to fit the new situation.

- Self-healing - Autonomic systems will detect, diagnose and repair problems caused by system failures. Using knowledge about the system configuration, a problem-diagnosis embedded intelligence would analyse the monitored information. Then, the services would use its diagnosis to identify and enforce solutions or alert a human in the case of no solutions available.
- Self-protection - There are two ways in which an autonomic management system must self-protect. It must defend itself as a whole by reacting to, or anticipating, large-scale correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures.
- The realisation of self-awareness properties at a component and system level revolves around increasing the level of automation of the intelligent control loop described as ‘Collect-Decide-Enforce’.
- ‘Collect’ is about monitoring that allows constructing a picture about the surrounding environment in order to build self-awareness
- ‘Decide’ involves inference and planning. The former refers to a process whereby the problem is diagnosed based on the collected information while the latter refers to the process whereby a solution is selected.
- ‘Enforce’ comprises deployment, which adds functionality by means of new components, and configuration, which changes the existing functionality by means of programmability of services.

The realization of this intelligent loop eventually leads to a distributed, adaptive, global evolvable system capable of fostering continuous changes as it depicted in the Figure 1.

2 Integrated ManagementWare Functional Entities for ANs

This chapter introduces the integrated ManagementWare functional entities developed by WP-D workpackage.

The first two FEs introduced are part of the ContextWare architecture, designed and built to manage network-context information. More precisely, ContextWare has the role of managing the context information in the Ambient Control Space, including its distribution to context clients/consumers. Context clients are context-aware services, either user-facing applications/services or network services, which make use of or/and adapt themselves to context information. The main building blocks are the *ConCoord FE* mapping context identifiers to storage locations and the *Context Management FE* made up by a number of different and dynamically created Context Managers that carry out network context information management tasks such as data-aggregation, filtering, dissemination, replication etc.

Self-configuration FE - All configuration tasks in AN are performed in a decentralized way without a general component that handles self-configuration in the ACS. Different configuration tasks are performed independently by different FEs. On the other hand, decentralized configuration requires in general some mechanisms to accomplish correctly their configuration tasks. Stability and oscillations avoidance are the major issues that must be tacked in a distributed scenario. The *Self-configuration FE* provides general services to other FEs to correctly accomplish their configuration tasks. Its main purpose, currently investigated, is to guarantee stability of a certain configuration over time and to avoid oscillations.

Real-time Monitoring FE – It address the problem of continuous monitoring with accuracy objectives for large-scale network environments. It is based on an efficient aggregation protocol (A-GAP) that allows control of the accuracy of the estimation. A-GAP allows for continuously computing aggregates of local variables by (i) creating and maintaining a self-stabilizing spanning tree and (ii) incrementally aggregating the variables along the tree.

The *Composition Management FE* manages the logical structure of the ACS as well as FE instances in this structure. This logical structure is created through network composition processes. In case of network integration composition type, two ACSs are merged into one. In case of control sharing composition type, the original ACSs are kept but a higher level ACS is created which is populated by functional entities having negotiated common management in the composed AN. The Composition Management FE is responsible for performing the above changes during the realization phase of network composition. This includes the modification of the logical structure of the ACS and the creation, deletion or modification of FE instances accordingly.

Policy Management FE is a policy framework for Ambient Networks. Its components are: a Policy language (syntax & semantics), Policy editor (Policy Authoring Point (PAP), Attribute Authoring Point (AAP)); Policy repository; Policy conflict detection/resolution; Policy evaluation and decision (Policy Decision Point (PDP)); Policy enforcement (Policy Enforcement Point (PEP)); Policy violation compensation

Registries Management FE -. The goal of a common AN distributed registry is to exploit synergies and provide a unified distributed registry functionality for all functional entities as a number of functional entities need registries to store and lookup various entities: UCIs in the ConCoord FE, policy data in the Policy FE, NID router locators in the Node ID Management FE, etc. Another important objective is to achieve scalability, robustness and fault tolerance at low maintenance overhead even in dynamic network environments. While scalability and robustness is achieved through the use P2P technology, fault tolerance and low maintenance overhead is ensured by bypassing the storage subsystem whenever possible. The registry interface offers primitives to register and lookup both permanent data

and node related entities while the registry management interface is used to notify the registry about composition and decomposition events.

2.1 Context Coordinator FE

The ConCoord is a distributed registry that maps Universal Context Identifiers (UCIs) to the location of context information objects. The ConCoord maintains this registry by receiving REGISTER requests from context sources (entities providing one or more context objects). Hence, context sources actively disseminate pointers to their context objects to the ConCoord. The ConCoord FE is the first point-of contact for context clients. When Context clients want to access context information, they send RESOLVE requests which contain one or more UCIs to the ConCoord. The ConCoord then responds by returning the locations of the corresponding context objects. Universal Context Identifiers (UCIs) are a new type of Uniform Resource Identifiers (URI) [36]. UCIs uniquely identify a given context object, but not its location within the network. Then clients contact the located context sources directly to GET the information, or to SUSBSCRIBE to context change events by receiving NOTIFICATIONs. If a context source no longer wants to be registered with the ConCoord it removes its entry in the ConCoord by sending a DEREGISTER request. The REGISTER, DEREGISTER and RESOLVE primitives with which the ConCoord communicates with the context sources and clients is a SIMPLE-inspired protocol.

The rationale for this design is to leave context information, which might change frequently, at the source, until it is actually requested by some client. A source registers an object only once, and a client resolves each object UCI only once. Any further interaction is then done between clients and sources directly.

The registry of the ConCoord is itself a context object, the meta-context object of all other context objects. The context source for this object is the ConCoord itself, and the meta-object is essentially the set of registered UCIs. This meta-object should also be accessible like any other context object by the ContextWare protocol primitives, as this enables context clients to subscribe to events like "notify me whenever a new object of this type registers". This is important for context clients to detect new sources of context information, or to learn that currently used context sources are no longer available.

The functions of the ConCoord can be summarised as follows:

- A distributed registry where a context source registers the UCIs of its context objects with its contact information.
- A function to authenticate and authorize source registrations, and client access to context objects.
- A function to resolve the UCIs required by context clients to the stored locations of context objects.
- A function to remove stored UCI location mappings.

The ConCoord's implementation is based on a Distributed Hash Table (DHT) [39]. The interface to the ConCoord is a Web Services Interface. The authorisation and access control in ConCoords is discussed as part of the ContextWare security framework in section 4.4. Locating the ConCoords, DHTs, the Web Services interface as well as operational and implementation details are discussed below.

2.1.1 Context Exchange Protocol

ContextWare is a SIMPLE-inspired protocol that employs Universal Context Identifier (UCI)to locate context. As we have seen above, the protocol contains six primitives:

[DE]REGISTER
With this message a context source will register 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.

In our distributed approach to ContextWare, we assume the existing of the following entities:

- Context Coordinator

The context coordinator will be distributed within the DHT overlay and handle 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 (CUA) 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 will use a CUA in order to register and resolve context information in the DHT overlay. A context manager will aggregate 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.

A context user agent can act as both a producer and / or as a consumer of context information, most likely it will act as a consumer and producer simultaneously. An application or service that uses the Ambient Service Interface (ASI) communicates with the CUA by resolving a UCI to the location of the CUA. The CUA then uses the DHT overlay to locate (in the case of a consumer) the context information. The Ambient Resource Interface (ARI) is used when an application or service wants to register (in the case of a source) a UCI with the help of a CUA.

The context user agent also acts as a node in a DHT overlay; logically a CUA will be treated as the rendezvous point for Context Information Objects by the applications and services requesting or registering context information (i.e. an application or service will use the messages of the ContextWare protocol to interact with the CUA which then will forward the requests to the DHT overlay.

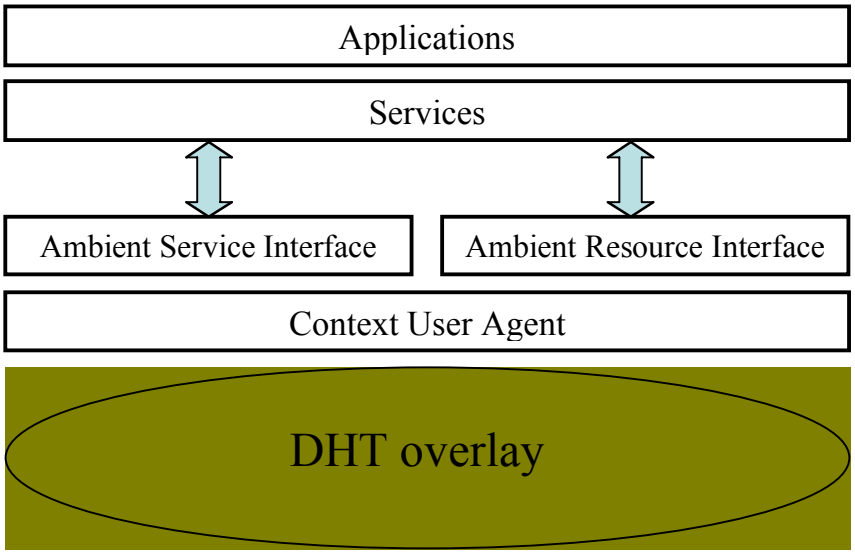


Figure 5 – Context User Agent Environment

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 will be described in more detail in following sections.

The Figure 5 above depicts the relations between the Context User Agent, DHT Overlay and Ambient Networks interfaces that were discussed above.

2.1.2 Context Exchange Protocol Design

This subsection summarizes the design of the prototype by defining a Distributed Context Exchange Protocol and its messages.

Transport protocol — The DHT overlay which helps the ContextWare to resolve, store, and register CUAs will be based on top of a UDP transport layer. Messages (XML) in ContextWare are mostly single notify message or subscribe message. The protocol design includes reliability mechanisms to insure acknowledgement of messages and retransmission of lost messages.

Creating an overlay

When a new node tries to join a DHT overlay ring 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 a new DHT overlay. If the node receives a reply, then it will join this existing overlay, by sending a request to join to the responding node. The responding node will grant, redirect, or refuse the joining node based upon its preferences and policies. If a node is refused to join a overlay it will not be able to register and resolve any context information.

The discover response message will contain 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 will join the network which equals the name of the ambient network the node intends to join.

Registering in a DHT overlay

Before a CUA can accept registrations and resolve 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 will look like the following; see the figure below. In this scenario there exists an overlay which nodes 1, 4, and 6 participates in, node 2 is the joining node which will start the discover procedure.

- Discover of overlay

A discover procedure starts when the new CUA broadcasts a DISCOVER message. The CUA will receive 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 could be received; the joining node will send its REGISTER message to the node which represents the overlay it wants to join.

- Registration in overlay

The local CUA will try to register 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). The figure below 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 will tell which overlay the joining node should join together with its calculated nodeID.

- Registration response

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

- Updating the overlay

When a joining node receives a REGISTER_RESPONSE message confirming the registration it will have to tell its new neighbour 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.

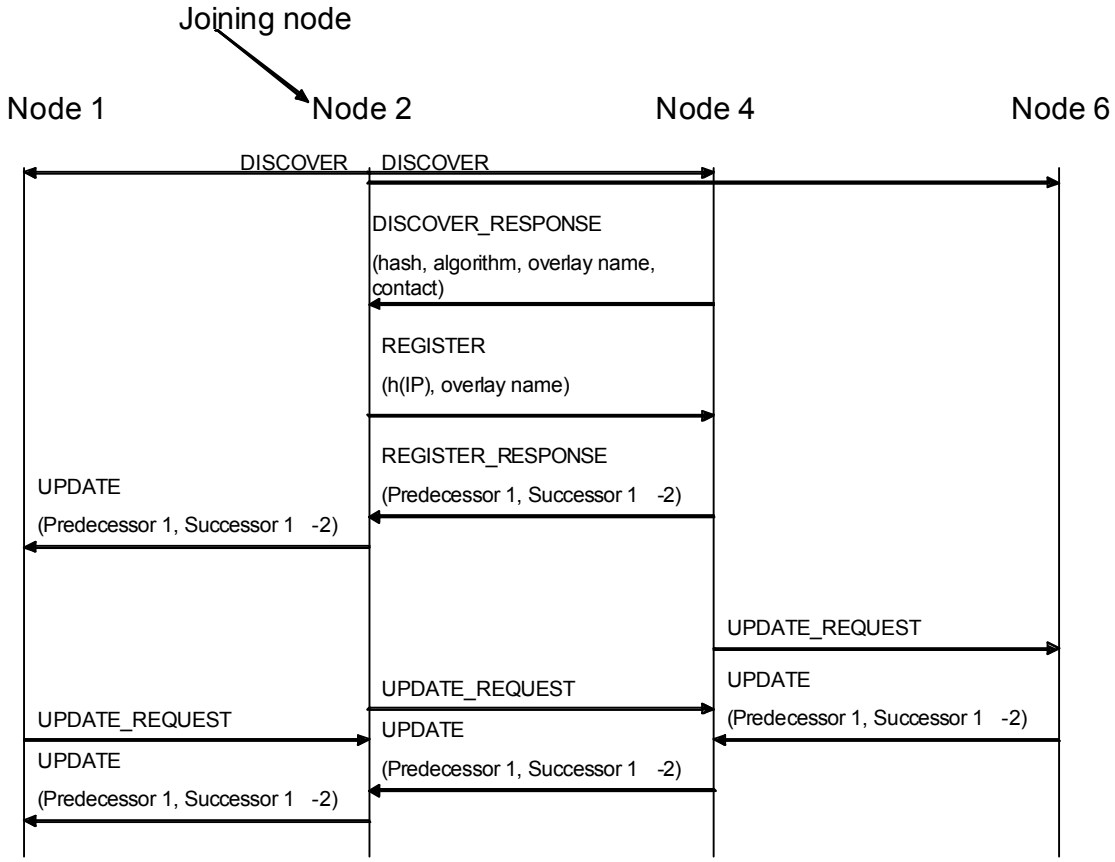


Figure 6 – A CUA registering in a DHT overlay

The registration process is completed after the UPDATE message is sent to the predecessor. The periodic UPDATE_REQUEST messages will be 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.

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 will be able to register its context UCIs within the overlay via a local CUA. The local CUA will receive a register request from a source, then it will register the UCI with the source's data information at the corresponding node in the P2P overlay (i.e. this message will be sent on behalf of a request from an application or service).

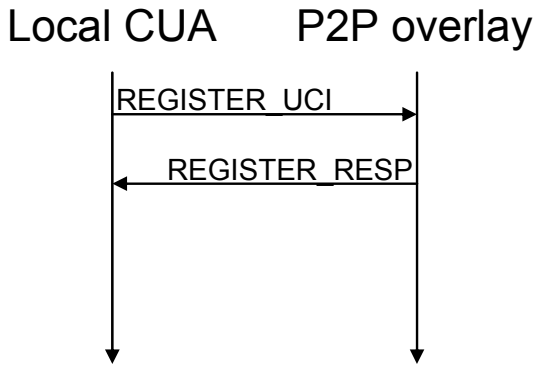


Figure 7 – A CUA registering a UCI

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 will be redirected to another more appropriate node. The local CUA in addition to registering the UCI within the corresponding node will also concatenate 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.

Resolving a UCI in DHT overlay

The DHT overlay will work much like a lookup service for the different resources. In this case the DHT overlay will store 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 will send a message to resolve the UCI. This message is routed as described above using a DHT algorithm. If a UCI can not be found, then the local CUA will try 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 will be 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 will prevent a single node from tampering with the registered data.

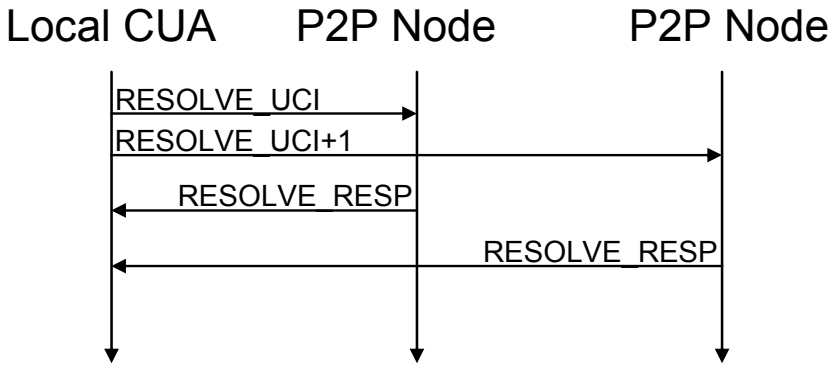


Figure 8 – A CUA resolving a UCI together with a replica

Protocol Definition

A context user agent will act as middleware between context clients and sources with the help of a DHT overlay in order to either register or 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.

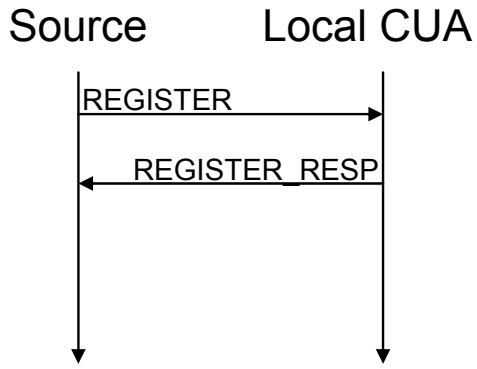


Figure 9 - A source registers its UCIs

Registering context information

When a source wants to publish context information it registers its context information in the DHT overlay. 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 later in this section. 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.

- *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 will also have to send the context information to the delegate. The context information is then stored within the DHT overlay together with the UCI.

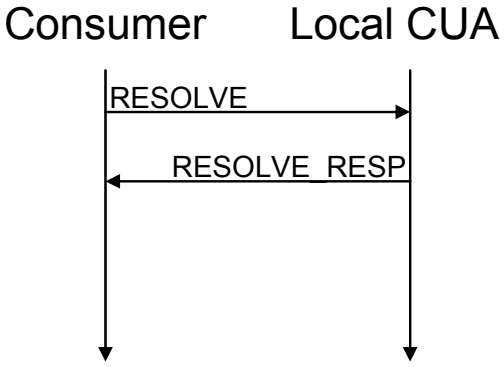


Figure 10 – A context client resolves a UCI

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 will do 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.

Get/Subscribe to specific context information

When a CUA has received the contact information from a lookup operation it will return 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 Figure 10 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.

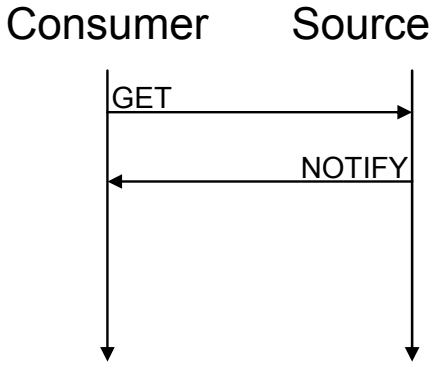
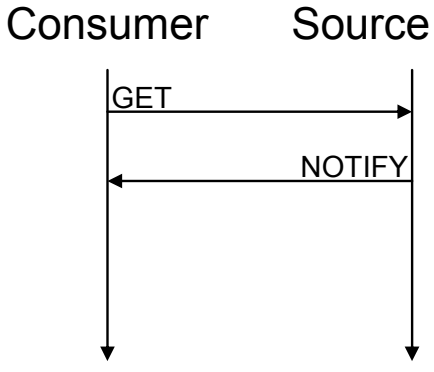


Figure 11 – A context client retrieves context information once



The other

Figure 11Figure 11 is used when a consumer wants to retrieve the context information when the information is updated. Figure 12 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 will minimize the traffic as the requesting node will not have to periodically poll the source for updates, instead the source will notify the subscribing consumer when context information is updated.

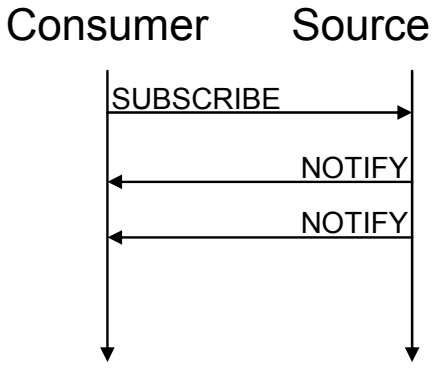


Figure 12 – A context client subscribe to context information

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 will end.

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 will be based upon the policies the network owners have established. Network 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 will be handled by the Composition management functional entity within each of the two networks' ACS. The ContextWare, specifically the Context Coordinator, will get information about the composition and can subsequently negotiate how to compose the two ContextWare networks.

2.1.2.1 ContextWare Composition

In AN phase I 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. This gateway approach could also be used to solve the composition problem within a DHT overlay. Another method will be absorption or a 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.

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.

2.1.2.2 Gateway (de-)composition

A gateway will act as middleware between the two DHT overlays. This approach could be 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 will 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, 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 will still be 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 will be some additional messages, as the newly composed overlay will form and messages have 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 will simply be the other and visa versa.

Figure 13 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.

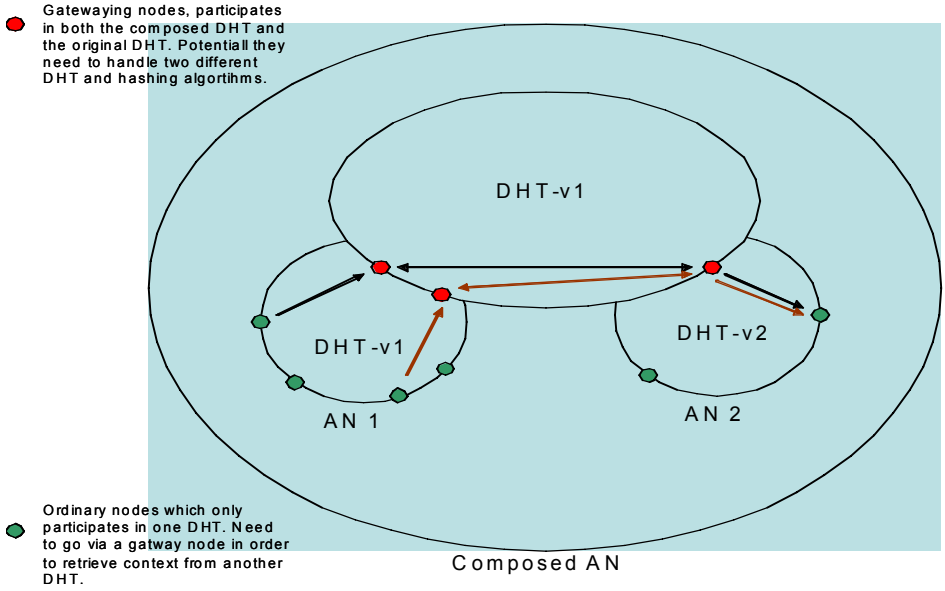


Figure 13 – Abstract gateway scenario

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 will confirm the set up with a COMP_REQ_RESPONSE message saying OK.

Node A2 and B1 will 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 will create a single node of failure) from each domain up to a large number of nodes.

If a node within AN A needs something from AN B it will contact one of the nodes on A's side which can redirect the query to a node in AN B.

The combined overlay will only handle keys with UCIs of the composed network. All the original keys will still be stored in the original two DHT overlays.

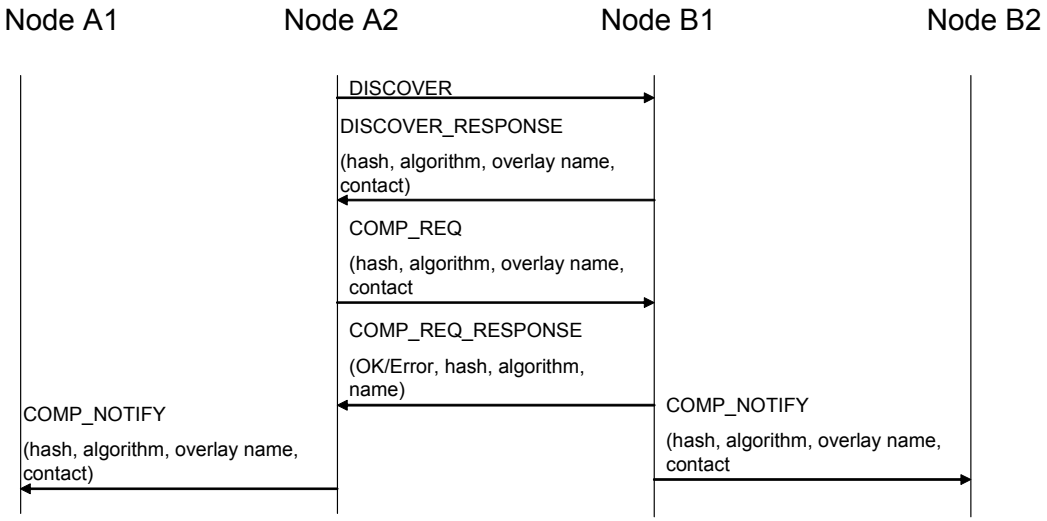


Figure 14 – Gateway composition scenario

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 will cease to exist and the two original DHTs should work without knowledge of each other (see Figure 15).

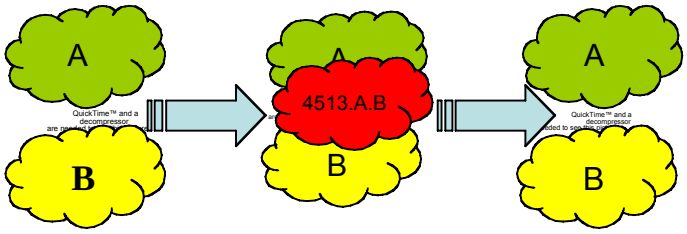


Figure 15 - Gateway composition of two ContextWares

Absorbtion (de-)composition

In order for a full compose, or absorption, there is need for another method more efficient than gatewaying when handling 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 will be the same as in gateway scenario; thus the discover procedure and negotiation will be 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. When they have agreed on the preferences for the common overlay it is 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 will grow as more and more of the selected nodes join. The goal is to form a completely new overlay with all keys registered in this new overlay without depending upon the two original overlays.

The new overlay will initially only function as a distributed gateway between the two networks. If a node within AN A needs something from AN B it will contact one of the nodes on A's side which will redirect the query to a node in AN B. The combined overlay will only handle keys with UCIs of the composed domain.

Sequentially will nodes that are not selected at the beginning join the composed overlay. Once these nodes have joined the composed overlay, then all new registrations and re-registrations will be made in this new overlay as well.

While forming the new composed overlay context can be used as the two original overlays still exist together with a distributed gateway. These previous overlays will exist until all nodes have left each AN and all keys are timed out.

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.

Locating the ConCoord

Within an Ambient Network, both sources and clients have to locate the local ConCoord node in order to either register their context objects or to resolve UCIs, respectively. In addition, any ConCoord must be able to locate any other ConCoord for inter-domain resolution. As will be discussed below the interface through which the ConCoord is accessed in a Web Services interface. Hence, the ConCoord appears as a Web Service to the outside world exposing the register, resolve and deRegister methods and is accessible through a URL. Each host in an Ambient Network runs its own local ConCoord node and hence the ConCoord Web Services interface is always accessible via the URL http://127.0.0.1:8080/axis/services/ConCoord. However, the ConCoord is also accessed using the real IP address of the host instead of the loopback address. Using the IP address of the host ensures that ConCoord nodes on remote hosts can be accessed. During the bootstrapping of an Ambient Network the ConCoord registers its location (http://IP_Address/axis/services/ConCoord) with the ACS registry. As a result, any client can query the ACS registry to obtain the location of the ConCoord. When the ConCoord's location is obtained the entity can now access the ConCoord.

When one ConCoord node (also called a ConCoord Service Access Point (SAP)) is started up on a host it determines whether it is the root node or the gateway. If the ConCoord node is a gateway, then it forms the DHT and listens for other nodes to join it. If the ConCoord node is not a gateway then it searches for possible gateways for it to join up to form a DHT. As a result of this, one ConCoord node which serves as a gateway must be started up first for other ConCoord nodes in the AN to join it to form the DHT. Future work will involve investigating the use of the GANS protocol [58] to locate ConCoord gateway nodes and the local ConCoord node.

A Distributed Hash Table based ConCoord

Distributed hash tables (DHTs) are scalable, decentralized registries that map a set of keys among participating nodes to certain stored values. The protocols in DHTs can efficiently route messages to the unique owner of any given key to find the corresponding value. DHTs are typically designed to scale to large numbers of nodes and to handle continual node arrivals and failures [39]. They have been proposed as a generic building block for many large-scale distributed applications [40][41]. DHTs form a routing overlay on which requests for data entries are routed toward the nodes currently managing them [39]. Each node in a DHT is assigned a unique nodeId from a defined keyspace (usually 2^{160} keys large). Each piece of data that needs to be stored in a DHT is assigned a key. A key is a fixed length hash. In the Bamboo [40] DHT implementation the data values are stored on the node whose node ID is closest to the data value's corresponding key and also replicated at a number of other nodes. Requests for access to the data stored in a DHT are routed to the nodeID closest to its key and will be routed in O(log N) hops where N is the number of nodes in the DHT [41].

The ConCoord is a distributed entity consisting of ConCoord nodes on several hostyes s. These nodes can potentially be constantly joining and leaving the network, resulting in constant churn. Due to a DHT's distributed storage capabilities as well as its ability to deal with constant churn, storage redundancy and its scalability, it was deemed an ideal implementation base for the ConCoord. The Bamboo DHT [40] implementation was used due to its ability to deal with constant churn and its redundancy.

Operational Details

The registry of the ConCoord maps context UCIs to the location information of the context objects identified by these UCIs. The registry of the ConCoord maintains a UCI → location mapping where the UCIs are the keys, and the locations are the values. Each ambient network node runs a ConCoord node. These ConCoord nodes together form the ConCoord DHT.

A context source registers its context object by contacting the local ConCoord node. A common, uniform hash function is applied to each UCI to create a constant length key. The resulting hash key identifies the node on which the entry is to be stored, i.e. the corresponding value is routed through the overlay and stored on the node with the ID closest to that key. For redundancy, the corresponding value for each key is not only stored on the ConCoord node with the node ID closest to that key but also at least two other nodes (note that this number can be increased or decreased). This makes the ConCoord resilient to nodes dropping out of the overlay.

When a context client issues a RESOLVE request for a given UCI, it contacts the local ConCoord node which then hashes the UCI and maps it to the ConCoord node whose node ID is closest to that hash. This request is then routed to the ConCoord node on which the corresponding value is stored. The value is then routed back to the local ConCoord node which passes it onto the context client. In this way, any ConCoord node can find out where the value corresponding to any key is stored and retrieve that information. Figure 16 below illustrates the interactions between context clients, sources and the ConCoord in an Ambient Network.

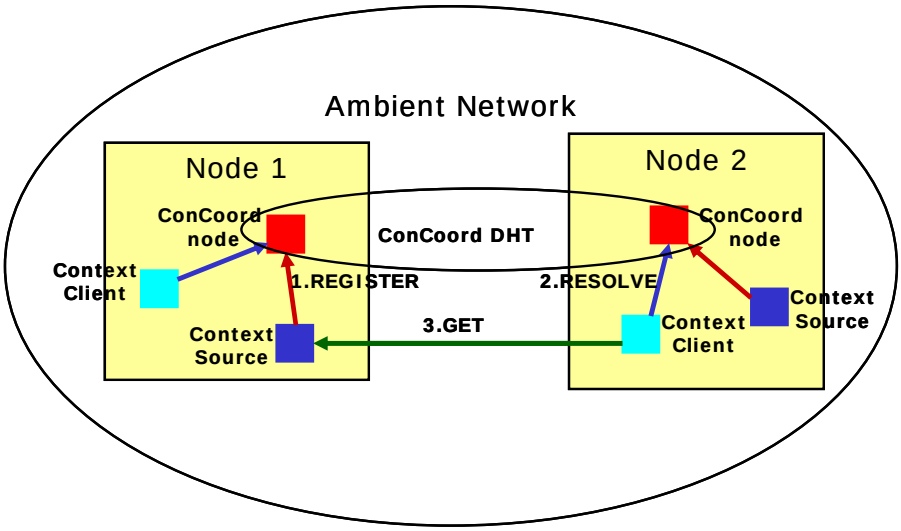


Figure 16 – Illustration of the interactions between context clients, sources and the ConCoord in an Ambient Network

Implementation Details

The Bamboo code was obtained from [40] and installed on each ambient network host machine. Java classes were written to interface with the Bamboo DHT as well as to implement the REGISTER and RESOLVE primitives, the storage redundancy, ConCoord location and basic access control. Further work will involve implementing a more complete

access control and authorisation framework as well as ConCoord location through SRV records and GANS.

As stated above the ConCoord's registry is implemented using a Bamboo DHT [40]]. To implement the ConCoord, the Bamboo code was obtained from [40] and installed on each Ambient Network host machine. Java classes were written to interface with the Bamboo DHT to implement the REGISTER, DEREGISTER and RESOLVE primitives. These primitives are called DHT interface classes. The nodes in the ambient network each run a Bamboo DHT node and along with the DHT interface classes, form a ConCoord node. These ConCoord nodes together form the ConCoord DHT. Furthermore, a class to interact with the ACS framework was developed to implement the ACS registration and bootstrapping functionality. A Web Services interface was developed as discussed below to allow interactions with ConCoord clients.

Web Services Interface

When a ConCoord Client (i.e. Context Source, Context Client or Context Manager wants to query the ConCoord, it does this through a Web Services interface. The ConCoord appears as a Web Service running in the Tomcat/Axis Web Services implementation. Through this Web Service the register, resolve and deRegister methodsAs discussed above, the local ConCoord on each node is accessible through a Web Services URL e.g. http://192.168.1.2:8180/axis/services/ConCoord Hence the location of any ConCoord node is unique in an Ambient Network (made unique through the IP address). Using this URL, the register, resolve and deRegister methods are accessible to ConCoord clients. To enable interaction with the ConCoord Web Service interface the context source, client or manager must run a Web Services client. To access the ConCoord through Web Services a Web Services interface has been implemented using a class called ConCoord. This is why the ConCoord's URL ends with "ConCoord". This Web Service class accesses the REGISTER, RESOLVE and DEREGISTER primitives by instantiating and calling methods from the DHT interface classes. Communication between the ConCoord clients and the ConCoord Web Services interface is through SOAP messages while that between the DHT interface classes and the Bamboo DHT node is through Java RPC. The Web Services based architecture for the ConCoord and a ConCoord client is illustrated in Figure 17 below.

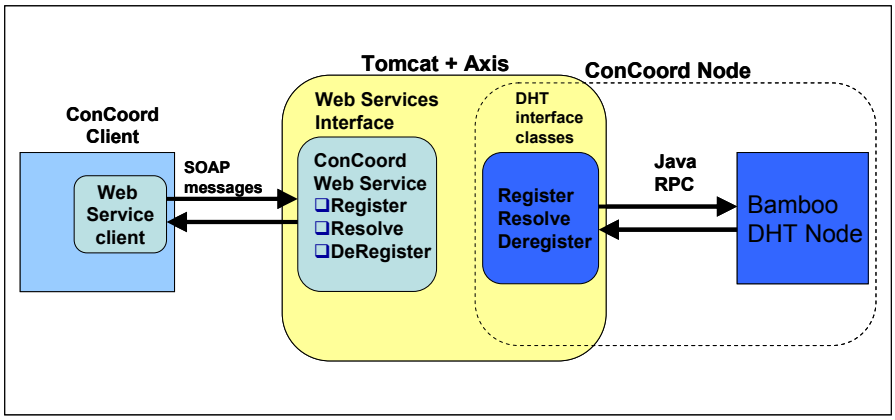


Figure 17 – Illustration of the interactions between a ConCoord client and the ConCoord using the Web Services interface

Composing ConCoords

Usually DHTs do not support the concept of composition, neither in the weak form of gatewaying nor the strict form corresponding to AN absorption. In contrast, ConCoords must be able to compose and de-compose in lieu with its ambient network. As a result,

methodologies for the composing and decomposing of ConCoords have been developed and are presented in Annex A5.1.

Conclusion

The ConCoord is a distributed registry mapping Universal Context Identifiers (UCIs) to the location of context objects. Context sources register their UCIs while context clients resolve UCIs to obtain the location of context objects. In AN, a ConCoord plays a central role in the ContextWare system. The ConCoord is the first point of contact for context sources and clients and allows dissemination of context information by bringing together context clients and sources. In our current prototype it is implemented using a Bamboo DHT and is accessed through a Web Services interface.

2.2 Context Management FE

As opposed to the ConCoord FE that maps UCIs to location of context information, the Context Management (CM) FE manages context within and across domains. It therefore provides the means to:

- allow context sources to delegate the distribution of the context information they provide
- allow recursive aggregation and processing of context information to be done once and on behalf of many clients
- manage the distribution of context information to the most appropriate location in the CIB to account for scalability, access rates and update rates
- schedule interactions between context sources and context clients, monitoring these interactions, re-allocating channels of interaction as needed

More precisely, these means are achieved through a number of specialised Context Managers, each of them dynamically created as need arise for carrying out a particular task. The Context Management FE therefore represents the service provided by a number of distributed processes that can:

- be dynamically created based on context client requirements
- provide aggregation, translation, inference capabilities
- cache context information on behalf of context sources
- cache context information at different locations to address performance optimisation and minimise retrieval time from clients.

A very specialised Context Manager that has been created in a different work-package to deal exclusively with mobility triggers is the Triggering FE. In this section we concentrate on the features of the more generic Context Management FE without necessarily specifying the particular task each of the context managers achieves but rather focusing on the infrastructure requirements and features.

Context managers once created they register their output type and capabilities with the ConCoord. The ConCoord's registry therefore maintains mappings to location of context sources and also of context managers. This also enables recursive multi-pipe establishment in a distributed way. The ConCoord locates the final context manager, which locates the managers for its input, which in turn locate the managers for their input, and so on, until the inputs are all initial objects (i.e. basic context sources).

In the remainder of this paragraph a number of features of the Context Management FE are illustrated in more detail *(may need reshuffling to follow more logical order)*, namely the concept of UCIs (Unified Context identifiers), a policy-based context modelling and

management, a Jini-based implementation of the Context Management FE, the features of a “composition-friendly” Context Information Base to conclude with a number of Context Sources / Sensors used in the implementation of ContextWare functionality.

2.2.1 Context Identifiers

A basic element of the CW System is the context association. A context association is a uni-directional relation from a context source to a context sink, i.e. the direction is that of the context information flow. The context source is the component providing/producing the context information, and the context sink is the component using/consuming it.

A context association has certain attributes, including Context Level Agreements (CLAs) and Quality of Context (QoC) specifications. The context association is a producer–consumer relation, where the context source acts as the producers, and the context sink acts as the consumer. Modes of retrieval (server push vs. client pull) are also attributes of a context association.

A context source is an entity, which provides context information; examples include context sensors, networking services, network resources, and flows. A context sink is any entity embedding a context client/consumer (making the entity context-aware. We do not make any assumptions about context sinks, like when or why they request, how they are affected by, or what decisions are made based on context information. A context source provides context information in the form of data objects. The content and structure of a context object are determined by its type, which may be, for instance, an SNMP MIB an XML Document Type Definition (DTD), a file format etc. A particular context object of a given type is identified by its Universal Context Identifier (UCI). The information represented by a context object may be dynamic and change over time, but only the context source can update its context objects. The basic operations to retrieve context information are that a context client fetches the content of a context object by means of a producer-consumer protocol, or subscribes for event notifications delivered when the context object changes its state.

Universal Context IDs (UCI) are a new type of Uniform Resource Identifiers (URI) [9] and uniquely identify a given context object, but not its location within the network. A UCI is the conceptual rendezvous point between client and sources, i.e. whenever a client wants to get specific context information; it has to know the UCI of the object representing this information. Similarly, a context source is assumed to know the UCIs of the context objects it wants to publish.

A fully qualified UCI is as follows: ctx://domain.org/path?options where

- “ctx” is the new URI scheme,
- “domain.org” is the DNS domain name within which the context object exists,
- “path” is a sequence of words separated by slashes (‘/’), and
- “options” specifies further modifiers like the data encoding format on the wire.

The next sections describe the AN ContextWare internal structure, the primitives for communicating with it and a possible authorisation and privacy framework. Context information is modelled as data objects identified by Universal Context IDs (UCIs). Given the nature of context information, its sources and its potential clients, we recognize that scalability of the solution is of prime importance. To fulfil these basic requirements we based the design of CW on the existence of two building blocks that separate the task of registering sources and the task of name resolution of context information for clients from the task of managing the actual context information in a distributed way. The Context Information Base (CIB) provides flexible context storage capabilities and it supports the operations of these two functional entities and in particular the uniform context dissemination from the vast diversity of network context information sources.

2.2.2 Policy-Based Context Modelling and Management

This section discusses the approach of modelling, managing and disseminating context using policies. Policies provide a coherent mechanism to represent, process (using aggregation, filtering…), and disseminate both simple and complex context. A policy can be described in the form of one ore more rules that specify actions to be performed in response to certain conditions.

2.2.2.1 Policy-based Context Modelling

The representation of context in terms of policies bridges the gab between simplified context models and complex ones with computational limitations. As shown in Figure 18**Error! Reference source not found.** the meta model describes the ambient space in terms of a set of entities and properties. Entities represent both physical objects (e.g., users and devices) and logical ones (e.g., FEs and services).

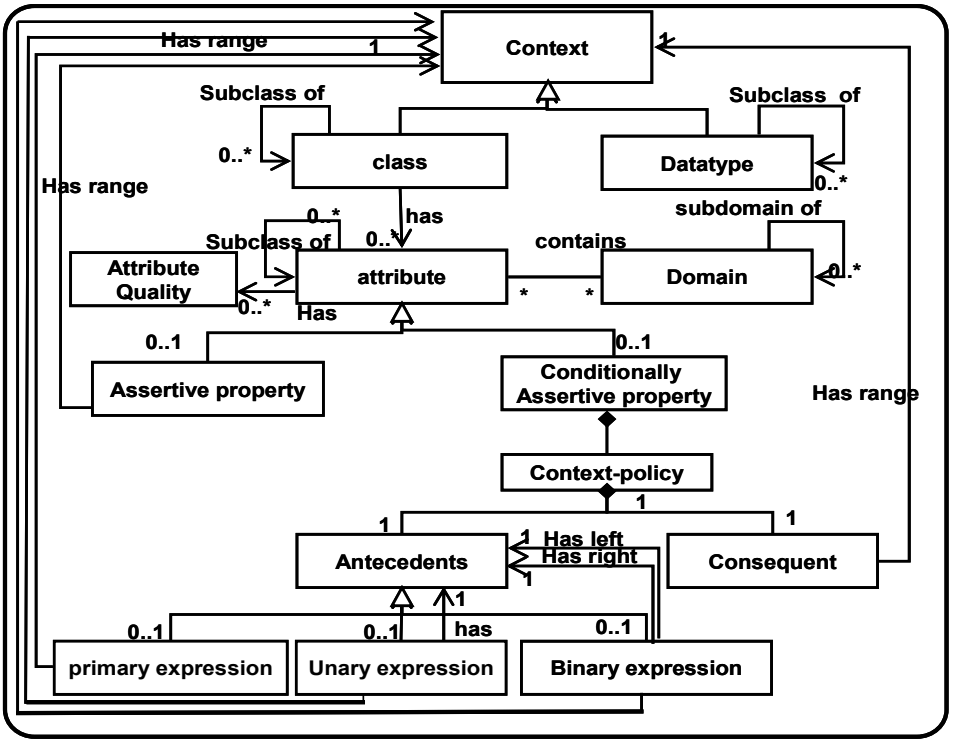


Figure 18 – AN Context-meta model

The model describes the ambient space in terms of a set of classes and properties or attributes. Classes represent both physical objects (e.g., persons and devices) and logical ones (e.g., FEs and services). The context of a class is described through a set of attributes (e.g., cost of a service and bandwidth capacity of a link) such that each property has a specified domain and range. Properties are also used to describe a relation between two classes (e.g., a device is part of an AN). Properties can optionally be associated with other attributes that describe context qualities such as its validity, accuracy and granularity. These attributes will be used during the CLA negotiations.

The model further distinguishes a property as either assertive or conditionally-assertive. An assertive property refers to a univalued attribute that can be described in the form of a symbol and a range-value pair. On the other hand, a conditionally assertive attribute, which describes a multi-valued context, is represented through a set of context-polices of the form IF (A) THEN (B), where A and B are referred to as an antecedents set and a consequent, respectively. An example of this type of properties is the cost a service or the availability of a service.

Context-policies facilitate the use of context by various clients including FEs, services and applications. On one hand, they provide a mechanism to describe the multi-valued context (i.e. multiple clients can get different context values from the same context source at the same time). On the other hand, they can be used to derive new context information through context transformation, filtering and aggregation. And finally, policies can be used to control and monitor context dissemination to services and applications which can either request context values or decisions of actions to be performed according to the current context.

While the purpose of the process of translating context to policies is accomplished at the context management level, the policy enforcement process is related to each client entity, where specific actions within that entity are invoked and enforced. As shown in Figure 19**Error! Reference source not found.**, each context client is therefore associated with a policy infrastructure to evaluate and enforce its own policies (i.e. policy decision and enforcement points). Multiple context clients may share a single policy infrastructure in case they are not equipped with their specific policy infrastructure.

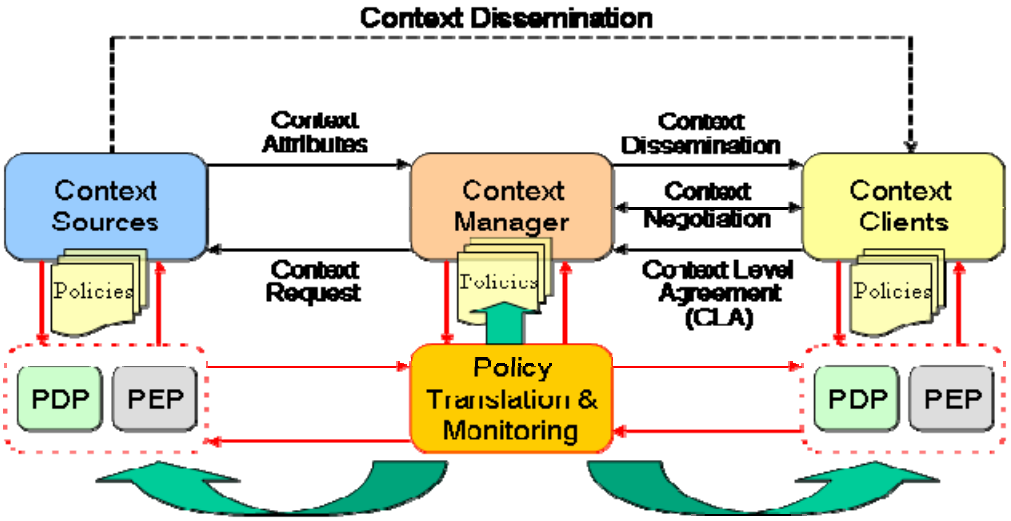


Figure 19 – An Overview of the policy-based context management process

The use of policy management infrastructure at the context sources level is also possible depending on the capabilities of the source. Specific policies may be associated with a context source to monitor its possible multi-values and multi-attributes contextual information and to provide context dissemination directly to the clients in accordance with the context management agreement. This mode of operation reduces dependencies on the context management entity and allows more flexible context dissemination besides the policy translation mechanism.

2.2.2.2 Context-Processing Policies

Context-processing policies are used to derive new contexts from primitive ones such as *at work*, *nearby*, and *in-meeting* derived information from the location, active services, and used devices contexts. A generic context-processing policy can take the form shown in Figure 20**Error! Reference source not found.**.

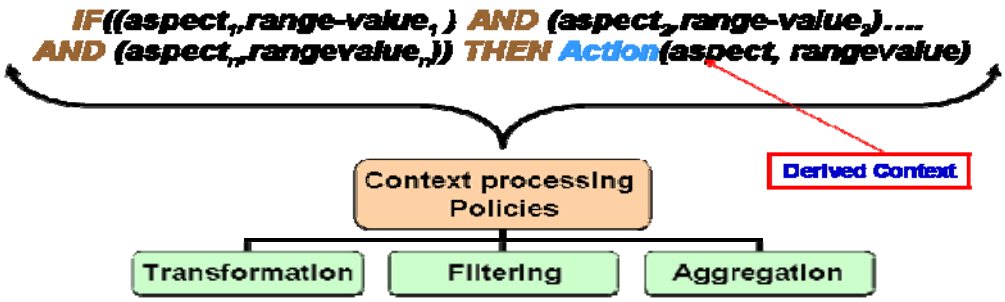


Figure 20 – Context Processing Policies Template

Context-transformation policies (CTP) represent predefined strategies for transforming some aspects to others without taking into account the clients. They are usually defined once at the context-model creation time and are specific to a certain aspect. CTPs have the same action, which is ***evaluate***. Evaluate action enforces the creation on new aspect with its rangevalue. The following example illustrates the creation of the aspect *indoor* with the rangevalue *true* from the location and the area aspects: IF((location(*),=P) AND (area(building),=(P1, P2, P3,P4)) AND (Inside(P, area),=true) THEN *evaluate* (***indoor***(*),true).

Context-filtering policies (CFP) perform summarization operations on primitive context to derive new ones. Examples of summarization operations are maximum, minimum, out of range and average values of contextual information bot in time and space. The following example illustrates the creation of the aspect *abnormal* from service response time context information: IF(responsetime(serviceA), >4 ms) THEN *evaluate* abnormal (serviceA, true). The CFPs actions can either be *evaluate* or a context control action such *store*, *notify*, *set* …

Context-aggregation policies (CAP) are the most expressive types of context and are used to describe derived context that is highly dependent on other contexts. CAPs are customized by clients using pre-existing context templates by replacing entity classes with specific entities as well as by adding and removing policy antecedents. The following example illustrates the creation of the aspect *nearby* from the location and activity of Alice and Bob: IF(distance (Alice, Bob), <100) AND (activity(Alice),"walking") THEN *evaluate* (nearby (Alice), Bob).

Actions in CAPs can be as simple as an evaluation of a newer context or it can contain actions to be enforced by context-clients.

2.2.2.3 Context Control policies

Context control policies define strategies operations that are applied within the context management FE. For example, they define what types of context should be stored, how frequent should they be updated, access control strategies as well as dissemination strategies. These policies are dynamically created or adapted to meet a predefined set of objectives in terms of storage capacity, utilization percentage and network communication cost. The following control policies are considered:

Acquisition and dissemination Policies (ADP) This set of policies controls the selection of the appropriate acquisition and dissemination mechanisms based on the specifications of the context clients, sources and the properties of disseminated context. For example, the policy: **IF**(number of context requesting clients > 10) **THEN** broadcast, indicates that a decision to switch from a single user delivery mechanism to broadcasting should be taken whenever the number of requests for a context exceeds a certain value. Another example is an acquisition policy is **IF**(contextsource= hardware sensor) **THEN** context should be broadcasted every 1 min).

Storage and updating policies (SUP) control the decision of whether to maintain context within the context management FE or at sources as well as the required frequency of updating the context within the CM FE. The need to store context within the CM FE depends on the frequency of requesting the context, the availability of the context-source and the communication cost to that source. Due the dynamic nature of ambient networks these parameters can widely vary, and hence, policies that control storage and update of context are frequently adapted.

Access-control policies (ACP) control the allowable types of access to specific context as well as permissible context-processing operations that can be applied to context within the specified policy domains.

2.2.2.4 Context Dissemination

Context dissemination is performed in one of the two forms: (i) delivered to context clients in the form of a list of context attribute-value pairs, (ii) or delivered as context-policies which are interpreted and enforced at the context clients' side.

As context changes occur, registered clients will be provided with pairs of context attributes and their respective values which may trigger some existing policies, or with context-policies aiding clients in performing appropriate actions in response to changes in the context.

2.2.2.5 Policy Infrastructure Requirements

The representation and the management of context using policies require the existence of policy tools and infrastructure at the context management level, context clients and optionally at the context sources. The translated context policies are monitored in the same manner as for interpreting and enforcing user-defined policies, making use of the policy decision points (PDPs) and enforcement points (PEPs). However, the context policies lifetime depend on the validity of the associated context information. The use of context policies makes the dynamically changing context more accessible to different entities and services, which in turn dynamically adapt their behaviour and operation mode by enabling and executing appropriate policy actions.

Figure 21**Error! Reference source not found.** shows the main policy components required for monitoring policies from the context management point of view. In addition to the components that are already defined in the IETF model (storage, decision making and enforcements points), new capabilities need to be supported. These capabilities include:

- Context-policy Reporting Points (PRPs) which can reside in context sources and communicate the context information to the context management FE according to a predefined protocol.
- Distributing policies to multiple PDPs and having the ability to evaluate policies across a number of PDPs in coordinated manner.
- Associating a PDP with multiple PEPs and achieving the decision making to the appropriate PEP.

Disseminating the context either by selecting and sending a raw context value, which enables and executes a pre-defined policy action if a situation occurs in a given context (i.e. the Raw Context Provider component), or by creating a dynamic context policy so that entities and services can behave according to a new context situation (i.e. the Context Policy Generator component).

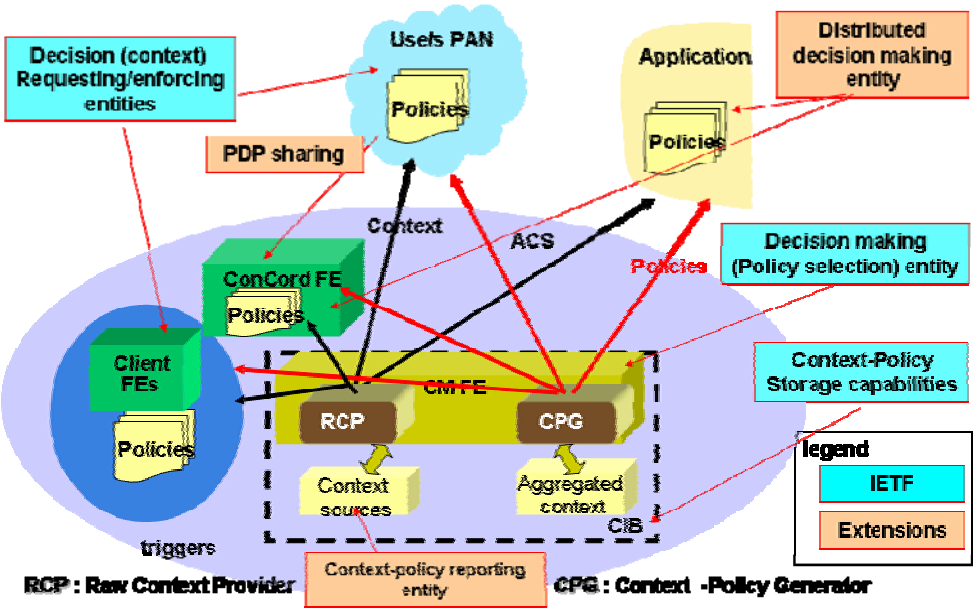


Figure 21 – Main Policy Components for monitoring Context Policies

2.2.3 A Jini-based implementation of Context Management FE

Jini technology can be used to build adaptive networks that are scalable, evolvable and flexible as typically required in dynamic computing environments. Jini provides a functionality, which is required by *Context Management FE*, as follows:

- Provides LookupService, which is a Jini service registry with lookup mechanism. LookupService may be distributed, and it may be found via LookupDiscoveryService. Services are searched by properties defined by user defined properties (e.g. UCI);
- All services in Jini are represented in LookupService as proxies. These proxies are written in Java language, however may communicate with services using service specific protocol, e.g. if service speaks with SNMP protocol, proxy must provide communication with service via SNMP;
- Dynamically equips a client with necessary proxy object which is provided as a contact information. The only thing a client must know, is an information to look up in the service; e.g. service interface and/or some user-defined properties for the service;
- Provides LeasingService. It assures that only up-to-date services exist in LookupService. Service must renew its lease in specified amount of time (e.g. one minute). If it is not done, service is removed from LookupService (e.g. in case of service or network crash);
- Offers support for remote events, which are required for *push* model of communication (context client subscribes for notifications about context changes). Remote events also use Leasing mechanism e.g. Jini LookupService may notify some clients about new services appearing in the network and about their disappearing.

2.2.3.1 Mapping Context Management FE to Jini

Context Management FE building blocks may be mapped to Jini programming model in the following way:

- Context Manager is mapped to Jini Service. This service provides methods for context management:
  - `get()` (gets context managed by Context Manager),
  - `subscribe()` (subscribes for notifications about context changes),
  - `sourceAdded()` (adds Context Source to Context Manager);
  - `sourceRemoved()` (notifies CM, that specified CS is no longer available in AN); this event is executed, when ConCoord notices, that CS is no longer renewing its lease;
- Context Source is implemented as Jini Service which provides methods for context dissemination: `get()` (gets context from this source), `subscribe()` (subscribes for notifications about context changes);
- Context Client is represented as Jini Client. Context Client contains method `notify()` (which notifies client about context changes), which is mapped to remote Jini event.
- Context Description; It is mapped to OWL Ontology (described in Section 2.2.3.2).

The proposed mapping shows that Jini programming model in a very natural way satisfies most of *Context Management FE* implementation requirements. Due to distributed architecture, notification and leasing mechanisms Jini assures elements of fault tolerance and auto-configuration. Moreover, Jini is a lightweight framework, designed especially for mobile devices, so it looks like ideal solution for Ambient Networks.

2.2.3.2 Ontologies

Ambient Networks have identified the use of a common ontology as a requirement for entities (Functional Entities) to exchange context information in a consistent and unambiguous way. Compositions/decompositions taking place in Ambient Networks require efficient knowledge transfer and automatic decision making. This in turn demands proper conceptualization of the context domains having roles in the process of composition/decomposition.

That is why an ontology language is a good language for defining context. Ontologies are often used to support semantic web, and may be adopted for context description.

We decided to use Web Ontology Language (OWL), standard recommended by W3C. OWL is represented by XML/RDF document, which may be read by any XML/RDF-enabled tools [47][48].

According to Ambient Networks issues, following requirements for *Context Management FE* are met by OWL:

- Conceptualization requirements:
  - formality;
  - efficient reasoning and inference mechanisms;
  - dynamic creation, modification, and extensions of the ontology model;
  - models merging, checking and partial knowledge validation;
  - support for interoperability between different models;
- Engineering requirements
  - ease of development of new ontologies for different domains in ANs,
  - upgrading existing ontologies / ontology evolution;

- integrating / merging existing ontologies;
- compatibility with standards;
- representation language independence.

This rather general consideration can be illustrated by a simple example. Let us consider situation, where there are few Context Sources and a Context Manager, which is operating as a Context aggregator. Using ontologies we are able to aggregate context from these sources. New ontology is the union of the source ontologies. The merged ontology captures all the knowledge from the original ontologies. It could be done in a semi–automatic way. Of course there is a problem in case of ontology conflicts (if few context sources provide context which overlap), however there is an idea to create Ambient Networks Ontology, to avoid this issue. Figure 22 depicts simple ontology prepared for testing model.

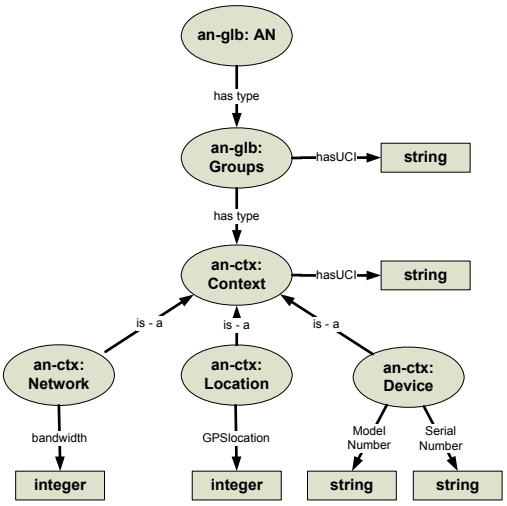


Figure 22 – Simple ontology

2.2.3.3 Ontology and UCI Relationship

Client may ask ConCoord only for UCIs that are registered and bound with Context Source or Context Manager. For instance with:

```
ctx://www.ambient-networks.org/ambientnetworks
```

must be associated Context Manager to perform context aggregation. This CM gets context from connected sources and sends it to the client. In our example CMs are used for aggregation of context from Context Sources and other CMs. Connected CMs forms structure of a tree, as depicted in Figure 23.

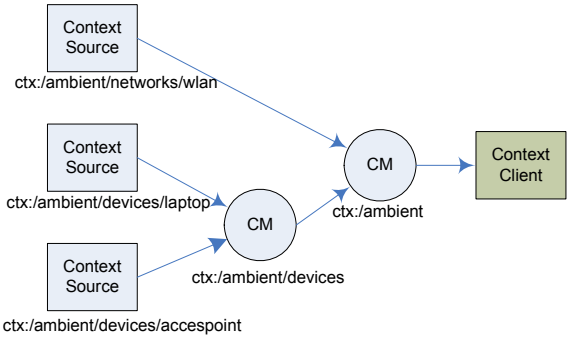


Figure 23 – Tree of context associations in AN with two CMs

Context Managers may also provide additional functionality, like caching context fetched from source to reduce network load. Important thing is, that depicted tree is rebuild automatically when new CM appears in the network or disappears (it may happen because of a very dynamic Ambient Networks structure).

UCI is strongly connected with ontology provided by Context Source/CM with specified UCI, as shown in Figure 24. Depicted context tree is an instance of ontology class mentioned before. Black arrows show relations between UCI and chosen ontology instances.

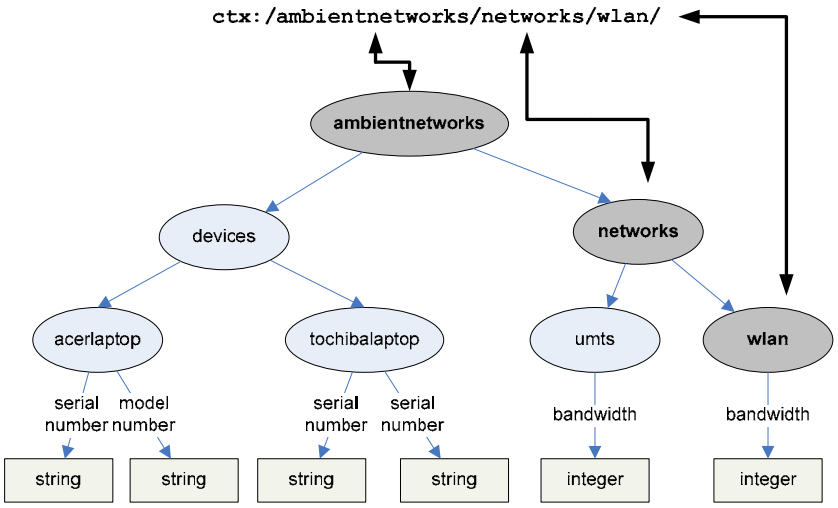


Figure 24 – Simple context tree for Ambient Networks

Each part of the UCI has a corresponding node (instance of ontology class) in the context tree. This way ontology could be considered as template for UCI.

2.2.4 Context Information Base / UCIs

The Context Information Base is co-located with the ConCoord and Context Manager, the two Functional Entities described earlier. The Context Information Base (CIB) is the architectural building block that provides flexible storage capabilities, mainly in support of the Context Management FE. The CIB is meant to address the following points:

Context information may need to be distributed for performance reasons (to allow both, efficient retrieval from context clients and fast update from context sources).

Not all the context information will be retrieved directly at its source, for various reasons (the owner of context info may want to publish it but not deal with all the requests from context clients, the owner of context info may not have the information available in the formats that the clients understand).

The diversity of context sources makes it unlikely that information can be retrieved directly from the sources according to a common ontology [10]. The CIB therefore offers the flexibility of storing context information according to a common ontology. This approach simplifies the implementation of context clients (by making it more generic rather than tailored to read from i.e. a particular sensor).

Context Managers described earlier need to have access to a storage facility to make the results of their processing readily available to clients. Note that this extends the applicability of the previous translation example to apply to context aggregation, interpretation, and other processing functions.

The Context Information Base therefore adds on top of the ConCoord one more level of flexibility, which is twofold. On one hand it allows sources of context information to delegate publishing the information they own to the Context Management FE. On the other hand it

allows us to keep in the design the flexibility of caching and pre-processing context information to accommodate better performance for both, client requests and source updates.

2.2.4.1 Context Composition

As part of the ContextWare system we have the Context Information Base (CIB). The composition of two ANs implies the composition of their related ContextWare systems, and therein the composition of their CIBs. Composing CIBs consists mainly on composing their content and enabling control sharing between their respective operations, needed for the manipulation of that content (e.g., collection, modelling, and dissemination).

Here we are interested in CIBs composition. We assume that the CIBs are designed to be composable, but they can be implemented using different technologies, to suit different networking environments. They can therefore be centralized (e.g. for fixed and centralized networks) or distributed (e.g. for P2P networks). They can use different information publication/discovery mechanisms and/or protocols and they can store different types of information (e.g. aggregated/raw information).

We present the general architecture that we are proposing for CIBs composition. We also present a solution for enabling the inter-communication between heterogeneous CIBs on the fly, one of the main problems encountered during the composition.

2.2.4.2 Overall architecture

As part of our architecture, we introduce a new functional entity as part of the composition FA of the Ambient Network. This new entity is called Registry Composition Entity (RCE) and is responsible for orchestrating the ContextWare composition process. Figure 25 illustrates the different components of the RCE.

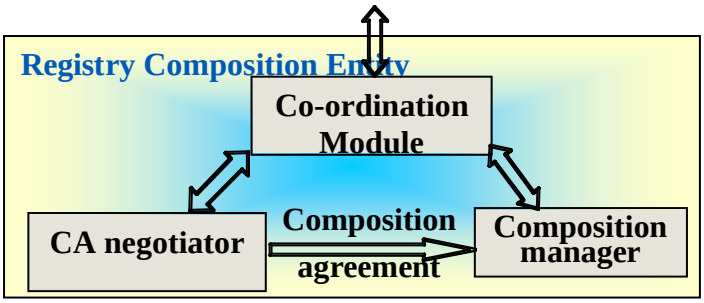


Figure 25 – Registry Composition Entity

The co-ordination module enables inter-coordination between the different RCEs involved in the composition process. The CA negotiator carries the negotiation between the different networks concerning the ContextWare composition and generates the composition agreement. The Composition manager is responsible for executing the composition agreement. The ContextWare composition is initiated by the Composition FAs.

2.2.4.3 Interworking between heterogeneous CIBs

Seen that different CIBs can use different information publication/discovery protocols, the RCEs may have to negotiate an interworking solution between the different CIBs as part of the composition. They may also have to negotiate how to implement this solution on the fly. The interworking can be provided using two possibilities. The first one is to create a gateway(s) between the concerned CIBs on the fly. The second possibility is to deploy a common protocol on the fly, to these CIBs. Example of the latter possibility is when we have to interconnect two CIBs: CIB1 and CIB2, which are using distinct protocols P1 and P2 respectively. We can either install P1 in CIB2 or install P2 in CIB1. To minimize the number of protocols that can be deployed, we can use a standard protocol that will be installed into all the CIBs that do not already support it.

To enable on-the-fly deployment of protocols, we propose a software architecture based on active networks and network programmability. Network programmability refers to the ability to inject executable mobile code into network elements (e.g. router, switch), to create new functionalities at runtime. Active networks are programmable networks and they are extensible at runtime. They can enable on-the-fly protocol and gateway deployment. For this reason, we used them as the foundation for a new architecture for protocol deployment.

In the following sub-section we discuss how to deploy a new protocol on the fly, using network programmability. After that, we present a software architecture that will enable this deployment.

2.2.4.4 Protocol deployment on the fly

To enable the automatic deployment of protocols, as part of facilitating inter-CIBs communication, we are proposing the following solution: for each network, we use a protocol server where we store the information discovery and publication protocol (IDPP) of the local CIB and/or the standard protocol. When protocol deployment is needed, RCEs negotiate the protocol to deploy and use one of the protocol deployment approaches provided by active networks to make the protocol available. The protocol agreed upon is downloaded to and installed in all of the appropriate nodes (CIBs and/or gateways).

2.2.4.5 Software architecture for protocol deployment on the fly

The software architecture that we propose is based on DINA, a programmable network platform that enables the deployment and management of programmable services. The main components of our architecture are the policy server, the protocol installer and the installation broker (Figure 26). The policy server and the protocol installer are added to the RCEs. The policy server includes the policies that regulate the registries' composition. The protocol installer is part of the composition manager entity and is responsible for the initiation of the protocol installation. The installation broker is added to the DINA platform on the registry side to enable and control the installation of new software.

To deploy a protocol that is stored in a protocol server, the protocol installer creates the installation active packet, which is sent to the registry where the protocol must be installed (steps 1 and 2). When the session broker in the registry side receives this packet, it creates an installation active session that executes the active code of the packet (step 3). This session downloads the protocol to install from the server, and then uses the installation broker to install and activate the new protocol in the current node (steps 4,5,6 and 7 respectively).

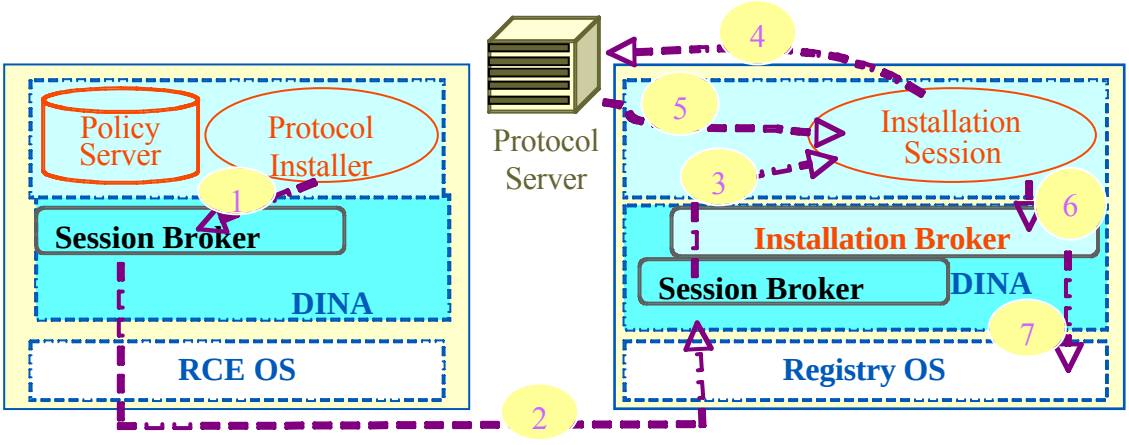


Figure 26 – Protocol Deployment using DINA

2.2.5 AN Network Sensors / Context Sources

Network sources play a key role in the management of AN, as all context or networking information – whether raw, derived, aggregated, inferred or archived – can ultimately be traced to an original physical or logical source. The programs that interface with these sources of context or networking information and provide the abstractions for integration with MangementWare are called *network sensors*. The network sensors can be pre-deploy or dynamically deployed on demand in the AN underlay or overlay for a specific use. A number of types of network sensors are under development within the project: P2P context event sensors, Quality of Service sensors, Node and device context sensors and flow context sensors. The information collected through sensors can be used for network and service self-adaptation, triggering network services, implicit QoS signalling, mobility support, in context-based flow classification, in content delivery, in mitigating network attacks, and in controlling malicious or resource-wasting flows such as spam and spit, interplay and optimisation of the underlay with the overlay.

2.2.5.1 P2P context event sensor

The P2P context sensor listens for events from a peer-to-peer overlay management platform (details of which may be found in [30]) that manages network composition. It provides notifications when members join and leave an overlay network, the identity of the super peers and normal peers in each overlay. This context information includes the peerIDs of each peer along with the corresponding overlayID and IP address. The p2p context information is stored in an SNMP MIB. SNMP traps are used to send context event notifications that arise when peer to peer network context changes. These context events are captured by the peer-to-peer sensors. Examples of these context events are: `newOverlayMemberJoined`, `overlayMemberLost`, `becomeSuperPeer`, `newMember` and `superPeerLost`.

2.2.5.2 Quality of Service (QoS) sensor

This class of sensor is envisaged to monitor per-interface traffic characteristics. Filters are defined as traffic measurements across an interface based on `iptables` entries. Each overlay is mapped to an `iptables` entry and hence we are able to calculate the traffic patterns specific to each overlay. The same can be done for services. These traffic measurements are combined with the actual bandwidth capacity of an interface to create computed context detailing the bandwidth utilisation. This context is handled by the QoS sensor and delivered to the context clients. The prototype currently measures only traffic rates (i.e. bandwidth) in multiples of bits per unit time (e.g. kbps) and packets per unit time, and does not yet support the measurement of delay, jitter, and other metrics.

This sensor may also be adapted to per-flow monitoring if each specific flow is mapped to a user-defined IP chain. However, an alternative sensor dedicated to per-flow context sensing is currently under development, and is discussed in Section 2.2.5.4

2.2.5.3 Node and device context sensor

This class of sensor obtains information related to the characteristics of the network node or device, such as its host name, processor type and load, memory size, available interfaces, or display capabilities. A prototype version of this sensor has been implemented as DINA code, and is able to obtain node-related context via DINA's InfoBroker service, which in turn relies on the SNMP MIB of the underlying device.

2.2.5.4 Flow context sensors

Flow context sensors are designed to obtain flow context, defined as any information that characterises the situation in which flows are generated, transmitted, received or processed within the network and in end-hosts. Unlike traditional approaches to per-flow monitoring and metering that tend to view and obtain information about a flow in isolation from the rest of its environment, flow context deals with both its *intrinsic context*, namely a

flow's internal state and properties, and *extrinsic context*, which pertains to factors or entities that are external to the flow itself, but which affect the flow nonetheless. Examples of intrinsic flow context include (traditional) metrics and statistics such as the flow's rate, size, duration, protocol type or transported payload; while extrinsic context may include the nature of the application generating the flow, the state and profiles of the devices along the flow's path, or even the high-level intentions of the users generating or consuming the flow.

A diagram showing the main components of the current flow context sensor prototype is shown in Figure 27, while Figure 28 shows a visualisation interface provided as a front-end for flow management applications.

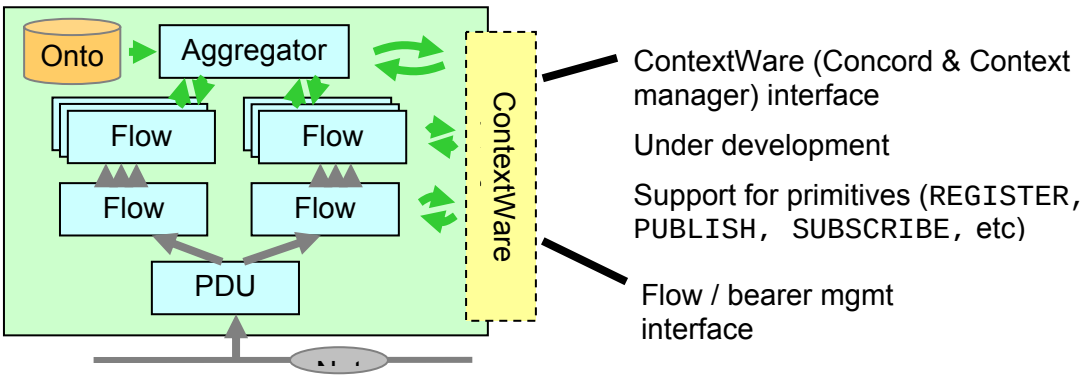


Figure 27 – Basic architecture of flow context sensor

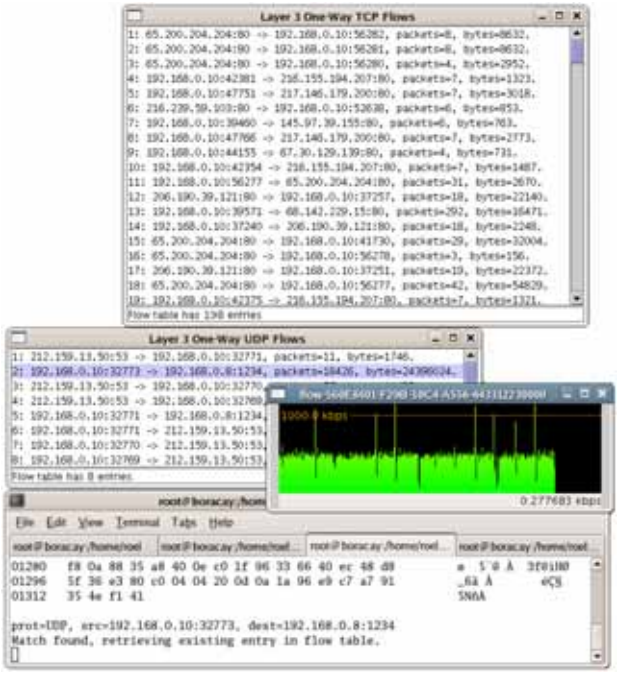


Figure 28 – Visualisation interface for flow context sensor

Development on the P2P context event sensor, the QoS sensor, and the node and device context sensor started in Phase 1 of the project, and further detail on their details on the design, implementation and integration of these sensors in a demonstration prototype may be found in [26][27]. The development of flow context sensors for use in Ambient Networks commenced in Phase 2. Other ongoing work on flow context sensors involves the development and integration with an ontology for flows and flow context, as well as the integration of the sensor with a back-end reasoner. For a more detailed discussion on the design, implementation and applications of the flow context sensor, the reader is kindly referred to [28][29][30][31][32].

2.2.5.5 Sensor integration and demonstration

The P2P, QoS and node context sensors are already developed. Work on the integration with the flow context sensor is currently ongoing.

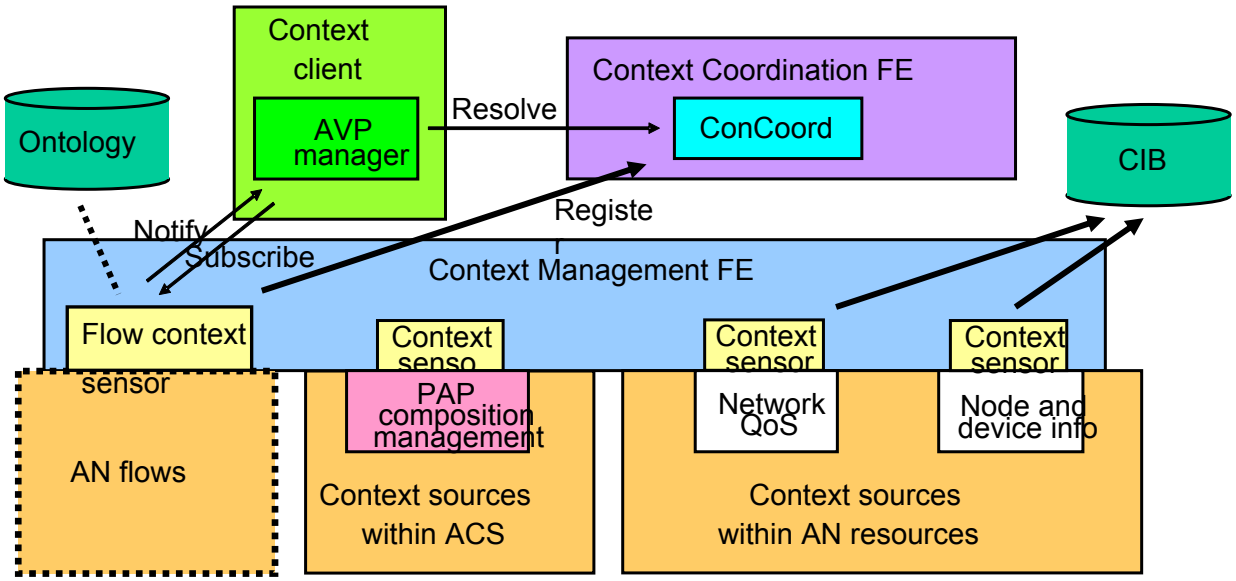


Figure 29 – Integration of context sensors in demo prototype

Figure 29 shows the positioning and integration of the various context sensors within the proof-of-concept demonstration prototype. It also shows some areas that are under active development, such as the integration of flow context sensors, further development and integration with the context ontology, support for the ContextWare primitives, and interoperation with other ContextWare functional elements such as the ConCoord.

2.2.5.6 Network sensor applications

At the start of this section we mentioned the general role that context sensors play within the ContextWare system. In this section we mention some specific applications that demonstrate their potential use within Ambient Networks.

In the Phase 1 demonstration prototype, the P2P, QoS and node context sensors were used to trigger the adaptation of the Ambient Virtual Pipe [27] in response to changes in network context, particularly within the peer-to-peer management overlay and QoS conditions. Our experience in the early prototype shows that we can extend the use of context sensors for the contextualisation and management of service overlays in general (i.e. SATOs). Figure 30 for instance illustrates the use of sensors distributed both within an overlay and in an underlying network to provide input for the computation of the cost of a particular service path.

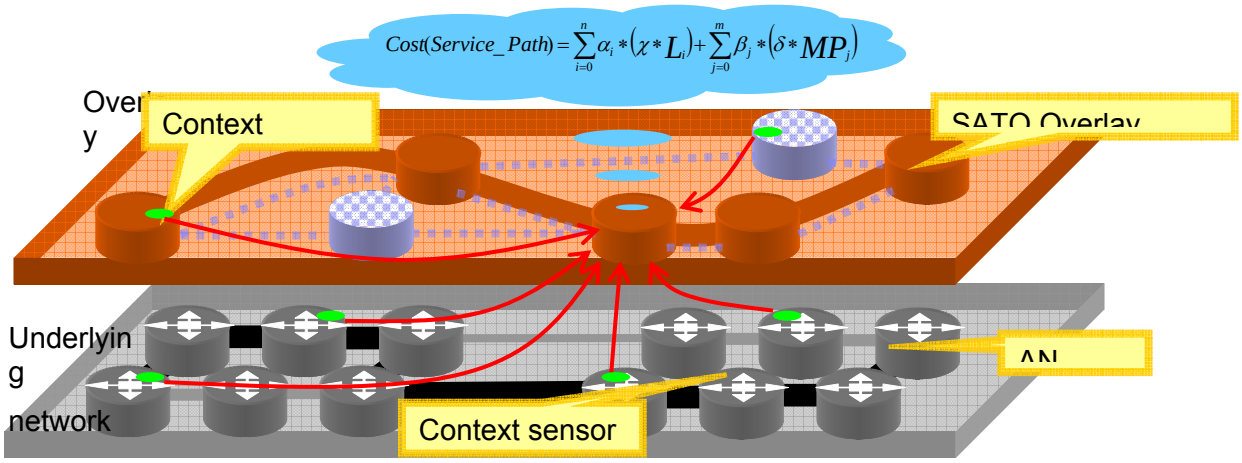


Figure 30 – Use of context sensors in overlay routing

Aside from demonstrating the use of sensors for network and service self-adaptation [27], we have also demonstrated their use in various other scenarios, such as in triggering network services [29], implicit QoS signalling [30], mobility support [31], and in context-based flow classification [32]. We have likewise outlined and are currently exploring their possible uses in content delivery, in mitigating network attacks, and in controlling malicious or resource-wasting flows such as spam and spit [28].

2.2.6 Performance evaluation

The performance of the AN Management Model has been studied with respect to the efficiency of context dissemination. A detailed description of the study and the results is given in A7.2. Two different architectures are considered:

- A centralised architecture where context information is transmitted from the sources to the ContextWare, which performs some processing on the data and notifies a given set of context clients.
- A decentralised architecture where the context sources notify the context clients directly without the involvement of the ContextWare.

In this study, the internal processing in the ContextWare is, for simplicity, represented by a single node called Mediator The context dissemination process has two time consuming operations:

- Policy enforcement (PE): Filtering, scale conversion, derivation of higher level context, etc.
- Transmission: The time it takes to transmit context information over an outgoing link.

Both the policy enforcement and the transmission are modelled as queuing systems. Using results from queuing theory, the performance of the centralised and decentralised architectures are evaluated. In this evaluation, some minor approximations were made, and the choice of parameter values will impact on the results, yet we believe that our investigations cover sufficiently large parameter ranges to give valid support for the following conclusions.

- The centralised architecture is the best choice for low load (low intensity of context messages). A centralised node (the Mediator, i.e. ContextWare functions) will require slightly more capacity than the corresponding sources, and may hence be used to reduce overall system costs. However, as the load increases, the

centralised architecture will experience scaling problems. The number of context sources has more impact on the performance than the number of context clients.

- The decentralised architecture scales far better, and will e.g. never reach saturation as the number of context sources increases.

2.3 Decentralized Real Time Monitoring FE

The ability to provide continuous estimates of management variables is vital for many management tasks, including network supervision, quality assurance, and proactive fault management. Often, management variables that are monitored in these tasks are aggregates that are computed from device variables across the network using functions such as SUM, AVERAGE, MIN, and MAX. Sample aggregates are the total number of VoIP flows, the maximum link utilization, or a histogram of the current load across routers in a network domain. While it is often crucial to know how accurate such estimates are, network management solutions deployed today usually provide only qualitative control of the accuracy and do not support the setting of an accuracy objective.

Engineering continuous monitoring solutions for network management involves addressing the fundamental trade-off between accurate estimation of a variable and the management overhead in terms of traffic and processing load [8].

Here we address the problem of continuous monitoring with accuracy objectives for large-scale network environments. Our goal is to achieve an efficient solution that allows us to control the accuracy of the estimation.

We introduce A-GAP, a generic aggregation protocol with controllable accuracy. A-GAP allows for continuously computing aggregates of local variables by (i) creating and maintaining a self-stabilizing spanning tree and (ii) incrementally aggregating the variables along the tree (Figure 31). It is push-based in the sense that changes in monitored variables are sent towards the management station along the aggregation tree. The protocol controls the management overhead by filtering updates that are sent from monitoring nodes to the management station. The filters periodically adapt to the dynamics of the monitored variables and the network environment. All operations in A-GAP, including computing the aggregation function and filter configuration, are executed in a decentralized and asynchronous fashion to ensure robustness and achieve scalability.

We developed a stochastic model of the monitoring process, which allows us to compute the filter widths as the solution to the problem of minimizing the management overhead for a given estimation error. A distributed heuristic solution to this problem is implemented in A-GAP. A distinctive feature of this approach is that it provides us with an estimation of the error distribution at the management station in real-time.

There are recent results available on monitoring network variables with accuracy objectives [8][10][13][14]. All these works, however, express the accuracy in terms of the maximum estimation error, which is often at least an order of magnitude larger than the experienced error, and its use in practical scenarios therefore questionable. For this reason, we advocate controlling the average estimation error. Furthermore, our protocol can provide an estimation of the maximum error if needed, while the other schemes cannot provide any estimation of the experienced error. Regarding other differences to related research, the scheme proposed by Olston et al. [8] is centralized, while ours is decentralized. Many schemes have been recently proposed in the context of wireless sensor networks [10][13][14] and are not directly applicable to network management systems. The main objective of those works is on extending the life time of the network, mainly through minimizing the amount of information exchanged among nodes.

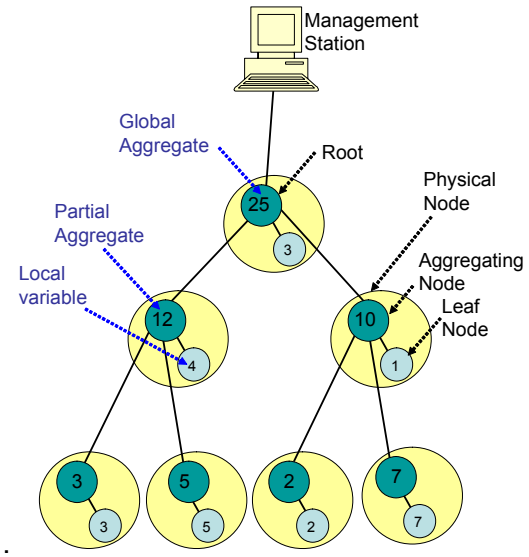


Figure 31 – Example of an aggregation tree with aggregation function SUM

2.3.1 The Problem: Real-time Monitoring with Accuracy System Architecture.

This work assumes a distributed management architecture, whereby each network device participates in the computation by running management processes, either internally or on an external, associated device. These management processes communicate via a *management overlay network* for the purpose of monitoring. We also refer to this overlay as the *network graph*. A node in this graph represents a management process.

The aggregation tree shown in Figure 31 spans the management overlay. Inside each management process runs a leaf node and an aggregating node of this aggregation tree.

The topology of this overlay can be chosen independently from the topology of the underlying physical network. We have considered two types of overlay topologies [11]. First, the case where the management overlay has the same topology as the underlying physical network; second, a class of grid topologies, where most nodes have four overlay neighbours.

Problem Statement. We consider a dynamically changing network graph $G(t) = (V(t), E(t))$ in which nodes $n \in V(t)$ and edges/links $e \in E(t) \subseteq V(t) \times V(t)$ may appear and disappear over time. Each leaf n has an associated local variable $w_n(t)$. The term local variable is used to represent a local state variable or device counter that is being subjected to monitoring. Local variables are updated asynchronously with a given sampling rate.

The objective is to engineer a protocol on this network graph that provides a management station with a continuous estimate of $\Sigma_n w_n(t)$ for a given accuracy. The protocol should execute with minimal overhead in the sense that it minimizes the (maximum) processing load over all nodes. The load is expressed as the number of updates per second a node has to process. The accuracy is expressed as the *average error* of the estimate over time.

Throughout the section we use SUM as aggregation function. Other functions can be supported as well, as discussed in [11].

2.3.2 A-GAP: A Distributed Solution

A-GAP, the protocol presented here, is based on GAP (Generic Aggregation Protocol), which we developed in our earlier work [2]. GAP is an asynchronous distributed protocol that builds and maintains a BFS (Breadth First Search) spanning tree on an overlay network. The tree is maintained in a similar way as the algorithm that underlies the 802.1d Spanning Tree Protocol (STP) [17].

In GAP, each node holds information about its children in the BFS tree, in order to compute the partial aggregate, i.e., the aggregate value of the local management variables from all nodes of the subtree where this node is the root.

GAP is event-driven in the sense that messages are exchanged as results of events, such as the detection of a new neighbor on the overlay, the failure of a neighbor, an aggregate update or a change in the local management variable. A drawback of such an approach is that it can cause a high load on the root node or on nodes close to the root, specifically in large networks. In order to reduce this overhead, one can either apply a rate limitation scheme, which imposes an upper bound on message rates on each link, or one can introduce a filter scheme, whereby a node drops updates when small variations of its partial aggregate occur. Note that, for both schemes, the overhead is reduced at the cost of introducing an error in estimating the aggregate.

For A-GAP, we choose a filter scheme. When a partial aggregate (or local variable) of a node *n* changes, then *n* sends an update to its parent if the difference between the value reported in its last update and the current value exceeds the local filter width F^n. See [11] for a detailed description of local filters.

The Optimization Problem. Estimating the network variable at the root node with minimal overhead for a given accuracy can be formalized as an optimization problem.

Let *n* be a node in the network graph, ω^n the rate of updates received by node *n* from its children, F^n the filter width of node *n*, E^{root} the distribution of the estimation error at the root node, and ε the accuracy objective, specified by the operator. We formulate the problem as

$$\text{Minimize } \underset{n\in N}{Max}\{\omega^n\} \text{ s.t. } E(|E^{root}|) \le \varepsilon \qquad \text{(eq.1)}$$

whereby ω^n and E^{root} depend on the filter widths $(F^n)_n$, which are the decision variables. *N* is the set of nodes in the network graph.

We have developed a stochastic model for the monitoring process, which describes individual nodes in their steady state and is based on discrete-time Markov chains. Local variables are modeled as random walks. The model is described in detail in a long version of this section [11]. For each node *n*, the model relates the error of the partial aggregate of *n*, the step sizes (which indicate changes in the partial aggregate), the rate of updates *n* sends and the width of the local filter. In a leaf node, the step size represents the changes of the local variable. The model permits us to compute the distribution of the estimation error at the root node E^{root} and the rate of updates ω^n processed by each node.

An optimal solution to (eq.1) can be computed using a (centralized) grid search algorithm, a well-known optimization technique. Such an approach, however, is not feasible for large networks, since the computational cost of this algorithm grows exponentially with the number of nodes. A-GAP realizes a distributed heuristic, which attempts to minimize the maximum processing load on all nodes by minimizing the load within each node's neighborhood. A-GAP maps (eq.1) onto a local problem for each node n as follows:

$$\text{Minimize } \underset{\pi}{Max}\{\omega^\pi\} \text{ s.t. } E(|E^n|) \le \varepsilon^n \qquad \text{(eq. 2)}$$

This means that node *n* attempts to minimize the maximum load in a neighborhood π for a given accuracy objective ε^n of its partial aggregate. The node attempts to solve (eq.2) by periodically re-computing the filters and accuracy objectives of its children, based on our model. Re-computing the filters $(F^c)_c$ allows node *n* to influence its own load ω^n, while re-computing the accuracy objective ε^c of a child c allows the node to influence the load ω^c on *c*. For a detailed description of the realization of A-GAP, see [11].

The two planes of A-GAP. A-GAP can be understood as operating in two different planes. On the data plane, updates of the local variables are propagated towards the root and filtering occurs. In this plane, information flows bottom-up, from the leaves towards the root.

In the control plane the filters are computed. In this plane, information flows in both directions, bottom-up and top-down. The model variables (step sizes and errors) flow from the leaves towards the root and are incrementally aggregated. The filters are distributed top-down, from the parents to their respective children.

2.3.3 Evaluation through Simulation

Simulation setup and scenario. We have evaluated A-GAP through extensive simulations using the SIMPSON simulator [6]. The results presented here are based on simulating A-GAP for two different types of overlay topologies. First, we consider an overlay that follows the physical topology of Abovenet [7], an ISP, which has 654 nodes and 1332 links. Second, we use grid overlay topologies with 25, 85, 221 and 613 nodes. These grid topologies are built in such a way that each node has 4 neighbors, except for the nodes at the edges of a grid, which have 2 neighbors, and the four corner nodes of a grid, which have 1 neighbor.

All evaluation scenarios share the following settings. Link speeds in the overlay are 100 Mbps. The communication delay is 4 ms, and the time to process a message at a node is 1 ms. For these simulations, the local control cycle in A-GAP is set to τ=1 sec,

The local management variable in the simulation experiments represents the number of HTTP flows traversing a given node. The local variables are updated asynchronously, once every second, based on packet traces captured on two 1 Gbit/s links that connect University of Twente to a research network [12]. The monitored aggregate is the number of HTTP flows in the network. (In the Abovenet scenarios, the aggregate is in the order of 20.000 flows.)

All simulation runs start with an initialization phase of A-GAP, which takes some 30 seconds simulation time, followed by the setting of the accuracy objective ε and a transient period of about 25 seconds, followed by the measurement period of 200 seconds.

The trade-off between estimation accuracy and protocol overhead. We have measured the protocol overhead (i.e., the maximum number of processed updates across all nodes) in function of the experienced error. Figure 32 shows the measurement results for grid topologies of different sizes. Every point in Figure 32 corresponds to a simulation run.

As can be seen, the overhead decreases monotonically, as the estimation error increases. For small errors, the load decreases faster than for larger errors. Consequently, the overhead can be reduced by allowing a larger estimation error. For example, an error of 35 flows reduces the load by 95%.

As expected, the overhead can be reduced by allowing a larger average estimation error. For example, compared to an error objective of 0 (which results in an achieved error of 4.5, see below), allowing an error of 3 flows (achieved error 5.4) reduces the load by 32%. An error of 10 flows (achieved error 10.7) reduces the load by 80%.

Let us mention in this context that the overhead incurred by A-GAP does not depend on the value of the aggregate, but on the absolute changes to the aggregate. This means that two scenarios with the same absolute changes result in the same overhead, independent of the value of the aggregate. This is why here we consider the absolute error as control parameter rather than the relative error. That said, A-GAP can easily be extended for using the relative error as control parameter.

For all measurement points, we find that the measured error is at most about 4.5 above the error objective. We explain this by the fact that updates from different nodes in the network

experience different delays for reaching the root, which distorts the evolution of the estimate at the root node. It is feasible to give a bound for this difference between the measured error and the objective in real-time, based on the evolution of local variables and the processing/communication delays, for the purpose of tuning the protocol at run-time.

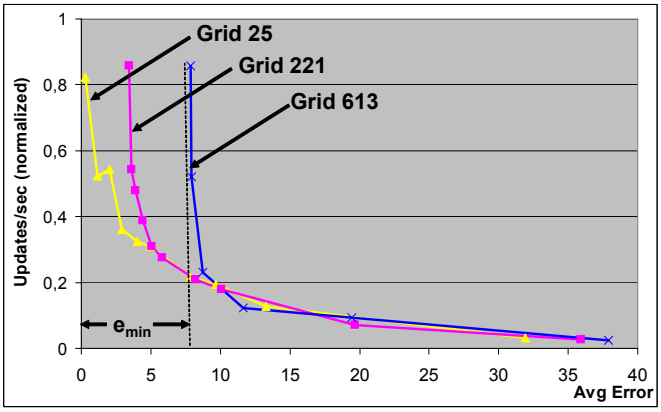


Figure 32 – Management overhead (normalized by the overlay size) incurred by A-GAP as a function of the accuracy of the estimation for grid overlays of different sizes

Handling topology changes and failures. We have analyzed the adaptability of A-GAP to the failure of an individual node. If a node fails, the aggregation tree is reconstructed, which may change the relative position of nodes or subtrees in the spanning tree. At the same time, the partial aggregates in some nodes are recomputed. The local mechanism for filter re-computation assures that, during a transient period, the filters in the nodes adapt to the new tree structure.

When A-GAP adapts to a node failure, we observe spikes of short duration in the estimation error. The size and duration of these spikes depend on the position of the failing node, but not on the accuracy objective. Second, we observe a transient increase in the protocol overhead when a node fails. The overhead increase and the duration of the transient depend on both the position of the failed node and the accuracy objective. The duration of the transient for overhead is generally much longer that that for the estimation error.

Distribution of the estimation error. Figure 33 shows the predicted error distribution computed by A-GAP and the actual error measured in a simulation run, for an error objective of 8 for the Abovenet overlay topology. The vertical bars indicate the average actual error.

As we can see, the predicted error distribution is close to the actual distribution. More importantly, the distributions have long tails. While the average error in this measurement period is 8.76, the maximum error during the simulation run is 44 and the maximum possible error (that would occur in an infinite measurement period) is 70. Based on this observation, we argue that an average error objective is more significant for practical scenarios than a maximum error objective, as suggested by other authors [8][10][13][14][15][16].

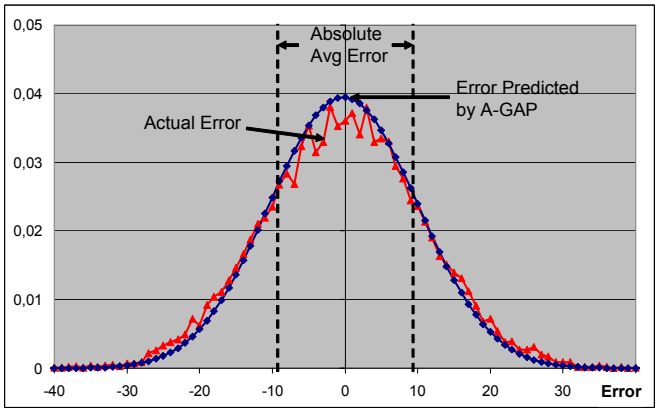


Figure 33 – Distribution of the error predicted by A-GAP and the actual error at the root node

2.3.4 Discussion

We have presented A-GAP, a protocol for continuous monitoring with accuracy objectives. A-GAP follows the push approach to monitoring and uses the concept of incremental aggregation on a spanning tree. A-GAP is decentralized and asynchronous, two key properties for achieving robustness and scalability.

The protocol dynamically adapts to changes in network topology and to node failures, as well as to changes in the statistics that are computed from the local management variables. We consider management protocols that exhibit this kind of autonomic behaviour essential for future communication systems.

The simulation results show that we can effectively control the *trade-off between estimation accuracy and protocol overhead*. When allowed a modest error in estimating an aggregate, *A-GAP reduces the overhead significantly* compared to an approach where all changes in local variables are reported. In one scenario (Figure 32), the overhead is reduced some 95% for an accuracy objective of 35. (As we pointed out, the protocol overhead does not depend on the value of the aggregate.)

A-GAP can be extended to give *performance prediction* at run-time. Based on our stochastic model, a manager can be provided with the distribution of the estimation error and the expected overhead for other potential accuracy objectives under the current network conditions. Similarly, the manager can be provided with the distribution of the estimation error that can be expected for a maximum allowed overhead under the current network conditions.

Since A-GAP provides the error distribution at the root node, it can support *other accuracy objectives* other than the average error (or the maximum error). For instance, instead of controlling the average error, a manager could set an interval that contains the estimation error at the root with a probability of 97%.

Although we have used SUM as the only aggregation function throughout this section, *other aggregation functions*, such as AVERAGE, MIN, and MAX can be supported with straightforward modifications. In general, aggregation functions which are composed of functions that are both commutative and associative can be supported in A-GAP.

2.4 Self configuration FE

Self-management is an important requirement towards Ambient Networks. This implies that configuration tasks are performed in a decentralized way. Currently there is no general component that handles self-configuration in the ACS. The approach adopted, instead, is that different configuration tasks are performed independently by different FEs. On the other hand, decentralized configuration requires in general some mechanisms to

accomplish correctly their configuration tasks. Stability and oscillations avoidance are the major issues that must be tacked in a distributed scenario.

The Self-configuration FE provides general services to other FEs to correctly accomplish their configuration tasks. Its main purpose, currently investigated, is to guarantee stability of a certain configuration over time and to avoid oscillations.

First, we investigate the specific problem of self-configuration of wireless base stations. Then we try to generalize the problem based on experience with wireless base station self-configuration and sketch the architecture of a generic self configuration FE.

2.4.1 Plug-and-Play base stations

Wireless base stations require a number of configuration steps to build an operative access network [66]. From a high level perspective, one should first of all configure the radio resources of the base stations and set up a service bearer to mobile terminals. These operations build the basic service of a wireless network and they constitute the layer 2 configuration. After this preliminary step, the wireless network is able to provide services to users and consequently its working conditions may change with time; the instantaneous load served by the base station is an example. To maintain a high quality of service, it is necessary to adapt the configuration of the base stations to these changes, running a continuous “optimization” process that takes in account the changes in the network.

The self-configuring base stations work on a collaborative behaviour by exchanging information about their current status. Each base station uses the management information controlled locally and the information gathered by the other base stations to control the local configuration. The decisions are generated by self-configuring algorithms designed for specific management domains; for instance, the configuration of power is under control of the module performing load balancing. The information is propagated in the network according to a paradigm that trades off between timeliness of the information distributed in the network and communication overhead.

A new base station plugged in the network scans the radio environment and builds the relationships with neighbouring base stations. With this step, the base station gets knowledge about the existing infrastructure, establishes communication with the neighbouring base stations and configures the radio channel. In particular wireless channels cannot be allocated arbitrary, because bandwidth overlapping might occur. This can lead to conflicts over the wireless medium when two sources are transmitting at the same time in the overlapping band and as a result the wireless transmission deteriorates; from user’s perspective this can lead to frequent frame retransmissions or even to service outage.

Since channel allocation is a fundamental management task in wireless networks, a specific self-configuring module is dedicated. The algorithm takes as input the channels scanned in the radio environment and chooses the most suitable channel for the base station.

2.4.2 Power control

The load balancing function aims at maximizing the usage of radio resources in a wireless network. Since users often appear unevenly in a wireless network, the load served by the base stations can result unbalanced. This disparity produces a degradation of the service to users; on the extreme case, users will be rejected by the access control of a base station, even if potentially some spare resources are available on the neighbour base stations. The load distribution in a wireless network is out of control of the management system, but it is bound to the behaviour of the users. Therefore a management function is needed in the base stations to ameliorate the load distribution [57].

Detailed simulation results show that a load balancing algorithm can effectively improve resource utilization in a WLAN hotspot [58]. The metrics adopted in the results are the

number of users rejected and the number of handovers occurred. From a management perspective, it is interesting to investigate how the decisions of the algorithm are performed and how they can be controlled. Figure 34 shows the change of power levels over time measured in a simulation of a realistic scenario. The thin lines report the results measured with a first version of the algorithm. The graph shows a typical problem in self-configuring algorithms, i.e. stability of values: the value of the power changes very frequently and in some cases oscillates between opposite values. To mitigate this instability, a mechanism to block opposite decision has been added to the algorithm. This mechanism poses the basis for a more generic approach, as explained in the next paragraph.

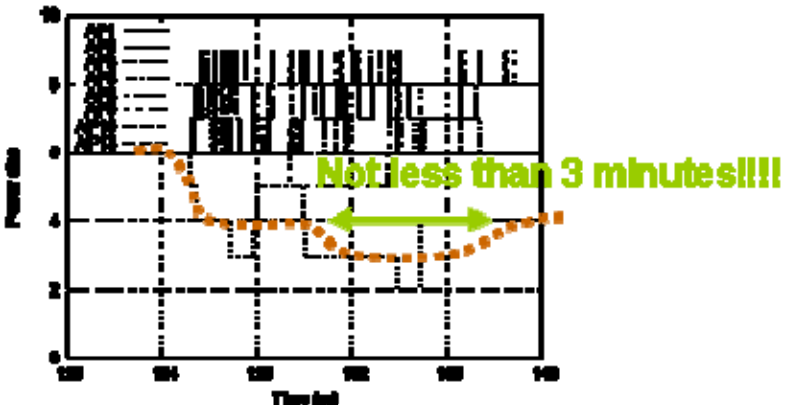


Figure 34 – Behaviour of power control

2.4.3 Generic control of self-configuring algorithms

The behaviour of the above load balancing algorithm is very generic and affects any self-configuring algorithm in general. Therefore, the self-configuring FE can be delegated to guarantee stability of a certain configuration domain. The properties of the self-configuring FE comes as a generalization of the results achieved in the deployment of the plug-and-play base stations.

Stability is achieved with different actions, depending on the type of the information that is being configured. In fact, a distinction is necessary on the type of management information under control:

- domain with non-measurable values: the configuration is identified by a certain state. Only the concept of stability can be defined in this domain.
- domain with measurable values: these values can be measured and in particular an order exists between two values. The concepts of stability and oscillations can be defined in this domain.

With respect to a self-configuring algorithm operating on management domain with measurable values, one can try to stabilize the output configuration with the aid of an average function. The problem is that it is difficult to judge whether an average is really the optimal instrument to achieve an optimal goal in the configuration and how the average should be computed anyway (e.g. the weights of each contribution). A more advanced approach relies on instruments of the control theory. For example Figure 34 shows the expect behaviour of a power control algorithm. Changes in this domain affect directly the end service to users and user might experiment service interruption whenever a change is applied. Since the final goal is to increase the Quality of Experience (QoE) of users, stability should be achieved for a length of period enough to avoid interruption noticeable to users; specific values should be defined by administrators through policies, but in this example values of few minutes are reasonable.

The self-configuring FE provides then generic algorithm to guarantee stability of measurable and non-measurable values for self-configuring algorithms implemented by

different FEs, like shown in Figure 35. For example if stability of the power level needs to be stabilized for a certain period, the self-configuring FE provides to filter the behaviour of a load balancing algorithm: if an average mechanism is employed, then the zero of the filter is set accordingly to achieve the desired stability. Further generalization to other algorithms are possible and for the moment left for future study.

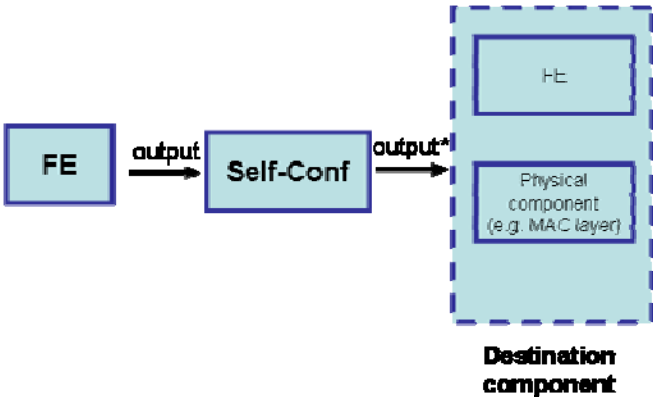


Figure 35 – Integration with other FEs

2.5 Composition Management FE

This section introduces the Composition Management FE and its interactions with the Composition FE, other FEs and the ACS framework.

The Composition Management FE manages the logical structure of the ACS as well as FE instances in this structure. This logical structure is created through network composition processes. In case of network integration composition type, two ACSs are merged into one. In case of control sharing composition type, the original ACSs are kept but a higher level ACS is created which is populated by functional entities having negotiated common management in the composed AN. Finally, the network interworking composition type does not change the logical structure of ACS.

The Composition Management FE is responsible for performing the above changes during the realization phase of network composition. This includes the modification of the logical structure of the ACS and the creation, deletion or modification of FE instances accordingly.

First, we analyse relationships between FE instances of the same type in a composed network. Then we present interactions and functionality of the Composition Management FE during the composition realization phase.

2.5.1 Relationships between FEs in composed networks

In one ACS, one functional entity may be present in multiple instances at multiple nodes for two reasons. First, some very basic functional entities need to be running at all nodes of the ACS, e.g. connectivity management. The second reason is robustness: if one node fails or leaves the network, FEs running at this node can be replaced by FE instances at other nodes.

For all FE instances of the same type in an ACS, there exists a dedicated FE instance called master FE. Whenever the FE is contacted from a different AN, name resolution automatically returns the URI of the corresponding master FE. Other FE instances may also be active, but they only participate in intra-ACS communication. When two ANs compose by network integration, the two ACSs are merged into one and it is necessary to replace the two master FE instances by selecting a single master FE instance (e.g. one of the two master FE instances of the composing networks). In case of network interworking, the logical structure of the network does not change and master FE selection can also remain untouched.

Relationship of FE instances of the same type in case of control sharing type of composition is more complicated. Control sharing means that some resources in the composing ANs will be (partially or entirely) under common control while others continue to be managed separately by the two networks. Common management can be represented visually as a higher level virtual ACS populated by FE instance under common control. To describe relationship of FE instances in the original ACSs and the virtual higher level ACS, three different delegation models have been proposed:

- partial delegation (see Figure 36)
- full delegation (see Figure 37)
- no delegation (see Figure 38)

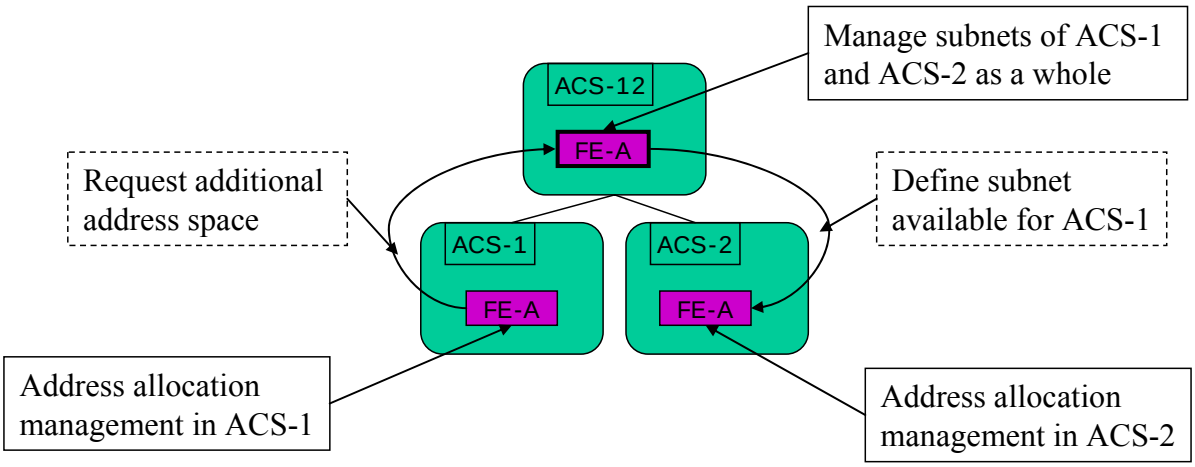


Figure 36 – Partial FE delegation (example)

Partial delegation means that upper level instances of FEs are working at higher abstraction levels. Taking the example of a fictive "address management" FE (Figure 36), this means that FE instances at the bottommost level are responsible for managing individual addresses inside an address domain, while the upper level FE instance manages subnets as a whole (higher abstraction level). The upper level FE instance is called master[4] while the lower level instances are slaves. It is up the specific FE to define semantics of this abstraction and it is also possible to define management primitives for interactions between master and slave instances. Coming back to the previous example, requesting additional address space from the master or ordering the slave FE to redefine an address domain due to address conflicts could be such management primitives.

[4] The meaning of the term master is used here to describe relationship of FE instances at different levels in a composed ACS. Inside a simple ACS, the term master denotes relationship between FE instances at the same level.

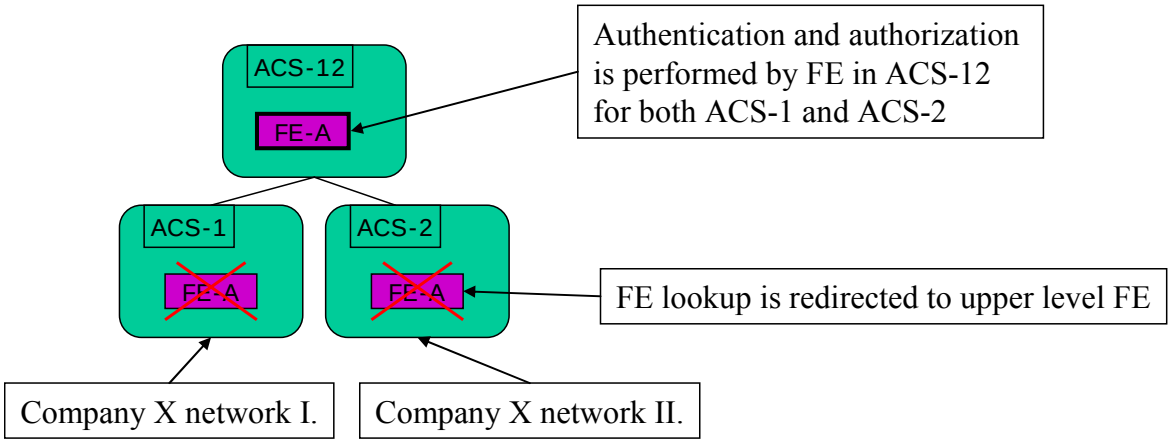


Figure 37 – Full FE delegation (example)

Full delegation means that entire functionality of slave instances is delegated to the master instance. This means that resources managed by the FE are under common control of the upper level ACS. In this case, it is not even necessary to have the FE instantiated in lower level ACSs, FE lookups can be redirected by the ACS framework to the upper level FE. Figure 37 shows a security example for full delegation. Assuming that nodes of both ACS-1 and ACS-2 belong to the same administrative domain (same company), security policies might require common authentication and authorization control of the two networks.

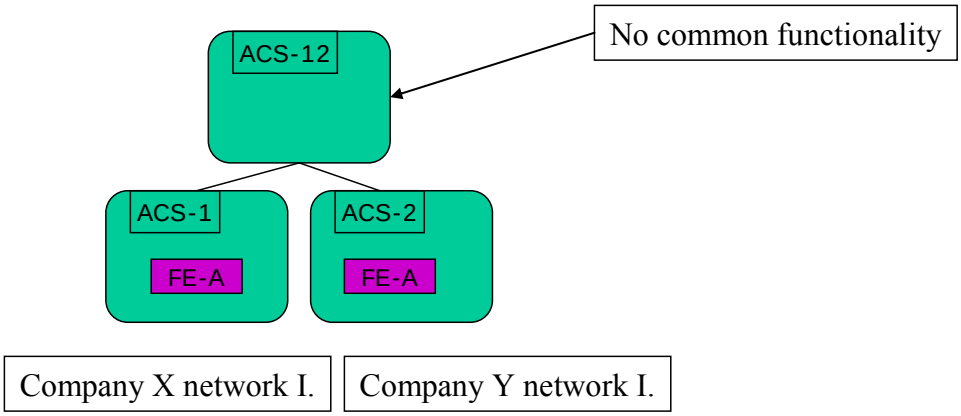


Figure 38 – No FE delegation (example)

Finally, **no delegation** is used when resources managed by an FE are not shared at all and thus common control doesn't make sense. In this case, the FE is not instantiated at the upper level ACS. Figure 38 shows a different security example for no delegation: Assuming that the two networks are owned by different companies, authentication and authorization FE should not be under common control.

Delegation type is included by each FE into the subCA created during composition negotiation. If an FE does not specify delegation type in the subCA, then no delegation is implemented by default ensuring interworking with FEs that do not support this delegation model.

2.5.2 Management Plane Composition Realization

Based on composition type decided during composition negotiation and FE delegation type extracted from subCA of each FE, the composition management FE modifies logical network structure and creates, deletes or modifies FE instance as necessary during composition realization. Interactions of Composition Management FE with other FEs during composition realization depend significantly on composition type. Figure 39 shows

interactions in case of control sharing composition type while Figure 40 shows interactions in case of network integration.

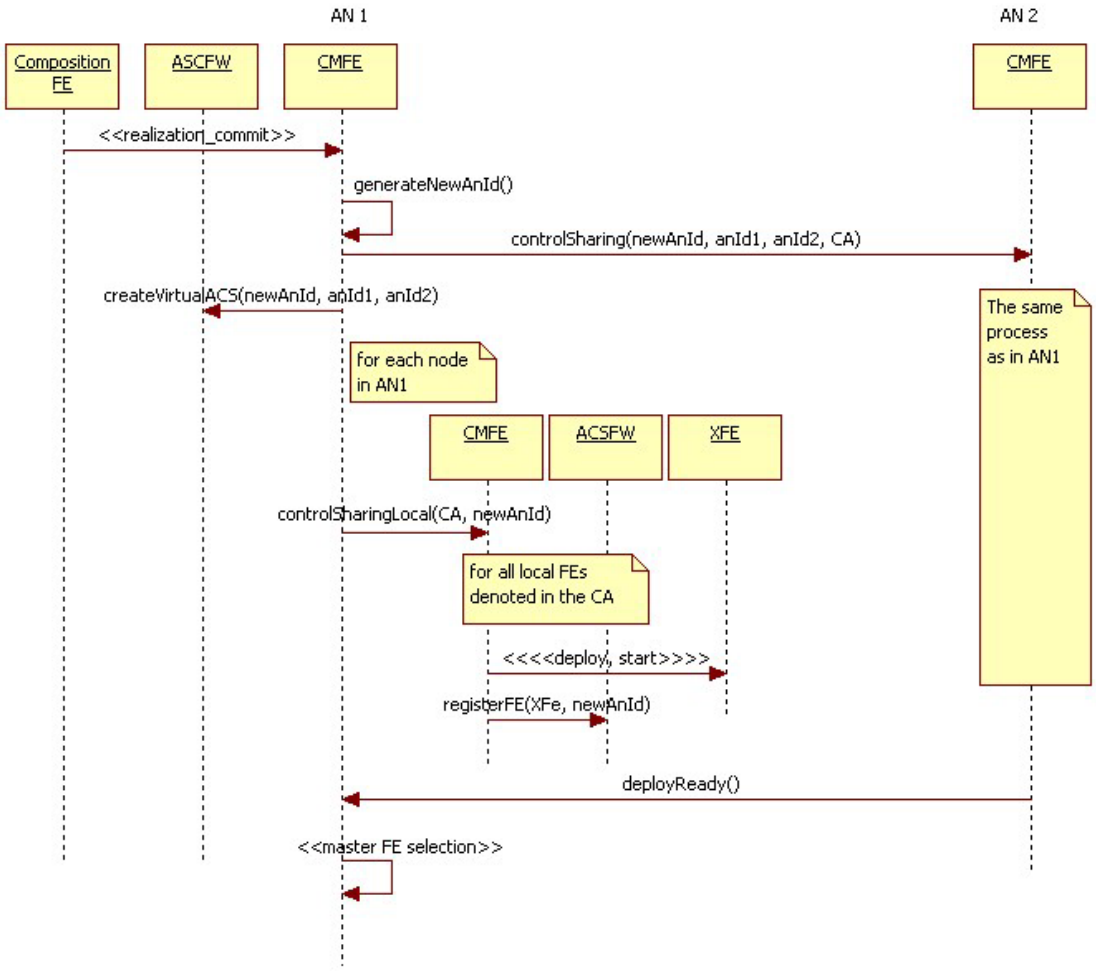


Figure 39 – Interactions of Composition Management FE with other FEs in case of control sharing

In case of control sharing, the Composition Management FE creates a new, higher level virtual ACS in the ACS registry and populates this virtual ACS by new FE instance for all those FEs that had previously negotiated FE delegation. The network integration case is simpler: One of the two ACSs is merged into the other and a new master FE instance needs to be selected for each FE type. In both cases, management plane composition realization is triggered by the realization commit message from the Composition FE. Creation of a virtual ACS, merging of ACSs, creation of new FE instance and selection of master FE instances is performed calling methods of the ACS framework.

Finally, in case of network interworking composition type, the logical network structure and FE instances are not changed thus Composition Management FE does not need to take any actions.

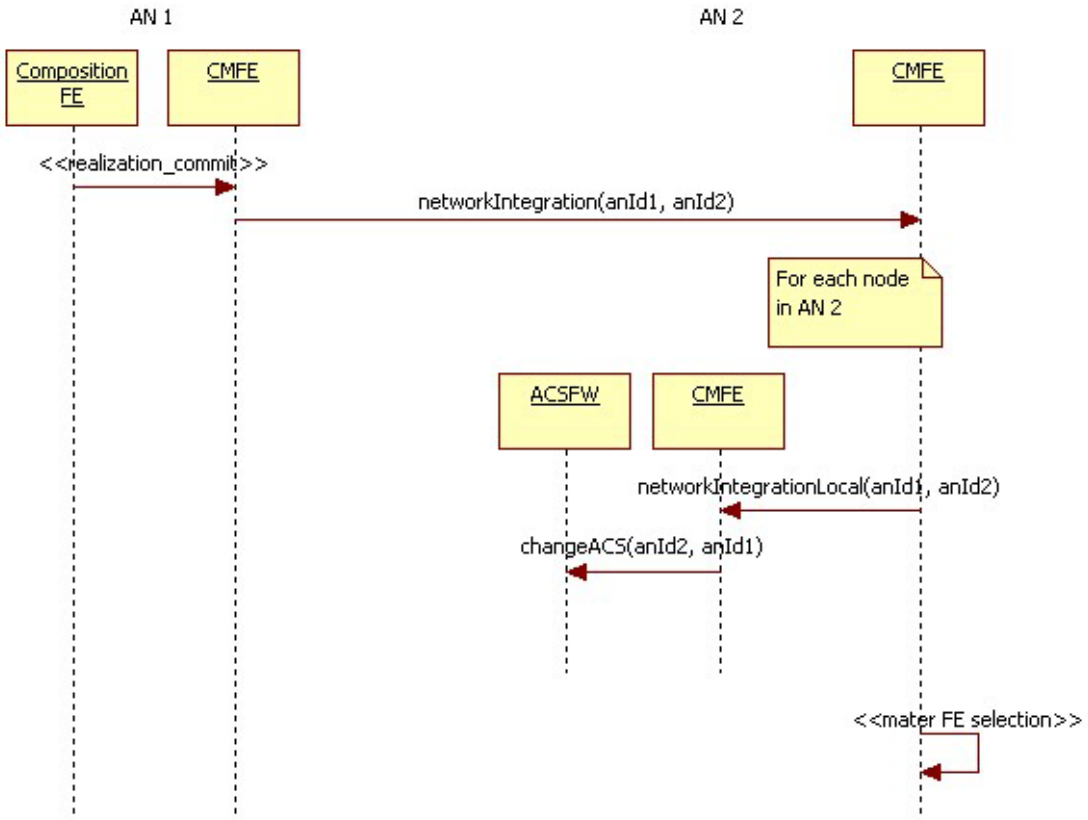


Figure 40 – Interactions of Composition Management FE with other FEs in case of network integration

2.6 Policy Management FE

Policies can take many forms including: conditional rules (e.g. if conditions then actions), event-condition-action rules, access control policies, logic specification or downloadable scripts. They can also be specified as a programming language that is interpreted by a software component. A network policy is also interpreted as a function, which takes as argument network state conditions and returns actions to be performed on the network. With this interpretation policies are rules or goals governing the network's behaviour.

Policy-based networking has been used to provide a degree of automation, by linking actions in the network to system-events. Policies permit adaptation of network behaviour without modification of the implementation. Policies introduce network management flexibility as they seek to control network behaviour using sets of high-level rules.

Access control policies define who has the right to access, use or modify resources. They can be used by the owner/manager of an AN to control the resources of an AN. They can also be used to manage access rights to resources between ANs as result of a composition.

The AN Policy Management FE is part of the AN Policy Framework. The figure below illustrates that the Policy FE is responsible for maintaining the policy sets and to evaluate policy requests from other FEs. The actual policy enforcement is done by the FEs themselves.

The next section gives a short overview of the framework components. The following sections describe some of the components in more detail and also some examples of how the framework can be used are given.

2.6.1 Policies and composition

Depending on the type of composition; network interworking, resource sharing or network integration the policy sets can be affected in two different ways.

In the case of network interworking each ACS keeps its policy sets, only adding new policies that are needed to fulfil the composition agreement. Before adding the new policies they must be checked for consistency against the existing policy set.

In the case of network integration the two policy sets of the original ACSs needs to be merged into one policy set for the new resulting ACS. When merging two policy sets they must be checked for consistency. If there are conflicting policies these conflicts need to be resolved. This can either be done automatically by a conflict resolution software that refers to some set of meta-policies (this is for further study) or by requests for human intervention from the ACS, the latter being an optional feature since the overall goal is to achieve unattended composition. When the new policy set has been created any additional policies needed, can be added.

In the case of resource sharing both of the above ways of handling the existing policy sets can apply as both the original ACSs (and their policy sets) can remain as well as a new one can be created if non-exclusive access to resources between the ACSs is allowed.

2.6.2 Components of the AN policy framework

The following describes each of these components in some detail and presents the current status of what is available as well as future development plans.

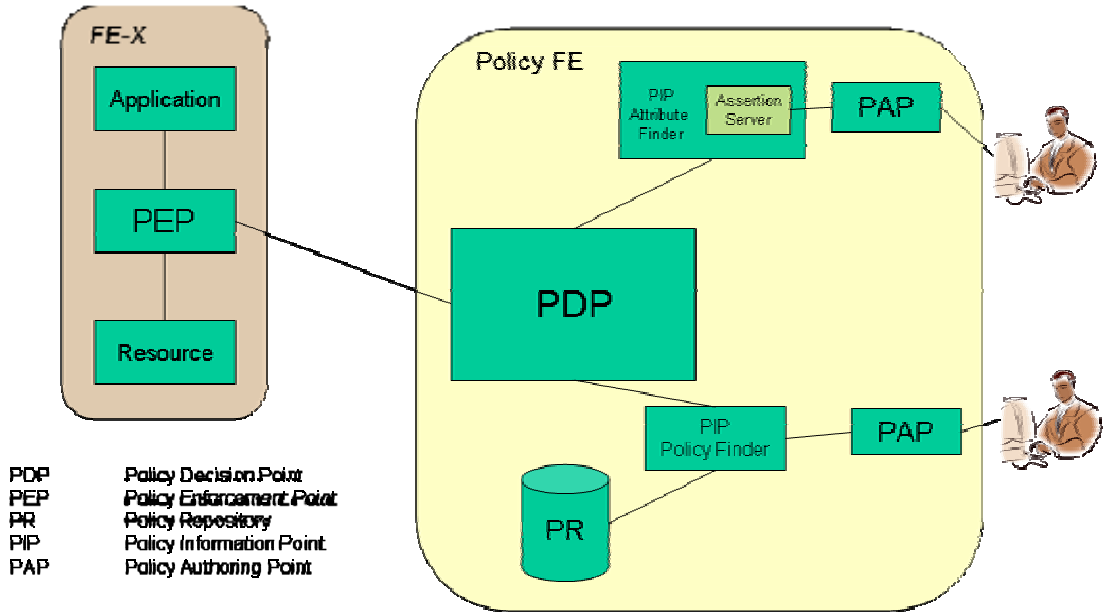


Figure 41 – Components of the basic AN Policy Framework

Policy language

The policy language used in AN needs to be standardized. The current working assumption is to use XACML 3.0, see below for more details. It was originally designed to be used for access control. Within AN it will be extended as needed to create the types of policies used within the AN ACS. These extensions will be part of future releases of the AN Policy Framework.

Policy editor

Policies can be edited using any general purpose text editor. As an alternative there is a policy editor for XACML 3.0 available from University of Murcia [107].

Policy storage

For this initial ANPF policies will be stored in plain text files. There is an initial implementation of the Policy Information Point (PIP) which implements an interface between the PDP and a text file based policy repository. The interface between the PIP and the PDP should remain stable even when the Policy Repository (PR) is implemented in a more advanced way.

Future work will investigate the use of DHTs for implementing the PR. Also other types of distributed databases might be investigated.

Policy evaluation and decision

This step is carried out by the PDP, which implements the XACML decision logic as defined in the XACML standard [97]. The Policy FE which implements the PDP, PR and PIPs has a well defined interface towards the PEPs, see section 2.6.6 below for more details.

Policy enforcement

Policy enforcement is done by the individual FEs. Most FEs are expected to include a PEP. When a policy decision is needed in an FE the PEP will send a request (resource allocation, access rights, etc.) to the PDP. When the response is received back the resulting action is implemented by the PEP.

Policy violation compensation

For obligation policies, which cannot be strictly enforced, there is a need for some type of compensation in the case a policy is violated. This issue is still under further investigation.

2.6.3 The use of XACML for AN policy framework

After evaluating a number of policy languages (including Ponder [105] and WS-Policy [106]) XACML 3.0 [97] was chosen as the policy language to be used in AN. The most important reason for selecting XACML was the availability of a stable supported implementation of a standardized policy evaluation scheme. As XACML is originally designed for access control, a simplified viewpoint, is that it currently provides yes or no answers to access requests. For policies aimed at controlling resource usage over an extended period of time, e.g. QoS provisioning, that needs continuous monitoring XACML has limited capabilities. We are currently investigating two paths for addressing this limitation:

- extending XACML to handle this type of policies, or
- complementing the use of XACML with some other policy system and creating some meta policy structure to tie them together.

The path forward will be resolved during the next year in light of the results from the first prototype.

2.6.4 Types of policies

There are two main types of policies, configuration policies and runtime policies. Configuration policies are used when an AN is initially configured. Changes to these policies will not have any effect on already configured ANs. Runtime policies on the other hand are used during the lifetime of an AN and changes to them should have an immediate effect on the operation of all concerned existing ANs.

2.6.5 Examples of how policies are used within AN

The two different categories of policies are: (1) those that are of the type of access control policies that can be answered by a simple yes/no answer, and (2) those that are controlling the usage of a resource under a prolonged period of time, e.g. a QoS Service Level Specification (SLS). The latter type requires monitoring of adherence to the policy and ways to repudiate if the agreement made is not honoured.

This second type of policies could be dealt with either by extending XACML to handle this type of policies or by adding a specialized policy language to deal with this type of policies. If the latter approach is taken then some meta level for policies is needed to ensure the consistency between the two policy systems.

2.6.5.1 Policies for access requests

To illustrate how the different entities within the policy system interwork the following example is provided, see Figure 42..

This example describes the planned implementation of access control for node-node interactions within or between ANs (AN 1 may coincide with AN 2). The realisations of node-network interactions (e.g. enrolment of a new node in an AN) and network-network interactions (e.g. network attachment) are similar from a policy management point of view. The entities involved are:

- The FEs in the respective ANs. (The procedure should preferably be described in terms of SAPs rather than FEs, with obvious changes.)
- The Nodes *a* and *b*, with cryptographic identifiers NIDa and NIDb, respectively.
- The HIP Daemons (HIPD) in the nodes, which are executing the authentication and key management protocol implemented as an AN influenced extension to HIP.
- The Node Managers (NM) enforcing the access control policies of the node. The NM acts as an XACML PEP and translates access requests mediated by HIPD to XACML requests, and responds back to HIPD.

The main difference between the node-node interaction and node-network or network-network interactions is that the node manager function is replaced by a “network manager” function embodied by the Security Domain Manager (SDM).

The Policy engine complies with the policy framework as described previously in this section.

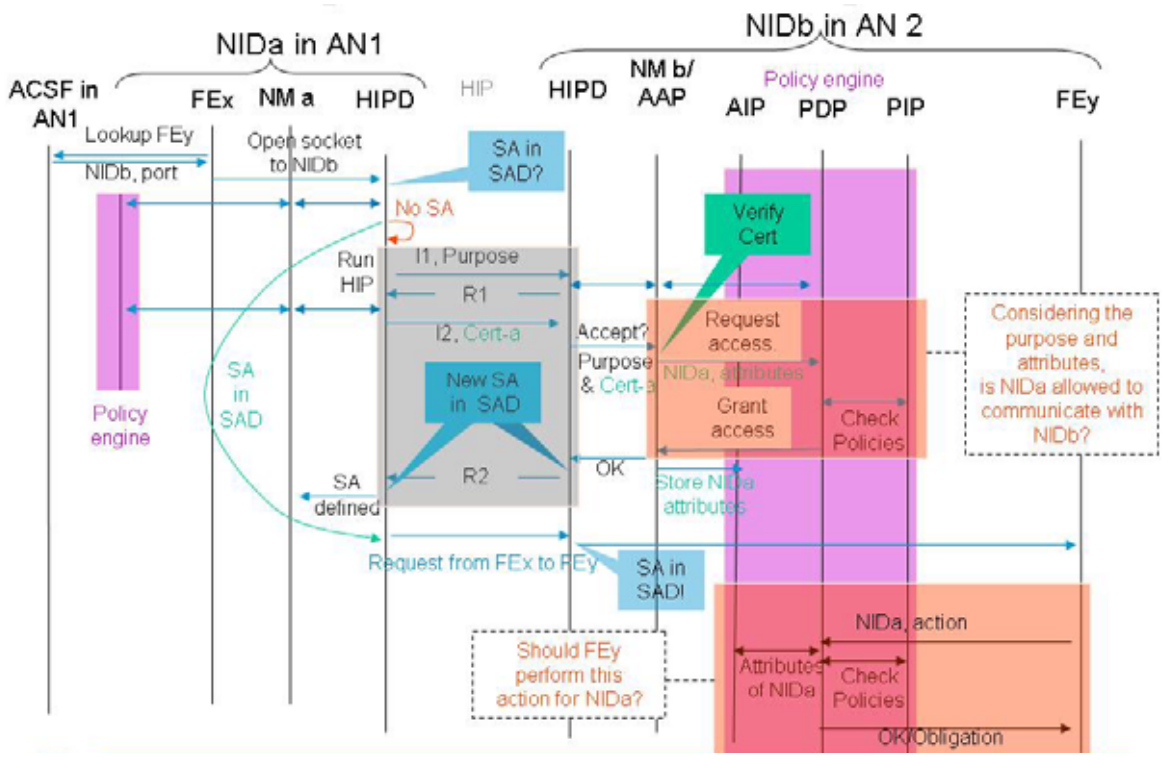


Figure 42 – FEx@AN1 interacting with FEy@AN2

This example starts with FEx wanting to send a request to FEy (as mentioned SAPs had better be used), looks up the Node ID & port number as provided by the ACS Framework and attempts to open a socket. If there is no Security Association with node *b*, the authentication and key management protocol is executed to set up an SA and forward the request to node *b* for further distribution to FEy.

Throughout the modified HIP base exchange, the HIPD consults the NM for directions, requests that are forwarded to the XACML PDP for a decision. (This applies on both initiator and responder side, but Figure 42 only details the responder.) For optimising performance, some decisions are cached in the NM. In general, the decision depends on the attributes of the subject (e.g. what relevant roles the subject has such as membership of security domain, delegate with particular privileges etc) and the access policies. Subject attribute assertions can be provided during the authentication protocol e.g. in public key certificates. (Note that the NM in addition to being PEP also performs AAP functionality by administrating attributes of the requesting subject asserted in the certificate.)

The procedure so far described enables access policy decision with the granularity of “node to node” rather than “FE to FE”. For many purposes this is probably sufficient, since legacy operating system security in general does not support such identification to prevent a malicious FE from masquerading as another FE in the requesting node. However, an additional control point can be accommodated by allowing the requested FE to perform access control on the requesting node using the same policy engine as is illustrated in the bottom right part of Figure 42. FEy thereby acts as a PEP, the attributes previously stored by the AAP may be involved in the decision. Same or different categories of policies may be applied in this decision compared to the previous. The combination enables a higher granularity for authorisation at the same policy configuration effort.

2.6.5.2 Policies for composition management

Network Composition is particularly concerned with policies for the control of resources. This is why we collect general requirements on the policy framework. Moreover policy-

related requirements to support composition as well as concrete examples of policies are given below.

Policies are involved in all types of compositions. Particularly they determine which Composition Agreement (CA) template to choose. In addition policies decide whether and how to react to a composition trigger or to advertisement and discovery messages from the NAD-FE.

For illustration some selected requirements to support composition are listed in the sequel. Firstly the Composition-FE (C-FE) must be able to contact a Policy Information Point (PIP) or Policy Repository (PR) and derive default policies. Then the C-FE must be able to derive and change policies agreed upon during the composition negotiation. This can include compensation policies for resource and service usage like AAA and mobility management service. The predefined policies may change during the composition and therefore dynamic policies must be manageable within the policy framework. If two policies overlap and contradict each other there must be a conflict resolution, e.g. via precedence of policies or other means. Otherwise the composition terminates.

Next it must be possible to group policies together in a group. Such groups could be:

- Security policies to be followed in the composed ANs.
- Policies that determine which CA template to choose, e.g. choice of CA template dependent on FEs mandatory for particular composition instance
- Policies that determine how / whether to respond to a composition trigger
- Policies that determine how to react to Advertisement and Discovery messages
- Policies that specify the default message transfer attributes (reliability, security and local processing) of each ANs GANS signalling protocol suite.

The negotiation mechanism for solving policy conflicts must run in an highly automated manner before the user is finally asked for manual interaction. Thus an automatic algorithm should be found and used. Only in case of a failure a manual negotiation should be performed to solve conflicting policies manually. All this is subject of future research.

Nevertheless in some special cases the policy conflict can be solved arithmetically. To give an illustrative example the policy

IF (user is in gold class) THEN 128kbps <bandwidth < 384kbps

conflicts with the policy

IF (bandwidth usage >= 80%) THEN do only accept any new user with 64kbps <= bandwidth <= 256Kbps.

However it is obvious that

128kbps <bandwidth < 256kbps

is a reasonable solution that solves the conflict imposed by the two policies above.

2.6.5.3 How to compose policy sets

Depending on the type of composition, network interworking, control sharing or network integration the policy sets can be affected in two different ways.

In the case of network interworking each ACS keeps its policy sets, only adding new policies that are needed to fulfil the composition agreement. Before adding the new policies they must be checked for consistency against the existing policy set.

In the case of network integration the two policy sets of the original ACSs needs to be merged into one policy set for the new resulting ACS. When merging two policy sets they

must be checked for consistency. If there are conflicting policies these conflicts needs to be resolved. This can either be done automatically by a conflict resolution software that refers to some set of meta-policies (this is for further study) or by requests for human intervention from the ACS. When the new policy set has been created any additional policies needed can be added.

In the case of resource sharing both of the above ways of handling the existing policy sets can apply as both the original ACSs (and their policy sets) can remains as well as a new one can be created if non-exclusive access to resources between the ACSs is allowed.

2.6.6 The XACML wrapper provided for the prototype

In the prototype the XACML implementation developed initially by Sun Microsystems will be used (also known as SunXACML). SunXACML is an open-source implementation being developed by a number of volunteers as a SourceForge project. The SunXACML release available from SourceForge offers XACML 1.2 support. The CVS head revision adds partial XACML 2.0 features, although no official XACML 2.0 compliant release has been made. The development team at SICS SPOT (Security, Policy and Trust Laboratory), also involved in XACML standardisation, is maintaining a patch set that updates an XACML 2.0 SunXACML snapshot to support the current version of XACML 3.0. As the XACML 3.0 specification is a work in progress, this implementation is subject to changes.

SunXACML (and consequently the SICS patch) is implemented in 100% Java. The API encompassing all XACML features however is way too complex, resulting in a steep learning curve. The goal is to provide tools for developers wishing to use policies in the prototype that allow them to get started easily. The XACML Wrapper implemented as part of the prototype has the following main goals:

- hide most of the complexity of the XACML API from developers,
- allow several clients of the Policy Framework to coexist by defining isolated policy contexts for each client, and
- provide a Web Service interface to allow seamless integration with components implemented in a programming language other than Java.

The XACML Wrapper is made up of two components, the wrapper itself, and an optional Web Service interface. Current features of the wrapper include:

- Assigning policies to isolated policy contexts as an XML document as defined in the XACML specification.
- Retrieving policies assigned to a specified policy context as XML documents.
- Evaluating requests in XML format as defined in the XACML specification. The results of the evaluation are returned as an appropriate XML document.

The following figure shows the overall architecture of the wrapper.

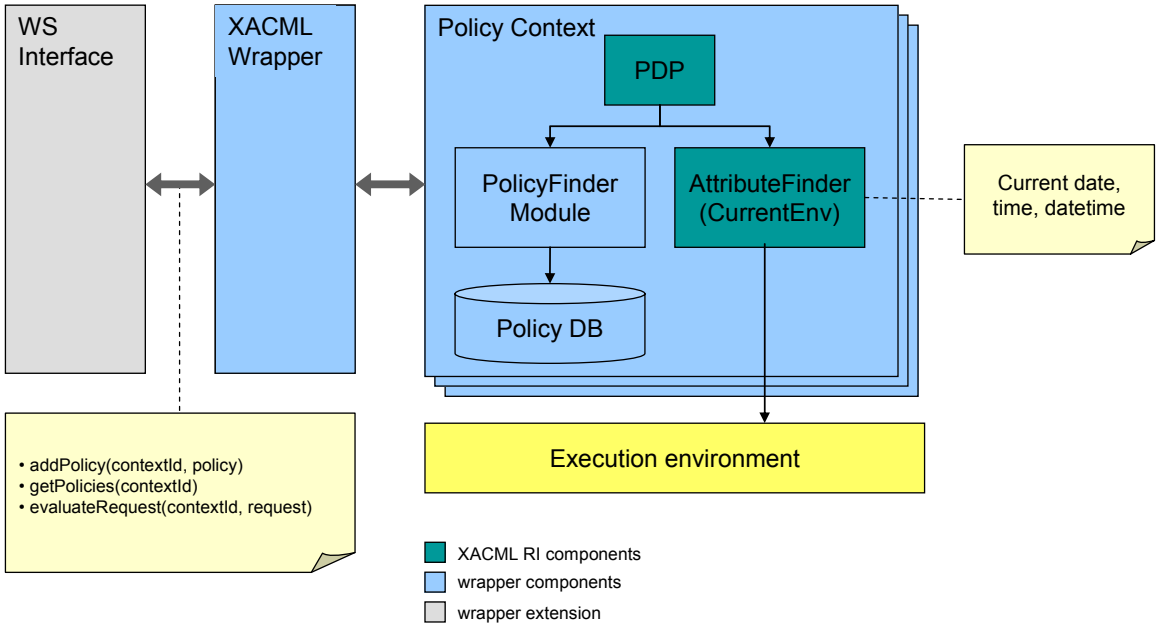


Figure 43 – Architecture of the XACML Wrapper

Future development of the wrapper will introduce a simplified XML format for defining policies and results. The XACML XML format while very powerful is rather complicated and difficult to author without proper XACML-enabled policy authoring tools. An additional feature will be a programmatic interface for building policy data and interpreting results. This interface would enable users of the wrapper to create simple policies and interpret the results quickly, without authoring and parsing XML documents. Additionally, based on this interface a GUI-based policy authoring tool could be easily implemented.

2.6.7 Policies for controlling access to context information

There will be policies that control who can access context information in the CIB. One of the subjects that will be asking for information from the CIB is the Policy system. It is not clear which rights the policy system should have in these cases. Possibly the rights should be those that the object generating the policy request has. One argument for this is to prevent subjects from retrieving context information that they do not have direct access to via indirect requests to the policy system. The disadvantage with this approach is that there might be cases where the policy system needs to have access to information, that the subjects do not have the right to access, to be able to evaluate the policies needed for a request. This is an issue that needs further study.

2.6.8 Conflict resolution

Policy conflicts can appear at different levels, there is a need to detect and resolve application-layer conflicts i.e. conflicts between different services as well as conflicts between components or hardware usage. The adaptation system must dynamically detect and resolve conflicts between simultaneously triggered policy actions.

Policy conflicts can also be resolved at different times. Some conflicts can be identified by checking the policy database for inconsistencies. This type of static policy conflicts should preferably be detected already as the policies are created and about to be entered into the policy database. The conflicts should then be resolved before the new policy is entered into the database. If such consistency control is enforced before entering new policies into the database it should be possible to keep the database free of static policy conflicts.

Other policy conflicts only occur at runtime, e.g. policies depending on context information might become conflicting due to the current status of some context information.

Another type of conflict that can occur is when policies with conflicting policy actions are being executed simultaneously.

One way of dealing with this latter type of conflicts is by doing realtime adaptation of policies. To enable this an Action Object Base (AOB) can be used, in which each action is considered as an action object that has a unique identification number i.e. an Action Object ID (AOID). Action Objects are references to adaptation code developed by SPs that are to be executed when certain conditions are reached. AOIDs are organised in a hierarchical structure to allow maximum flexibility for SPs to define their own new actions. This approach is described in more detail in the paper “Towards Flexible Service-aware Adaptation Management in Ambient Networks” [73].

2.6.9 Continued work

2.6.9.1 Consistency and coordination between policy sets

Policies are used in a number of different areas within AN. Still all FEs in the ACS need to use the common AN policy framework to ensure consistency between the different policy sets. Policies are both needed in the composition process and new/modified policies are often part of the result of a composition. When composing there is a need to check that the policies remain consistent also after the composition.

The initial approach to building a policy system will need to be pragmatic due to the complexity of this area of study.

The figure below illustrates different approaches to building a policy system. The first picture shows a policy system with unified policy machinery for all the FEs. The two other pictures show variants of policy systems having separate policy machinery for each of the FEs. The latter two approaches, while being less complex, leaves significant co-ordination task to the users/operators. While the aim is for an integrated policy solution in AN, a simpler solution might be considered in the initial prototype.

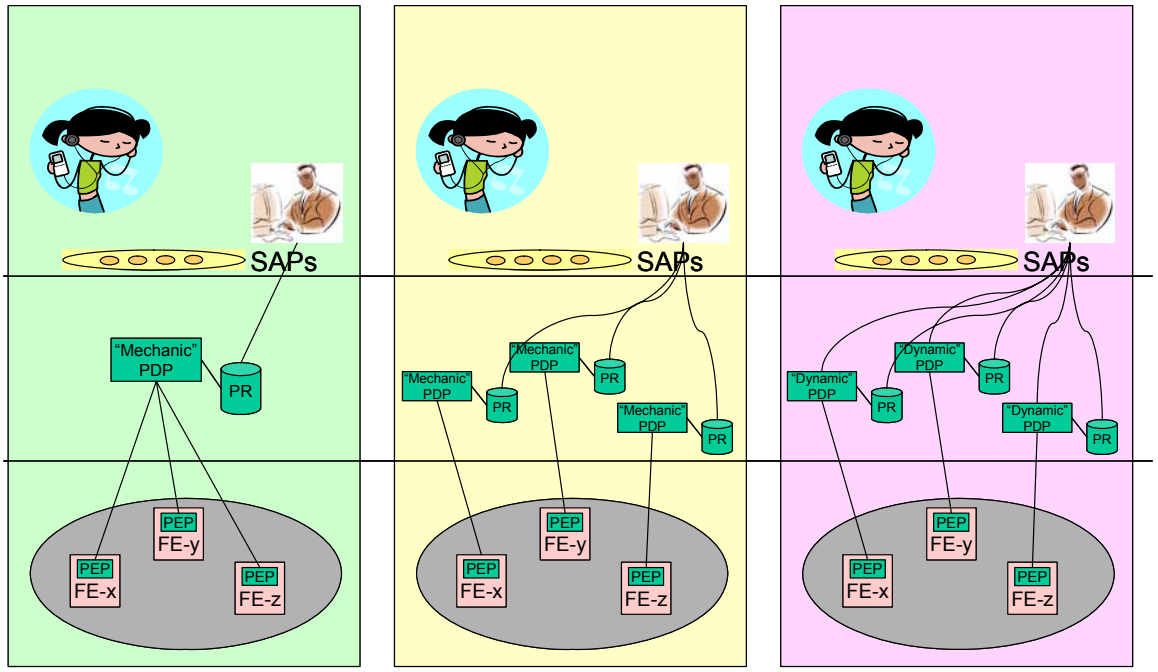


Figure 44 – Different approaches to building a policy system

2.6.9.2 Policies and performance

The normal policy evaluation process will not be fast enough for “realtime” processes, like very fast access selections. The possible alternatives include:

- caching of policy decisions (need for time-to-live stamping of policy decisions)
- using policies to configure parameters that are used for quick decisions

The first prototype will be used to get an idea of what response times can be obtained from the policy system. These results will be used to determine which functions of the ACS that should be considered “realtime”. Solutions will then be designed to cater for those special policy needs. Investigations will then be carried out to determine which synergies can be made with the mechanisms (i.e. push/pull, ConCoord, Context Managers) provided by the context management and triggering framework.

2.7 Registries Management FE

A number of functional entities need registries to store and lookup various entities: UCIs in the ConCoord FE, policy data in the Policy FE, NID router locators in the Node ID Management FE, etc. The goal of a common distributed registry is to exploit synergies and provide a unified distributed registry functionality for all functional entities. Another important objective is to achieve scalability, robustness and fault tolerance at low maintenance overhead even in dynamic network environments.

2.7.1 Design considerations

Given the dynamic nature of ambient network environments, distributed registries should be used instead of the traditional client-server based approach in most cases.

Although the required registry functionalities are very similar, the nature of entities to be stored and looked up varies in a broad range: context and policy data, services and resources offered by a node or simply a node or a user itself. Some of these entities are only defined when the node hosting them is currently part of the network. For example it does not make sense storing in a registry the locator of a service if the node offering that service is currently not accessible. Other examples are resources of a node or a user or the node itself. Let’s call these entities **node related entity**. The role of registries for such node related data is not storage but only lookup of the locator of the given entity. Other entities (e.g. network wide policy or network context information) are not strictly associated with a specific node and need to be available after the departure of the node, which inserted the entity into the registry. Let’s call them **permanent data**.

Permanent data requires a complete storage system (e.g. a DHT) not only a lookup service. Lookup services for node related entities can also be implemented using a DHT storing an ID → locator mapping for each of these entities.[5] However, there is also another option: organizing entities into an overlay and provide lookup via peer-to-peer routing on top of this overlay. This is in fact equivalent to the routing subsystem of a DHT. The only difference is that instead of real nodes, the DHT is made up of virtual nodes corresponding to the given entities. Using only P2P routing to lookup node related entities instead of a complete DHT (P2P routing + storage) has a number of advantages:

[5] Note that the term *locator* and *ID* are used here in a wider sense compared to locators and identifiers in WP-E. By locator, we mean here any object – e.g. a URI, an IP address, etc – containing information necessary to access a specific entity in the network. By ID, we mean here any object identifying an entity in the network independent of its location – e.g. node ID, name of a user, name of a service, etc... So basically, distributed registries adopt the same locator-identifier split concept but this concept is applied to any network entity in general, not only to nodes

Security issues: It is much more difficult to provide security for a storage system than for a routing system. In a distributed storage system, other nodes have to be trusted not only to forward messages in the right direction but also to manage data on behalf of other nodes.

Maintenance overhead: In a dynamic network environment, DHTs have to provide redundant storage placing replicas on multiple nodes in order to prevent data loss when a node fails or leaves the network ungracefully. This increases considerably maintenance overhead. In a routing overlay, entities do not delegate storage of their ID → locator mapping to other nodes but every entity has the responsibility of creating and maintaining its connections in the overlay to be accessible. Consequently, in a routing overlay, there is no maintenance overhead associated with management of replicas.

Composition: When composing two DHTs, the key space has to be repartitioned. As a result, a large number of data items have to be moved to another node creating a burst of maintenance traffic. In case of P2P routing, only the two routing overlays has to be composed which can be implemented at relatively low communication overhead (note that composition of overlays has to be performed for DHTs too- see Appendix A5.1 for details on DHTs (de)composiontion).

P2P routing has only one limitation compared to DHTs: it increases maintenance overhead if the number of node related entities per node is high. In this case, several virtual nodes have to be created at each node. However, this is not the case for most of the mentioned examples. E.g. one node has only one or a few IDs, the number of active users per node is one while the number of resources and service offered by one node is also typically small.

2.7.2 Architecture

Figure 45 shows the high level architecture of the proposed registry. While scalability and robustness is achieved through the use P2P technology, fault tolerance and low maintenance overhead is ensured by bypassing the storage subsystem whenever possible.

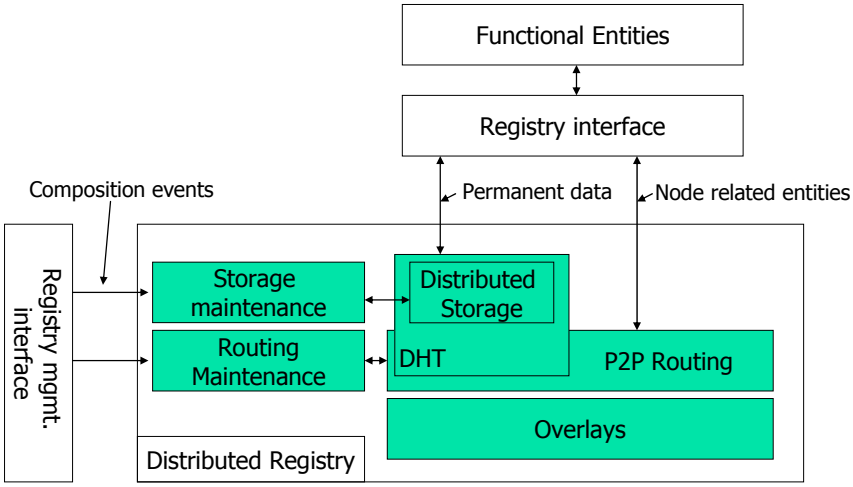


Figure 45 – High level architecture of the registry

The registry interface offers primitives to register and lookup both permanent data and node related entities while the registry management interface is used to notify the registry about composition and decomposition events.

The following subsections present in detail the management of storage and routing subsystems of the registry in a dynamic environment. Both routing and storage maintenance have been designed to minimize maintenance overhead in dynamic network environments without degrading lookup performance.

2.7.3 Maintenance of P2P routing

P2P routing is based on an overlay network that can be described by a directed or undirected graph. Each routing mechanism uses a one or multi-dimensional metric space defining logical distances between the identifiers of nodes and the overlay network graph is embedded into that metric space. A node is found by forwarding lookup requests to the node being the closest to the target.

Efficient and reliable lookup is provided by creating two different connection types: *short-range* (or local) connections and *long-range* connections. Each node has short-range connections to some specific subset of its closest neighbours and has long-range connections to some distant nodes so that the probability of having a long-range connection between to nodes is inversely proportional to the d^{th} power of their distance (where *d* is the dimension of the metric space). In [60], Kleinberg has shown that the above are necessary requirements to provide efficient distributed search based solely on local information. Short-range connections guarantee success of greedy forwarding. If each node is connected to its closest neighbours in the metric space, it is always possible to forward lookup requests at least a small step closer to the target. In contrast, the role of long-range connections is to expedite the lookup process and to provide *O(*log *N)* bounds on the average number of lookup hops. This is achieved by ensuring that the average distance from the target decreases by a constant factor after each lookup step.

The key difference between various P2P routing mechanisms is the degree of determinism in selecting long-range connections (see section A6.5.3). This does not affect routing performance in a static or quasi-static network, but is crucial for maintenance in dynamic environments. In the proposed network model, determinism of the selection of long-range connections is reduced to the minimum by applying a stochastic model. It is shown that exploiting this flexibility in long-range connection selection, maintenance overhead can be significantly reduced using a stochastic maintenance mechanism (see section A6.5.1).

In the proposed network model, long-range connections can be considered as a stochastic process in a one dimensional Euclidian metric space derived from a stationary Poisson process by an exponential transformation (see section A6.3.1). Based on this stochastic transformation model, long-range connection density can be defined as the λ intensity of the generating Poisson process and it can be shown that this λ parameter unequivocally determines distribution of node degree and lookup cost in the system. A loose stochastic maintenance mechanism is proposed ensuring that the long-range connection density λ is within a predefined range and evolution of the system under a given node arrival and departure rate can be described by a Markov chain model.

To evaluate maintenance overhead, the approach presented in [61] by Liben-Nowell et al is used. Maintenance is characterised in terms of the half life of the system, which essentially measures the time for replacement of half of the nodes by new arrivals. It can be shown both analytically and via simulations that communication overhead of long-range connection maintenance per node and per system half life is *O(log n)*, where *n* is the size of the network. Furthermore, it can be pointed out that maintenance overhead cannot be further decreased since this is the theoretical lower bound for maintenance traffic to ensure connectivity of the network[61].

The detailed analytical model and simulation results for the proposed P2P routing maintenance can be found in Annex 5.

2.7.4 Maintenance of distributed storage

Maintenance of the storage subsystem of a DHT is strongly related to key space management. When a new node joins the DHT, it takes over one part of the key space of another node. After departure of a node from the system, the whole key space of that node is taken over by another node. Finally, during network composition and decomposition, the key space of a large number of nodes has to be changed.

Key space repartitioning is an expensive operation. Whenever a node transfers responsibility over one part of its key space partition, it also has to transfer all the data items belonging to that partition. Additionally, in a dynamic network environment, DHTs have to provide redundant storage in order to avoid data loss after node or link failures. Data replication implied by redundant storage further increases the cost of key space repartitioning.

To minimize maintenance overhead related to key space repartitioning, two different algorithms are proposed to compose and decompose DHTs: *gatewaying* and *absorption*. The absorption model refers to two (or more) individual DHTs (that are owned by two or more ANs respectively) completely merging together, resulting in one uniform DHT across the composing ANs. The gatewaying model refers to bridging two (or more) individual DHTs together without modifying their original key space. Both approaches enable information sharing between DHTs, but are tailor-designed to accommodate different network environments.

The gatewaying approach has the advantage of saving a large amount of maintenance traffic avoiding key space repartitioning. One drawback is that normal search operation in the composed DHT is slightly more expensive. In contrast, the absorption model provides normal search in the composed DHT but a careful key space repartitioning process is needed in order to avoid large communication overhead during the composition process.

A detailed description of both DHT composition models can be found in Annex 4.

2.7.5 Continued Work

One important future extension will be a scope feature in the architecture. Based on analysis of requirements from other functional entities, two different registry scopes are needed: global scope and ACS scope registries. This scope parameter should be included into methods of the registry interface. Furthermore implementing global scope registry as a set of cooperating ACS scope registries instead of creating a separate global scope registry is also an interesting research issue.

To lookup an entity in a DHT or P2P routing overlay, the requester has to know the key – which is the hash of the entity. For example it is not possible to lookup directly nodes providing printing service in the ACS, the requester needs to know first the ID of a specific service. Implementing advanced content-based search mechanisms in the registry can be another useful extension to the current architecture.

2.8 ManagementWare FEs Interworkings

This section describes how issues related to distribution of management tasks are currently tackled inside AN. AN relies on distributed management to achieve scalability and advanced autonomic functionalities. In general, all the FEs introduced in section 1.2 collaborate concurrently in a distributed manner: for example, context and triggering information is gathered from different locations in the AN space through the related FEs. Nevertheless this section reports how classical management processes can be supported in the distributed architecture of AN. In particular the following relevant management functions can be used as instruments in the distributed architecture of AN: monitoring, control of self-configuration and distributed registries.

The challenges to support these management functions in an efficient manner come from AN environments, constituted by many domains and frequent domain compositions. As result, such environments exhibit a dynamic networking structure, that requires advanced management instruments. Therefore a set of FEs have been designed to efficiently support management functions in such environment. Figure 46 shows how this set of FEs is built on top of the overlay layer and how they offer services to external FEs (mobility, RRM,

service, etc.); it should be noted that policy and context information can be used as well to control the behaviour of these FEs.

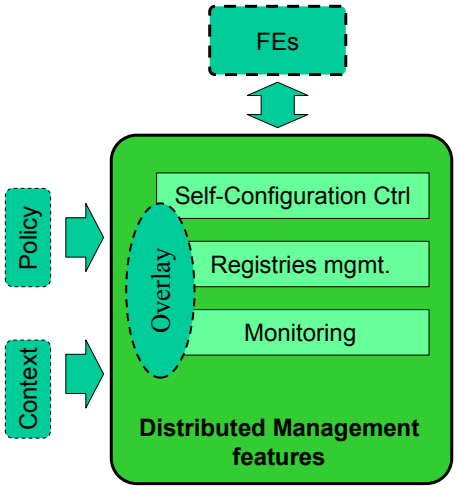


Figure 46 – Distributed Network Management in Ambient Networks

The following management functions have been designed:

- Control of self-configuring processes (e.g. guarantee stability): optimizes decision making of algorithms (e.g. guarantee a certain stability of a configuration over time);
- Registries management: Provides distributed registry functionality and maintenance of registries in dynamic network environments;
- Real-time monitoring of network-wide state: provides estimation of network aggregates (e.g. active flows in the domain).

The design goals for these features are:

- Robustness
- Scalability
- Self-organization
- Stability

Figure 47 shows how the different management functional entities relate to each other and with other components in WP-D.

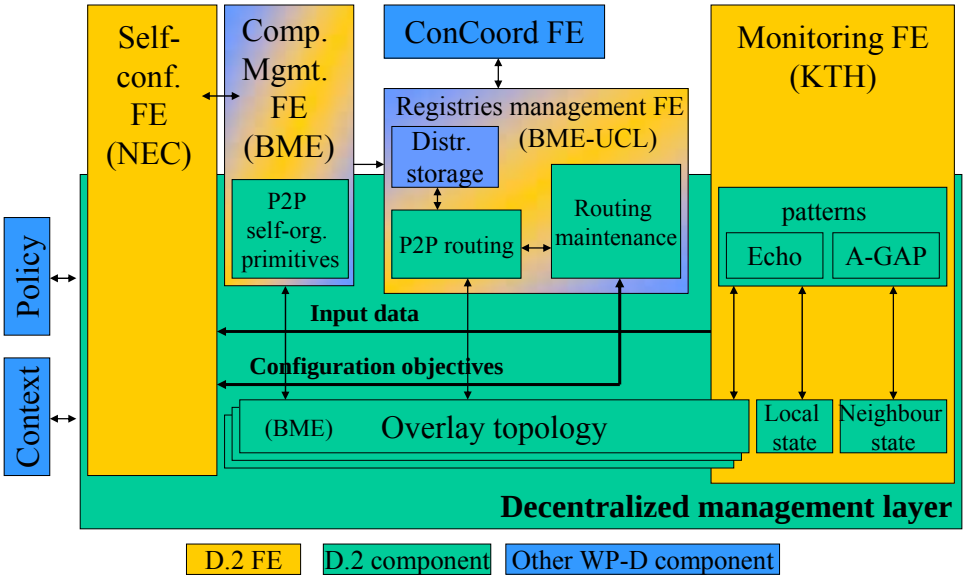


Figure 47 – Relationship between different management functional entities within WP-D

Overlays play a key role in the decentralized management layer. The Monitoring FE is running patterns on top of an arbitrary overlay, P2P routing in the Registries Management FE creates and maintains a special routing overlay and finally, the virtual network structure created and maintained by the Composition Management FE can be described by a hierarchical overlay. The choice of neighbouring nodes in various management overlays can differ significantly, however, independent of neighbour selection criteria, each node in these overlays have to maintain a list of neighbours consisting of ID → Locator pairs. Common management of these neighbour lists for each management overlay have a number of benefits. For example the monitoring FE – which is able to run on top of an arbitrary overlay – can use one of the overlays maintained by the Registries management FE instead of creating and maintaining its own. Another benefit is that the Registries Management FE can increase its robustness and lookup performance taking into account additionally neighbours of the node in other overlays.

Components within the decentralized management layer interact with each other, interact with policy and context management and offer management services to other functional entities. Within the decentralized management layer, the registries management FE uses services of the self-configuration FE to control long-range connection density in the P2P routing overlay (see Annex 5), the Monitoring FE provides input data to the Self-Configuration FE and finally, the Composition Management FE notifies the Registries Management FE whenever the routing and storage parts of the registry in two networks need to be merged as a result of a composition process.

Towards other FEs, the Monitoring FE provides real-time estimations of network aggregates (network-wide variables). The Registries Management FE provides registry functionalities and ensures maintenance of registries in dynamic ambient network environments. The Self-Configuration FE guarantees stability and optimizes decision making of algorithms and finally, the Composition Management FE creates and maintains logical network structure on top of the physical network topology.

3 ManagementWare FEs Interworking in ACS

Context, policy and network management play a very important role in the overall operations of the Ambient Networks and in the services that they provide. As a result, there is much interaction between ManagementWare functions and other functional entities in the ACS. This section presents the interworkings and work in progress between the WP D and other work packages in this project. This section represents the major interactions between the functional entities developed in this work package and those developed in the other work packages. By interacting with other work packages, not only does this make integration of all the functional entities in the ACS simpler and easier, but it also makes possible further development of the functional entities presented earlier. The interworking is between ContextWare and the Triggering functional entity is presented first. Other interworking between the functional entities described earlier and those in other work packages and presented here are with the resource management, bootstrapping, the overlay management and compensation functional entities. Furthermore, the interactions of the context, policy and network management functional entities through the ASI are presented.

3.1 Context Management & Triggering & Transfer for AN (WP-D/B)

As a part of the effort to minimise overlapping activities amongst different workpackages in the project, some degree of overlap has been identified between the WP-B work on Triggering and WP-D work on context management. To resolve such issues a common study item has been progressed jointly amongst these two workpackages, resulting in the detailing of the Ambient Network Information Service, which provides support for management of both information for enhanced mobility management as well as more generic network context information aimed at ensuring longevity and a future-proof validity of the service created. The proposed framework illustrated below details the two major components (see Figure 48). The first one has been extensively described earlier in this document (ContextWare) and is aimed at guaranteeing flexibility and wide applicability of the AN information service. The second component called the Triggering FE, or TRG, is focused on mobility support and is involved with the management and dissemination of triggering events. As illustrated before, the Context Management FE is a collection of distributed Context Managers each carrying out a specialised task.

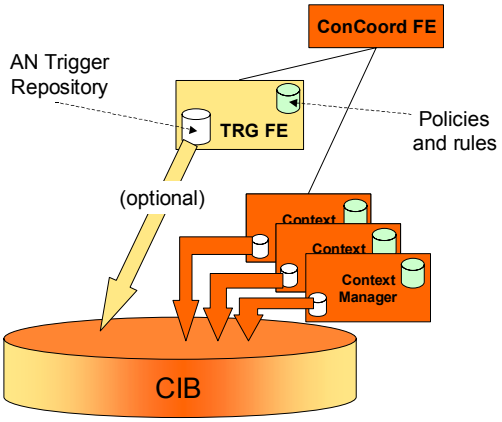


Figure 48 – Diagram showing the integration of ContextWare and the Triggering FE

If we limit the scope of context information to data needed for mobility and handover purposes, the Triggering FE, developed within WP-B of the project, can be seen as a specialised Context Manager dealing mainly with mobility related data and whose role is to notify trigger or triggering event changes to registered consumers.

As it was stated earlier, a Context Manager could implement aggregation, inference and manipulation of low-level context information. This is particularly true in the case of the Triggering FE where clients have the opportunity to specify rules that need to be applied before any notification is sent. So for example, a trigger consumer can register to receive the network load on a particular network, provided that it exceeds e.g. 75% of the overall network capacity. The consumer therefore needs to register as a subscriber for this particular trigger (network load) at this particular condition (i.e. notify *when* exceeds 75%), whereas the role for the triggering FE is to monitor continuously the network load and notify that particular client not on every change, but only for those changes for when the specified filtering rule is satisfied.

The Triggering FE can be seen as a useful example implementation of a particular context manager and currently relies on a centralised approach where the Trigger repository including all triggers and filtering rules is centralised. The same consideration applies for high-level policies and rules, which need to be considered as they affect the decision taken by the consumer decision logic on whether or not to handover based on the trigger notifications. These policies and rules are currently centrally stored in isolation from the policy framework illustrated earlier in this document. Future work will involve the trigger repository being integrated with the CIB and the distributed storage for policies.

3.2 Integrated Management and AN resource management (WP-D/C)

Radio Resource Management (RRM) aims at an efficient usage of radio resources and WP-C is investigating the scenario of multi-radio access in ambient networking environments. The motivation is that today there are many different heterogeneous wireless communication technologies that differ in their data rates, support for mobility, coverage, quality of service, and possible business models. In the future, additional technologies are expected with other characteristics supporting new challenging networking scenarios, but most likely not replacing the existing technologies. Coordinated use of different radio access technologies, so called multi-radio access, can potentially yield significant gains for both providers and end-users of wireless networks.

While many functionalities of RRM are related to the estimation of the channel quality of the radio environment and can therefore be performed as independent elements, some decisions need to be coordinated with other parts of the Ambient Management Layer. Figure 49 shows the interactions inside the Ambient Control Space to perform access selection between different radio access networks. The Multi-Radio Resource Management (MMRM) module manages the access selection between different networks based on different input parameters, like channel quality or flow load, but the decision must finally receive additional information from the management components of the upper layers. In particular the context information and policies can provide additional constraints in the network selection, like accounting information, application requirements, user's preferences etc. In this interaction, policies define the goals to perform decisions, while the MRMM is in charge to enforce them in conformance to the current characteristics of the radio environment.

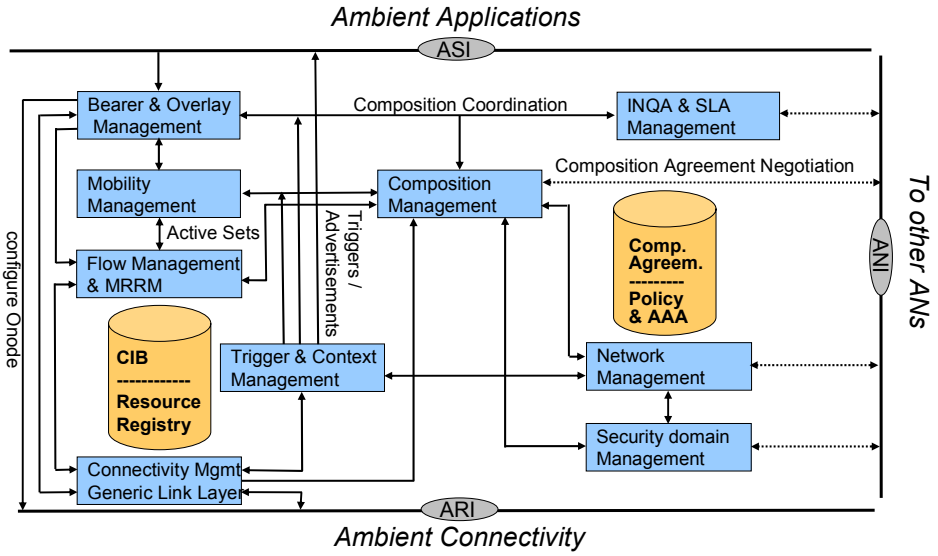


Figure 49 – Interaction diagram for Multi-Radio Resource Management (MRRM)

3.3 Bootstrapping and Reconfigurability (WP-D/E)

Reconfigurability is a natural requirement in mobile ad-hoc environments such as Ambient Networks. It must be supported by the underlying network in a way that the communication of AN nodes must not be interrupted by the effects of mobility and other ad-hoc events.

The locator-identifier split must be introduced within the network level to separate the functionalities of permanent node identifiers and temporary locators. For this the Host-Identity Protocol (HIP) [46] can be used that also incorporates secure communication between nodes. A third user-readable – possibly hierarchical – namespace can also be used for user friendliness that can be translated into a node identifier by some translation mechanism. The use of the locator-identifier approach also eases the reconfiguration of locators as any change in the locator space will not affect the node ID based communication, so locators can be changed any time if needed.

3.4 Integrated Management and Overlay Node architecture (WP-D/F)

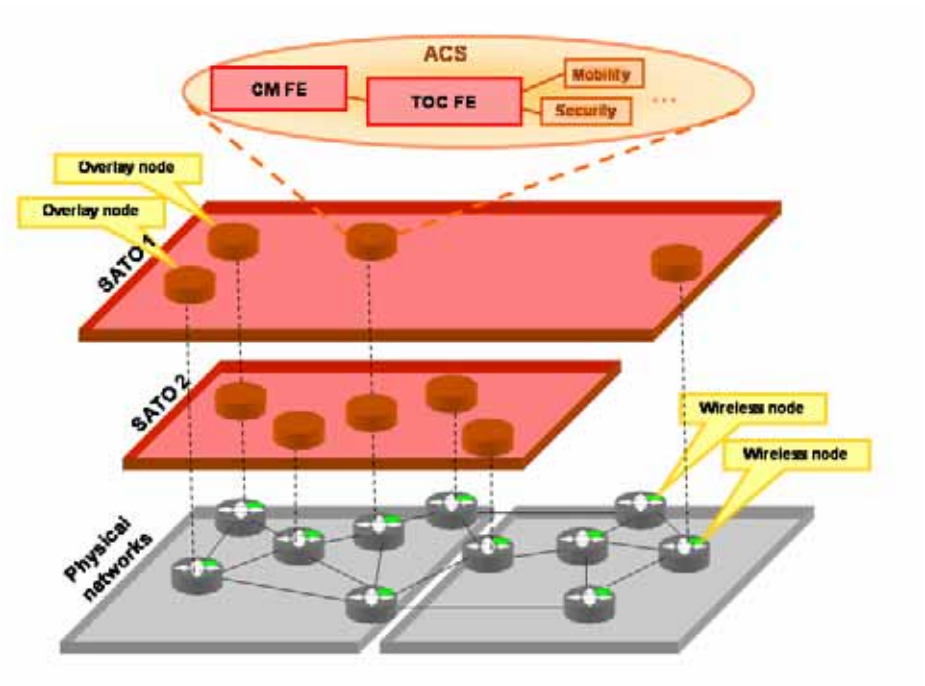


Figure 50 – An illustration of the ACS with example establishments of SATOs

The Transport Overlay Control (TOC) Functional Entity (FE) is responsible for dynamically establishing and maintaining service-specific overlays in ANs, known as Service-aware Adaptive Transport Overlays (SATOs). Example SATO establishments are shown in Figure 50. SATOs are service-specific overlays established on top of physical wireless networks that consist of wireless nodes; as such, a wireless node could be a member of multiple SATOs (i.e. multiple overlay nodes on one wireless node). Note further that as the name suggests, a SATO provides a managed transport channel for a service. By managed transport channel, we mean through the establishment of these service-specific overlays, specific resources (and other supportive services) in the network that are needed to support a (new) service, could be reserved, configured, and shared within and across ANs to support the (new) service. For example, a QoS-SATO is capable of providing a quality-assured video transmission service for AN end-users, by discovering, reserving, and controlling transcoding services and QoS control services that are available in the AN. For more details on the concepts of SATO, readers are referred to [43].

The key to SATO is the SATO Lifecycle Management (LM). The SATO LM implements the TOC FE, and it is capable of dynamically deploying (new) Context Sensors in ANs for monitoring and collecting service-specific network context that are of interest to the SLM for service management. The SATO SLM inter-works with the ConCoord to locate collected, distributed network context in a scalable and distributed manner. For example, to manage a quality-assured video transmission service in an AN, the SATO LM deploys a specific type of Context Sensor to monitor where transcoding services (i.e. a service context) are available in the AN (so that the video can be transcoded when required). The results from the Context Sensor would be the network (or overlay) location(s) of where the transcoding services is(are) available. The results are stored by the ConCoord in the AN. The SATO LM then inter-works with the ConCoord to obtain the location of the available transcoding service(s) and subsequently provides and manages a quality-assured video transmission service in the AN.

3.4.1 Network Context-awareness in SATOs

In our previous research [43], we have already identified that the collection, management, and use of network context information will enhance service management and service provision in ANs. This is due to the dynamic nature of AN i.e. individual AN nodes (i.e. where network context resides on) may dynamically and constantly join and leave ANs. Thus, network context may become (un)available when nodes join(leave). Real-time network context information must be made readily available to SATOs, in order to enable SATOs to dynamically self-adapt itself accordingly (and hence managing end-users' services). For example, if a new node that hosts a transcoding service joins an AN of which previously none of its nodes hosted transcoding services, this piece of new network context i.e. "transcoding service now available on node x in AN y" must be made available to the QoS-SATO, so that a quality-assured video transmission service can be provided.

This implies that there is a need for the provision of network context-awareness in SATOs. By network context-awareness, we refer to the capability of a SATO to use network context information for self-adaptation, or in the provision of services. The provision of network context-awareness is needed in SATO in order to aid in distributed service management in heterogeneous wireless environments, in response to the ever-changing network context. Examples of network context that are useful for service management in AN are service context, node context, QoS context, flow context… etc. As a summary, the requirements of SATO service management are:

- to support efficient deployment, activation, and (re)configuration of (new) Context Sensors in a scalable and distributed manner on (potentially) large numbers of AN nodes; in order to monitor and collect specific pieces of network context that are needed to support service management in SATO LM, and;
- to provide a scalable, distributed mechanism for locating (the collected) distributed network context within an AN; in order to support subsequent network context retrieval by AN service managers (i.e. SATO LM) for service management.

3.4.2 Integration between SATO LM and ConCoord

As discussed in previous section, the ConCoord is the implementation of the CM FE. The ConCoord handles the indexing, registration, authorisation, and resolution of the locartion of the deployed Context Sensors and context object identifiers. Context object identifiers are location references of network context that are collected by Context Sensors. In brief, Context Sensors deployed and activated in an AN register themselves and the type of network context that they are collecting with the distributed ConCoord through the ContextWare Interface (CWI) (Figure 51). The ConCoord then provides the facility to store location references to the source of the collected network context in a scalable and distributed manner, to ease the subsequent retrieval process of the SATO LM through a resolution process (Figure 51).

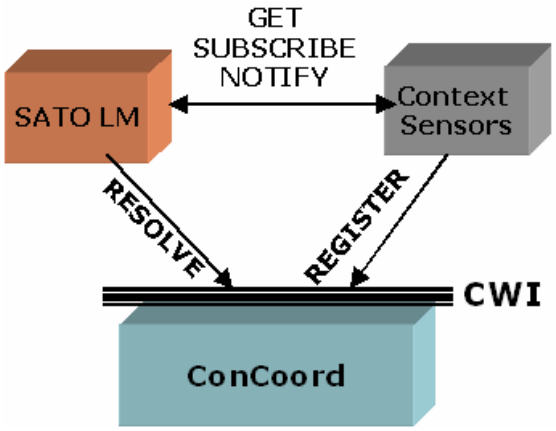


Figure 51 – The inter-working between SATO LM and ConCoord

The basic ConCoord functionalities are implemented by mapping the defined ConCoord primitives REGISTER, RESOLVE, SUBSCRIBE and NOTIFY to the counterpart operational primitives of the DHT. These primitives enable context clients (i.e. the SATO LM) to request for notifications of specific types of network context event. For example, suppose some node context sensors have already been deployed in an AN, and the sensors are registered with the ConCoord. If the SATO LM needs network context that are being monitored by the (already deployed) sensors, it will subscribe to the ConCoord. The ConCoord will then notify the SATO LM should the network context of interest become available. A context client will only receive notifications regarding the network context that it has subscribed to.

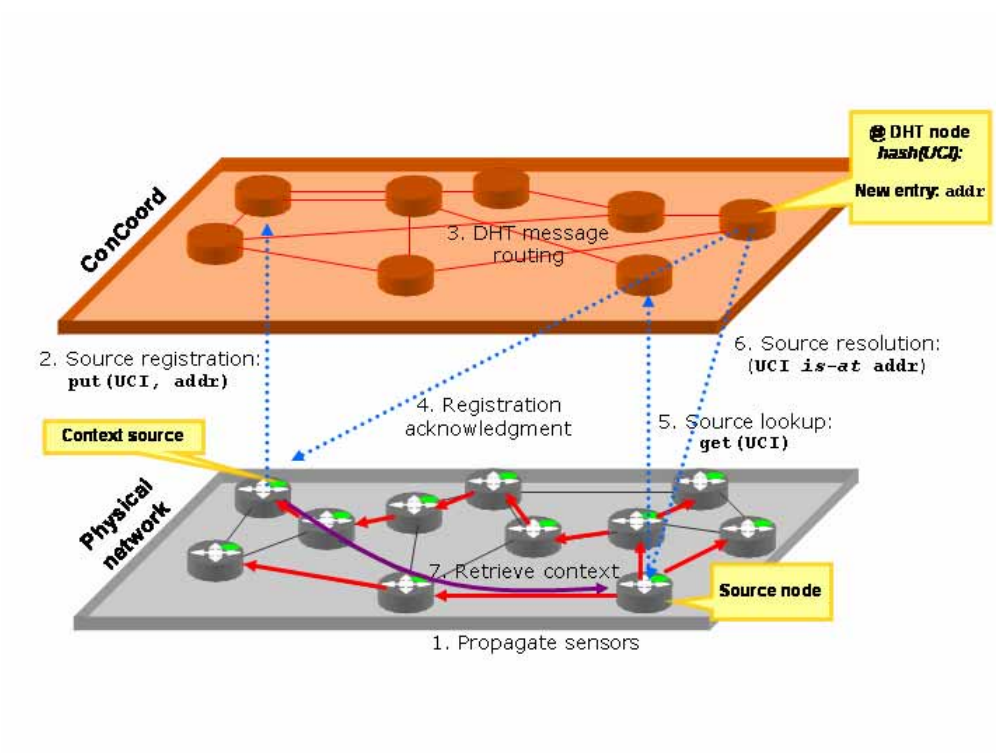


Figure 52 – An illustration of the inter-working between SATO LM and ConCoord

Figure 52 shows an illustration of the inter-working between SATO LM and ConCoord to achieve dynamic deployment of (new) Context Sensors and retrieval of service-related network context for service management in AN. For example, suppose a new type of Context Sensor has been created to monitor a piece of network context that is useful for

managing a particular service in the SATO LM on the source node. The to-be-monitored service-related network context is identified using a Uniform Context Identifier (UCI). Then, the SATO LM deploys the Context Sensor from the source node using the echo pattern (step 1 in Figure 52). Once deployed and activated, the Context Sensors register themselves and the UCI with the ConCoord by exercising a put(UCI, sensor-address) operation (step 2 in Figure 52), and the value pair is stored in the ConCoord through DHT message routing. The information pair is then stored in the ConCoord through DHT message routing (step 3 in Figure 52). This operation may either store the location references of the network context or the actual piece of network context itself. Once the information has been stored in appropriate location in the ConCoord, the node on which the Context Sensor is residing on will receive an acknowledgement (step 4 in Figure 52). The context client, in this case the SATO LM (i.e. the source node), may exercise a RESOLVE operation that is mapped to a get(UCI) in the ConCoord i.e. a “pull” model for context delivery (step 5 in Figure 52), which returns the sensor-address parameter to the node issuing the request (step 6 in Figure 52). SATO LM can then obtain the network context information directly from the sensor using the returned address (step 7 in Figure 52).

3.4.3 Summary

Due to the dynamic nature of ANs, there is a need to investigate into a scalable and distributed approach for collecting and storing distributed network context information, to support service management in SATOs. Service-specific network context needed for managing specific services in ANs must be monitored and collected in a scalable and distributed manner. Collected network context information must also be stored in a scalable and distributed manner to ensure readily accessibility to AN service mangers. This section presents an inter-working approach between the SATO LM and ConCoord to the problem space. SATO LM uses a scalable and distributed algorithm i.e. the echo pattern for rapid deployment, activation, and (re)configuration of (new) Context Sensors. Deployed Context Sensors are registered with the distributed ConCoord, which is implemented using DHT techniques, to create a logical distributed database that keeps location references to the collected network context information from the Context Sensors. As such, the SATO LM inter-works with the ConCoord to provide a fully distributed approach for collecting and storing network context in order to support service management in SATO.

3.5 ASI Management primitives (WP-D/F)

The Ambient Service Interface (ASI) comprises the collection of Service Interfaces (SI) exported by the Functional Entities (FE) that constitutes the ACS. It interfaces between the ACS and the services external to the Ambient Network that wishes to make use of it and hides the individual FEs from the external services. As a result, the structuring of an ACS into FEs is therefore only an internal issue and not visible to these services. The ASI is accessible to any service clients (e.g. end-user applications, management applications, control applications, etc.) outside the ACS with appropriate access permissions. When a service uses the ASI to access the functional entities in the ACS, it becomes an ASI client. Figure 53 below shows the bindings between the Functional Elements and the ASI primitives.

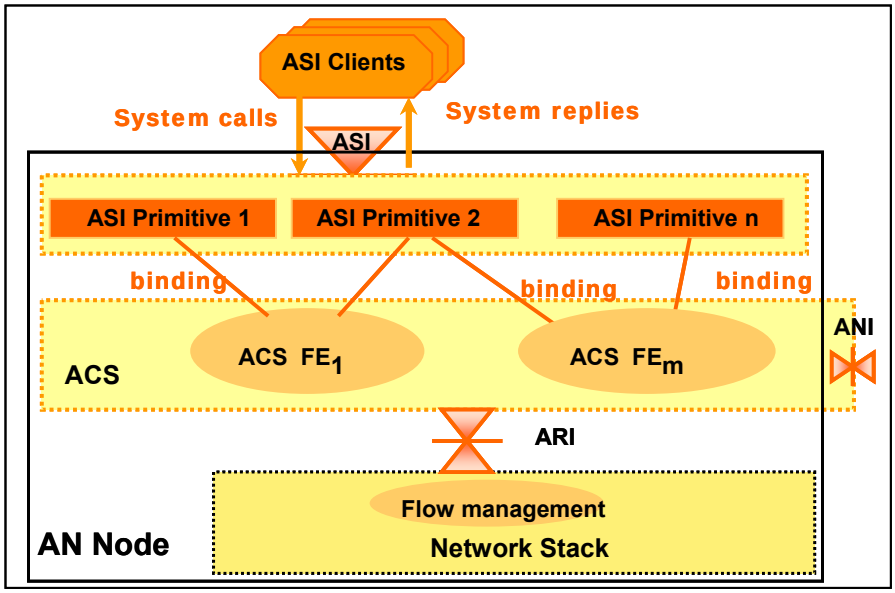


Figure 53 – Diagram depicting the bindings between the Functional Entities and the ASI primitives

ASI primitives describe a minimal communication sessions executed to trigger a specific action, usually consisting of a request and a response. This action involves the ASI clients using the services provided by the functional entities in the ACS. As a result, the ASI primitives can be grouped according to specific FE that it is trying to use. ASI primitives can be implemented using a request-response message-based protocol or function calls. This section provides a specification for the ASI primitives for a subset of FEs (i.e. for WP-D FEs), which includes: ContextWare ASI Primitives (WP-D1), Network Management ASI Primitives (WP-D-2) and Policy Management ASI Primitives (WP-D3). The ASI primitives are presented in an interface template provided in Deliverable D1.5 [58] from AN Phase One. Each table presents a subset of the ASI interface, which contains some of the ASI primitives. This specification was provided to the ASI task force and the relevant milestone report WP-F Milestone F-2 “ASI Design & Baseline Analysis”.

3.5.1 ContextWare ASI Primitives (WP-D1)

This section presents the ASI primitives for the ContextWare functional entities. These ASI primitives enable the transfer of context as well as context object registrations across the ASI. Three interfaces are presented for the: the Source-ConCoord interface, the Client-ConCoord interface, and Client-Source interface. The Source-ConCoord interface details the interactions that take place when context sources register their context objects, identified by UCIs (universal context identifiers) at the ConCoord. The Client-ConCoord interface presents the primitives used when context clients want to locate context sources while the Client-Source interface contains primitives for the direct interaction between context sources and clients.

3.5.2 Source ConCoord Interface

The Source-ConCoord interface is used by context sources to register their context objects at the ConCoord. Each context object is identified by its unique UCI. The UCI location pair is stored in the ConCoord so that any context client can query the ConCoord to obtain the location of this UCI. This interface consists of one primitive, REGISTER.

Interface Name	Source-ConCoord-Interface
Interface Type	Operation

List of Interactions			
Operation: REGISTER.request (Interrogation) #1		Parameter Name	Parameter Type
A context source registers itself with the ConCoord. The context data object(s) provided by the source are uniquely identified by their UCIs. The source is authenticated based on its credentials. The source provides access policies to restrict access in case of sensitive context information. The location denotes a URL where the context data object(s) can be accessed (normally the URL of the registering source itself).		one or more UCIs (universal context IDs) (m)	String
		Credentials (m)	TBD
		Access Policies	TBD
		Location	TBD
Termination: REGISTER.response #1.1		Parameter Name	Parameter Type
The ConCoord responds to the registration operation of a context source. The response code indicates the result of the registration request (success, auth-failed, etc)	Success, auth-failed, etc	Response Code (m)	Enumeration

Table 1: Primitives for the Source ConCoord Interface

3.5.3 Client ConCoord Interface

The Client-ConCoord is used by context clients to locate context objects in the network. Clients send a RESOLVE.request with one or more UCIs to the ConCoord, which answers with a RESOLVE.response containing the locations of the context objects. This interface consists of one primitive, RESOLVE.

Interface Name		Client-ConCoord-Interface	
Interface Type		Operation	
List of Interactions			
Operation: RESOLVE.request (Interrogation) #1		Parameter Name	Parameter Type
A context client requests the ConCoord to resolve one or more UCIs. The client is authenticated based on its credentials. The ConCoord may also apply further access checks based on the access policies, which have been provided by the sources during registration.		one or more UCIs (universal context IDs) (m)	String
		Credentials (m)	TBD
Termination: RESOLVE.response #1.1		Parameter Name	Parameter Type
The ConCoord responds to the resolve operation of a context client. The response code indicates the result of the	(success, auth-failed, access-denied, etc)	Response Code (m)	Enumeration

registration request (success, auth-failed, access-denied, etc). The locations denote the URLs where the context data objects(s) identified by the UCIs in the request can be accessed.		Location (m)	String

Table 2: Primitives for the Client ConCoord Interface

3.5.4 Client Source Interface

The client-source interface is used by clients to retrieve the values of context objects from sources. There are two modes of context retrieval. With the GET primitive, a client can directly obtain the value of a context object from the source. In order to do this the context objects UCI must have been resolved to obtain the location of the context object. With SUBSCRIBE/NOTIFY, a client can subscribe for changes in context objects and will subsequently receive NOTIFY primitives for specified change events. Clients should *never poll* a context object, but subscribe to it instead.

Interface Name		Client-Source-Interface	
Interface Type		Operation	
List of Interactions			
Operation: GET.request (Interrogation) #1		Parameter Name	Parameter Type
A context client requests context information from a context source. The source authenticates the client based on its credentials. The source may also apply further access control.		one or more UCIs (universal context IDs) (m)	String
		Credentials (m)	TBD
Termination: GET.response #1.1		Parameter Name	Parameter Type
A context source responds to the GET request of a context client. If the response code indicates success, the response contains the context information identified by the UCIs in the corresponding request. The access policies are optional and should included if the client stores the contained context information on behalf on the source.	success	Response Code (m)	Enumeration
		Context Info	Data
		Access Policies	TBD
		Location (m)	String
Operation: SUBSCRIBE.request (Interrogation) #2		Parameter Name	Parameter Type
A context client subscribes to context information identified by the UCI at a context source. The source authenticates the client based on its credentials. The source may also apply further		one or more UCIs (universal context IDs) (m)	String
		Credentials (m)	TBD

access control. The event-condition specifies when the client is to be notified. The special event-condition constant “cancel” means the subscription is to be cancelled and no further notifications are to be sent.		Event Condition	EXPRESSION | cancel
Termination: SUBSCRIBE.response #2.1		Parameter Name	Parameter Type
A context source responds to the SUBSCRIBE request of a context client. If the response code indicates success, the client will receive notifications (as described next) whenever the event-condition holds at the context source.	success	Response Code (m)	Enumeration
Operation: NOTIFY (Announcement) #3		Parameter Name	Parameter Type
A context source notifies a context client having an active subscription pending that its event-condition has become true. The NOTIFY announcement contains the current context information for the subscribed UCI.		one or more UCIs (universal context IDs) (m)	String
		Context Info (m)	DATA

Table 3: Primitives for the Client Source Interface

3.5.5 Network Management ASI Primitives (WP-D2)

The ASI is envisaged as a set of interfaces containing ASI primitives. One such set of primitives are the networks management primitives, which allows applications and services to issue ASI requests to the Ambient Control Space concerning the establishment, maintenance, and termination of end-to-end connection between functional instances connecting to the ASI.

The network management primitives are a small set of general management primitives allowing bi-directional communication between services and ACS components for management functions. The network management interface of the ASI allows services to query interfaces of ACS components and use elements of those interfaces to interact with the ACS components.

3.5.6 ACS Component Registration Interface

The ASI management primitives enable the external services to access a registry of ACS components and interfaces. Each ACS component that needs to be accessible from outside the ACS registers three things with the ASI:

- an **ACS component identifier** identifying the ACS component,
- an **ACS component interface** describing the publicly available methods of the ACS component,
- the **ACS component instance** implementing the interface.
- The **RegisterACSComponent** interface enables the registration and deregistration of the ACS components and interfaces and is presented below.

Interface Name	RegisterACSComponent
Interface Type	(Operation/Stream)

List of Interactions			
Operation: #1RegisterACSComponentIdentifierRequest		Parameter Name	Parameter Type
To register the ACSComponent		ACSComponentID	String
		Register	boolean
Operation: #2RegisterACSComponentInterfaceRequest		Parameter Name	Parameter Type
To register the ACSComponentInterface		ACSComponentInterfaceName	tbd
		Register	boolean
Operation: #3RegisterACSComponentInstanceRequest		Parameter Name	Parameter Type
To register an instance of the ACSComponentInterface		ACSComponentInstance	tbd
		Register	boolean
Termination: #1.1 RegisterACSComponentIdentifierResponse		Parameter Name	Parameter Type
		ACSComponentID	String
		RegisterSuccess	Boolean
Termination: #2.1 RegisterACSComponentInterfaceResponse		Parameter Name	Parameter Type
		ACSComponentInterfaceName	tbd
		RegisterSuccess	Boolean
Termination: #3.1 RegisterACSComponentInstanceResponse		Parameter Name	Parameter Type
		ACSComponentInstance	tbd
		RegisterSuccess	Boolean

Table 4: Primitives for the registration and deregistration of ACS components and interfaces

3.5.7 ASI Connection Interface

When the ACS components and interfaces are registered then the external services can create connections through the ASI to access the instances of these interfaces. Three types of connections have been defined, the **`ConnectorRequest`**, **`ConnectorResponse`** and **`ConnectorNotification`** and are presented below.

Interface Name		ACSConnection	
Interface Type		(Operation/Stream)	
List of Interactions			
Operation: #1 ConnectorRequest		Parameter Name	Parameter Type
Request sent from the external services to the ACS		ACSComponentIdentifier	tbd
		Method Signature	tbd
		Method Parameters	tbd
Operation: #2 ConnectorNotification		Parameter Name	Parameter Type
Notifications sent from the ACS to the external services		ACSComponentIdentifier	tbd
		Message	tbd
Termination: #1.1 ConnectorResponse		Parameter Name	Parameter Type
		Method Return Values	any
		Method Exceptions	Object

Table 5: Primitives to provide the external services access to the ACS

3.5.8 Policy Management ASI Primitives (WP-D3)

The Policy Management FE enables network, service, security, composition and context management through the use of a Policy-Based Management Framework. There is a need for the external services to specify to the ACS certain policies which can define the interaction between these services and the ACS. These external service polices are stored in the Policy Repository. Detailed below are the primitives contained in the **PolicyManagement** interface.

Interface Name	PolicyManagement	
Interface Type	(Operation/Stream)	
List of Interactions		
Operation: #1 SendPolicyRequest	Parameter Name	Parameter Type
Policy sent from the external services to the ACS	ServiceID	tbd
	PolicyID	tbd
	PolicySetID	tbd
	Policy Rules????	tbd
Operation: #2 RemovePolicyRequest	Parameter Name	Parameter Type
Removal of existing policy	ServiceID	tbd
	PolicyID	tbd
	PolicySetID	tbd

Operation: #3 ActivatePolicyRequest		Parameter Name	Parameter Type
Activation/ de-activation of service policies		ActivatePolicy	boolean
		PolicyID	tbd
		PolicySetID	tbd
Termination: #1.1 SendPolicyResponse		Parameter Name	Parameter Type
		PolicySetID	tbd
		PolicyID	tbd
		PolicyAccepted	boolean
Termination: #1.1 RemovePolicyResponse		Parameter Name	Parameter Type
		PolicySetID	tbd
		PolicyID	tbd
		PolicyRemoved	boolean
Termination: #1.1 ActivatePolicyResponse		Parameter Name	Parameter Type
		PolicySetID	Tbd
		PolicyID	Tbd
		PolicyActivated	Boolean

Table 6: Primitives for the Policy Management Interface

3.6 Network Composition Management – Interactions with composition FE (WP-D/G)

The interactions between Management of Network Composition and Composition Management FE are presented in section 2.5.2.

3.7 Management of Compensation (WP-D/G)

Compensation information should be stored in the context management framework. The negotiation, and therefore the stored information must include the compensation aspects of the AN service being negotiated among ANs (accounting, pricing, reporting, payment method, etc.) but also aspects about the configuration of the CeFE (compensation FE) after the agreement (i.e. which AN will take which responsibilities regarding compensation functions, are Helpers needed, etc.).

The decision on which AN to connect to or which network connection or FE to use depends also on the policies for compensation, for example “always choose the cheapest network connection/FE”, but also compensation rules should be taken into account in the nodes when providing a service to third party entities. Therefore the policy framework is also affected; policies for composition should be stored in the policy repository and the policy enforcement points should take compensation rules into account, as well.

The compensation FE needs context change events, in case the topology or neighbourhood changes, e.g., when new ANs appear or old ANs disappear. Topology or neighbourhood change would trigger a composition/compensation decision, which will be performed according to the applied policies.

4 Management Approaches

We are assuming that there is certain cooperation possible between AN underlay and Service overlay layers. Information and /or policies at the overlay could be provided at underlay and vice versa in order to better solve an overlay/underlay problem, including cross-layers optimisation. Cross-layer design enhances the traditional design, where each layer of the protocol stack operates independently by facilitating information exchange between layers and by optimising end-to-end performance of different problems.

A number of cross-layer management approaches are presented in this chapter, including Context management in AN underlay / overlay, Self-organisation in AN underlay/overlay, Policies for AN underlay/overlay. In addition management mechanisms for Security in ContextWare and for Distributed Management are also presented in this chapter.

4.1 Context-aware Service Management

The Overlay Management (OM) Functional Entity (FE) is responsible for dynamically establishing and maintaining service-specific overlays in ANs i.e. SATOs. Readers should note that SATOs are service-specific overlays established on top of physical wireless networks that consist of wireless nodes; as such, a wireless node could be a member of multiple SATOs (i.e. multiple overlay nodes on one wireless node). Note further that as the name suggests, a SATO provides a managed transport channel for a service. By managed transport channel, we mean through the establishment of these service-specific overlays, specific resources (and other supportive services) in the network that are needed to support a (new) service, could be reserved, configured, and shared within and across ANs to support the (new) service. For example, a QoS-SATO is capable of providing a quality-assured video transmission service for AN end-users, by discovering, reserving, and controlling transcoding services and QoS control services that are available in the AN.

The key component that handles SATO management is the SATO manager. The SATO manager implements (some of the functionalities of) the OM FE, and it is capable of dynamically deploying (new) Context Sensors in ANs for monitoring and collecting service-specific network context that are needed for service management through SATOs. The SATO manager inter-works with the Context Co-ordinator (ConCoord) to locate collected, distributed network context in a scalable and distributed manner. For example, to manage a quality-assured video transmission service in an AN, the SATO manager deploys a specific type of Context Sensor to monitor where transcoding services (i.e. a service context) are available in the AN (so that the video can be transcoded when required). The results from the Context Sensor would be the network (or overlay) location(s) of where the transcoding services is(are) available. The results are stored by the ConCoord in the AN. The SATO manager then inter-works with the ConCoord to obtain the location of the available transcoding service(s) and subsequently provides and manages a quality-assured video transmission service in the AN.

4.1.1 Network Context-awareness in SATOs

We have already identified that the collection, management, and use of network context information will enhance service management and service provision in ANs. This is due to the dynamic nature of AN i.e. individual AN nodes (i.e. where network context resides on) may dynamically and constantly join and leave ANs. Thus, network context may become (un)available when nodes join(leave). Real-time network context information must be made readily available to SATOs, in order to enable SATOs to dynamically self-adapt itself accordingly (and hence managing end-users' services). For example, if a new node that hosts a transcoding service joins an AN of which previously none of its nodes hosted transcoding services, this piece of new network context i.e. "transcoding service now available on node x in AN y" must be made available to the QoS-SATO, so that a quality-assured video transmission service can be provided.

This implies that there is a need for provisioning of network context-awareness in SATOs. By network context-awareness, we refer to the capability of a SATO to use network context information for self-adaptation, or in the provision of services. The provision of network context-awareness is needed in SATO in order to aid in distributed service management in heterogeneous wireless environments, in response to the ever-changing network context. Examples of network context that are useful for service management in AN are service context, node context, QoS context, flow context, etc. As a summary, the requirements of SATO service management are:

- to support efficient deployment, activation, and (re)configuration of (new) Context Sensors in a scalable and distributed manner on (potentially) large numbers of AN nodes; in order to monitor and collect specific pieces of network context that are needed to support service management in SATO LM, and;
- to provide a scalable, distributed mechanism for locating (the collected) distributed network context within an AN; in order to support subsequent network context retrieval by AN service managers (i.e. SATO LM) for service management.

4.1.1.1 Dynamic Service Context Sensor Deployment

In this section, we introduce the echo-pattern solution that is designed for the distribution, installation, activation, and (re)configuration of (new) context sensors. SATO managers are distributed in the sense that SATO manager is part of the distributed OM FE; and each SATO manager (hence each SATO node) is capable of deploying (new) service context sensors. There are three identified scenarios for service context sensors deployment in SATO:

a) *distributing (new) sensors*: a large number of nodes have just joined the network, and they notify other nodes about the list of their installed sensors. Suppose a type of sensor has already been deployed in the network (that are currently being used to support a SATO). The SATO managers must deploy the sensor on the new joining nodes. Another occasion would be when a node joins and requests for a (new) service that the corresponding sensors have not been launched before in the network, or when a SP is about to launch a new service that requires new type of service context to be monitored. As such, (new) sensors would have to be launched;

b) *(de)activating deployed sensors*: a type of service context sensor has already been deployed in the network, but there is a need to (de)activate the sensor(s). For example, a node that requests for quality-assured video transmission service has left the network, thus the flow context sensors are no longer needed to be functional (assuming the video service was the only service that needs information from the flow context sensors);

c) *(re)configuring deployed sensors*: a (type of) service context sensor has already been deployed in the network, but there is a need to (re)configure the sensor(s). For instance, the size of the network has reduced (say a large number of nodes have left the network), and now there is a need to monitor the top five flows in the network only, instead of the top ten. Thus, existing flow context sensors should be re-configured to monitor and collect information on the top five flows only.

When deploying a (new type of) service context sensor, an explorer packet is propagated from the source node[6] through the network. The explorer packet contains the implementation and installation code of the new sensor. The figure 6.6.-1 shows a list of items carried in an explorer packet:

Sensor_ID, [Sensor_code], [Exe_code]

Figure 54 – A list of items carried in an explorer packet

Items in square brackets are optional. `Sensor_ID` is an identifier that identifies the type of the sensor carried in the explorer packet. This is a cryptographically created random number based on some random inputs and the source node's preferred identifier (e.g. the IP address/overlay ID of node A in Figure 54) that is used for other nodes to refer to the sensor. `[Sensor_code]` is the field that keeps the actual sensor implementation and installation classes. This field is optional e.g. included for case (a), but may also be included for case (b) and (c) (see later). `[Exe_code]` is also optional; it is either the (de)activation code for case (b), or the (re)configuration code for case (c). Activation code should be included for case (a) if the new sensor was to be activated as soon as it was installed.

For case (a), the SATO manager on the source node keeps the `Sensor_ID` of the (new) sensor to be propagated. The explorer packet is then propagated across the entire network using the distributed echo pattern: the SATO manager on the source node (e.g. node A) sends the explorer packet to all of its immediate neighbours (e.g. node B and C), which process the explorer packet respectively (see shortly later), and then redistribute the explorer packet to their immediate neighbours, and so on. At each node (other than the source node), upon intercepting the explorer packet, the node checks the `Sensor_ID` of the intercepted explorer packet against the list of its locally installed service context sensors. If a new type of sensor is distributed in the intercepted explorer packet, or if the node has not previously installed the sensor (i.e. a new joining node), then no match should be found (i.e. the sensor is new and therefore couldn't possible have been installed on the node). An evaluation is then performed on the intercepting node, prior to installing the sensor. The evaluation process includes verifying the compatibility and availability of the required software/service modules to install the sensor. For example, to install a sensor that requires `tc`, then `tc` should be available on the node; else, the sensor should not be installed or the node should try to install `tc` prior to installing the sensor. The sensor should be installed only if the evaluation is satisfactory. Installation of sensors is carried out through executing executable installation classes (carried in `[Sensor_code]`) through programmable approaches [76]. Executable installation classes are classes that contain the code for installing sensors. These code are provided by the Service Providers who implement the sensors (discussion on implementation and installation of sensors is out of scope of this chapter). Once the installation has completed, the node updates its list of locally installed sensors (to include the recently installed sensor), and executes the code in `[Exe_code]` if any code is included in that field.

[6] A source node would be the node on which the (new) service context sensor is about to be distributed to the AN.

For case (b), the `Sensor_ID` (of the sensor to be activated) is embedded in the explorer packet together with the sensor's activation code (kept in `[Exe_code]`). The sensor's implementation and installation classes may be included in `[Sensor_code]` as well (optional, see later). The explorer packet is then distributed in the same fashion as described above. SATO managers on intercepting nodes check whether the corresponding sensor has already been installed (by checking against its list of already installed sensors):

if the sensor has already been installed, the activation code in `[Exe_code]` is executed;

else if the sensor has not been installed (i.e. the intercepting node is a new joining node), and if the sensor implementation and installation code are kept in `[Sensor_code]`, the sensor will be installed and activated.

The same applies to case (c), except that `[Exe_code]` contains the (re)configuration code that is to be executed. The choice on whether to store sensor implementation and installation classes in the explorer packet for case (b) and case (c) depends on the type of sensor involved. For example, this option may not be needed for some common types of sensors (such as the Node Sensors that are installed in all SATO nodes). But for less common types of sensors, or a new type of sensors, the classes may be included. Including sensor classes in this optional field enables new joining node to install sensors at the time of explorer packet interception, without making further queries.

Once sensors have been distributed, installed, activated and (re)configured, they will start monitoring and collecting service context. As explained in earlier section, there is a need to have a scalable and distributed technique for locating the collected service context (collected by the distributed service context sensors), in order to enable SATO managers to retrieve the collected service context, which are subsequently used for managing end user services. A distributed service context registry, known as the Service ConCoord, provides this functionality.

4.1.1.2 An Overview on the ConCoord

The ConCoord is the implementation of the Context Management (CM) FE in the underlay. The ConCoord handles the indexing, registration, authorisation, and resolution of the location(s) of the deployed Context Sensors and context object identifiers. Context object identifiers are known as Universal Context Identifiers (UCIs), which are used for identifying different types of network context. In brief, Context Sensors deployed and activated in an AN register themselves and the type of network context that they are collecting with the distributed ConCoord through the ContextWare Interface (CWI). The ConCoord then provides the facility to store location references to the source of the collected network context in a scalable and distributed manner, to ease the subsequent retrieval process of the SATO LM through a resolution process (Figure 55).

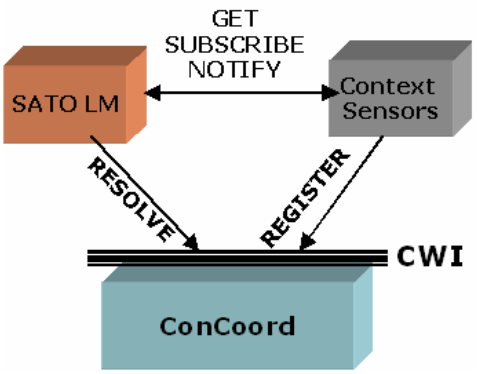


Figure 55 – The inter-working between SATO LM and ConCoord

The basic ConCoord functionalities are implemented by mapping the defined ConCoord primitives REGISTER, RESOLVE, SUBSCRIBE and NOTIFY to the counterpart

operational primitives of the DHT. These primitives enable context clients (i.e. the SATO LM) to request for notifications of specific types of network context event. For example, suppose some node context sensors have already been deployed in an AN, and the sensors are registered with the ConCoord. If the SATO LM needs network context that are being monitored by the (already deployed) sensors, it will subscribe to the ConCoord. The ConCoord will then notify the SATO LM should the network context of interest become available. A context client will only receive notifications regarding the network context that it has subscribed to.

4.1.1.3 Integration between SATO LM and ConCoord

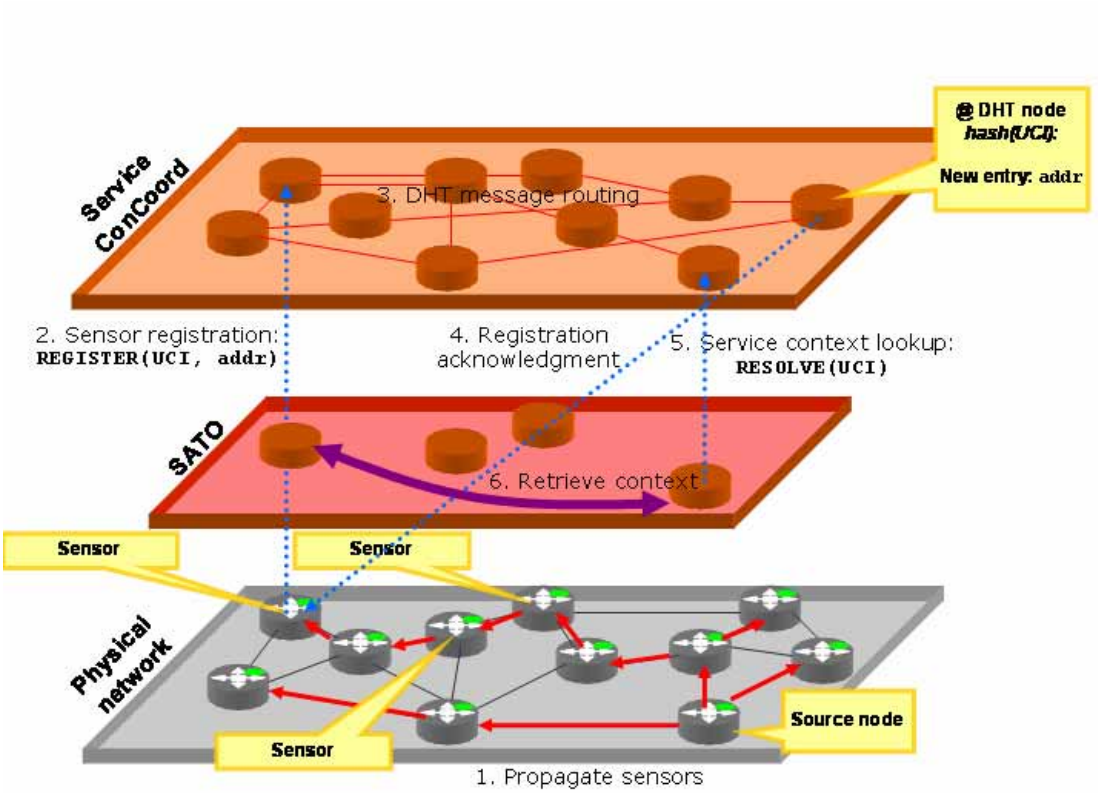


Figure 56 – An illustration of the inter-working between SATO LM and ConCoord

Figure 56 shows an illustration of the inter-working between SATO LM and ConCoord to achieve dynamic deployment of (new) Context Sensors and retrieval of service-related network context for service management in AN. For example, suppose a new type of Context Sensor has been created to monitor a piece of network context that is useful for managing a particular service in the SATO LM on the source node. The to-be-monitored service-related network context is identified using an UCI. Then, the SATO LM deploys the Context Sensor from the source node using the echo pattern (step 1 in Figure 56). Once deployed and activated, the Context Sensors register themselves and the UCI with the ConCoord by exercising a put(UCI, sensor-address) operation (step 2 in Figure 56), and the value pair is stored in the ConCoord through DHT message routing. The information pair is then stored in the ConCoord through DHT message routing (step 3 in Figure 56). This operation may either store the location references of the network context or the actual piece of network context itself. Once the information has been stored in appropriate location in the ConCoord, the node on which the Context Sensor is residing on will receive an acknowledgement (step 4 in Figure 56). The context client, in this case the SATO LM (i.e. the source node), may exercise a RESOLVE operation that is mapped to a get(UCI) in the ConCoord i.e. a “pull” model for context delivery (step 5 in Figure 56), which returns the sensor-address parameter to the node issuing the request. SATO LM can then obtain the

network context information directly from the sensor using the returned address (step 6 in Figure 56)

4.1.1.4 Analysis

The advantage of the integrated approach between SATO LM and ConCoord is that, because specific service context are located through the distributed ConCoord in the network (that is implemented using DHT techniques), the collected service context are distributed in the network, and are accessible to all nodes in the network, rather than made available to the source node only. As service context collected by sensors are located through the distributed ConCoord, the collected service context are made available to any nodes in the AN as soon as the REGISTER, SUBSCRIBE and RESOLVE processes are completed. There is no need to establish other connections with the source node.

The integrated approach between SATO LM and ConCoord is scalable, that is state maintenance complexity on one node is proportional to the number of its immediate neighbours (on the DHT overlay), rather than to the size of the entire network. The minimum number of ConCoord required to be established in the AN is one per physical network. This is because a ConCoord (say, implemented using a 256-bit keyspace) can host up to 2^{256} entries. Thus, the same ConCoord can be used as a location reference to all the results collected by different types of sensors that were deployed on all nodes in the network. Thus, our approach scales well.

In addition to location references, the actual piece of service context collected by sensors may also be stored through the ConCoord. The decision on whether to store location references of collected service context, or to store the actual piece of service context, depends on the type and size of service context that is being collected. For more static type of service context or small size service context e.g. node context such as CPU speed, RAM size… etc. are static information and are small in size, the actual piece of service context may be distributed through the ConCoord. However, if the type of service context is changing in real-time or the size of the service context is large e.g. flow context such as the current bandwidth usage on one interface, node context such as real-time CPU usage, location references of the collected service context should be stored through the ConCoord. The arrangement of storing location references (of the collected service context) in the ConCoord is to enable the SATO LMs to directly query the source of service context for real-time information, without exercising the RESOLVE operation. This is because otherwise when new service context (value) is available, the (old value of the) service context in the ConCoord would have to be updated, and the SATO LMs would not be able to retrieve the newest values until they have been notified.

4.1.1.5 Summary and Conclusion

Due to the dynamic nature of ANs, there is a need to investigate into a scalable and distributed approach for collecting and storing distributed network context information, to support service management in SATOs. Service-specific network context needed for managing specific services in ANs must be monitored and collected in a scalable and distributed manner. Collected network context information must also be stored in a scalable and distributed manner to ensure readily accessibility to AN service mangers i.e. SATO lifecycle managers. This section presents an inter-working approach between the SATO LM and the ConCoord to the problem space. SATO LM uses a scalable and distributed algorithm i.e. the echo pattern for rapid deployment, activation, and (re)configuration of (new) Context Sensors. Deployed Context Sensors are registered with the distributed ConCoord that is implemented using DHT techniques, to create a logical distributed database that keeps location references to the collected network context information from the Context Sensors. As such, the SATO LM inter-works with the ConCoord to provide a fully distributed approach for collecting and storing network context in order to support service management in SATO.

4.2 Self – organization in AN Underlay/Overlay

In Ambient Networks a self-organising management system is needed to automatically and dynamically determine (or to select) a set of nodes in the underlay and overlay, to be responsible for different context aggregation processes. Existing approaches rely on real-time negotiation(s) for node selection. However; real-time negotiations are expensive for bandwidth-limited environments such as ANs. This chapter presents and evaluates a novel self-organising solution to the problem space, known as the *AN Virtual Management Backbones (VMBs)*. Each VMB has a dedicated (distributed) context aggregation task, and consist of a set of dynamically selected VMB nodes, where aggregation is carried out. The selection of VMB nodes is conducted *without* heavy-weighted real-time negotiations. Our solution achieves this goal by utilising the scalability and dynamicity advantages of Distributed Hash Tales (DHTs) and ConCoord.

4.2.1 Introduction

In a large scale, dynamically changing and heterogeneous AN, with potentially many different types of network context to be aggregated, different nodes should dynamically be selected for aggregating different types of distributed network context, to enable even load balancing and to avoid centralisation.

Thus, there is a need to investigate a technique to dynamically select a set of nodes in an AN underlay and overlay, to carry out dedicated network context aggregation tasks. These nodes are known as the *Virtual Management Backbone nodes* (VMB nodes). Each VMB node is responsible for aggregating a particular type of distributed network context. If there are multiple VMB nodes in the network that concurrently aggregating the same type of distributed network context, these VMB nodes will form an overlay among themselves, known as a Virtual Management Backbone (VMB) (Figure 57). In other words, the VMB nodes in a VMB share the same network context aggregation task (i.e. they all aggregate the same type of network context). An AN node may host multiple VMB nodes.

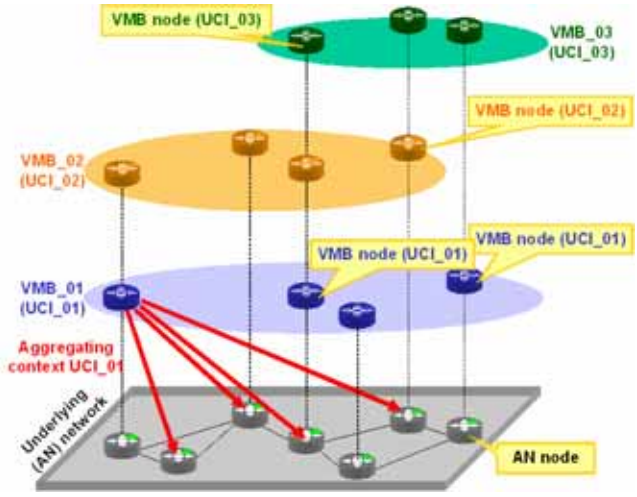


Figure 57 – Different VMB nodes and VMBs in an AN

The challenge is to dynamically select a set of nodes to become the VMB nodes for a VMB. This chapter presents a self-organising approach, known as the VMB protocol, that can automatically and dynamically determine in real-time a set of nodes (i.e. the VMB nodes) to become responsible for network context aggregations, *without* any form of real-time negotiations between participating nodes. The VMB protocol achieves this by utilising the scalability and load-balancing features of Distributed Hash Tables (DHTs). The VMB protocol also has a novel mechanism that allows us to adapt the protocol to maintain good service availability levels, to suit different levels of dynamicity and heterogeneity in networks.

We will present the fundamental concepts of our solution through the simplified VMB protocol, which is followed by the standard VMB protocol. The simplified VMB protocol is *not* our final solution, but it is discussed to outline the essential concepts of our approach based on the assumption of a homogeneous network environment[7], which enables readers to understand the additional challenges in VMB establishment. Our final solution, the standard VMB protocol, covers deeper aspects of our solution to address dynamicity and heterogeneity in ANs.

4.2.2 Problem Space Explained

We have discussed that there is a need to dynamically determine a set of nodes in an AN to become responsible for network context aggregation, in contrast to assigning tasks to nodes statically. To improve robustness, a (small) set of nodes should be assigned with the responsibility to carry out the same context aggregation task. So, should one node fail, others are there to "backup". This approach is similar to the *SuperPeer* approaches [86][87][88], in which a set of nodes in a Peer-to-Peer (P2P) network are dynamically selected/elected (through real-time negotiations) to carry out dedicated tasks. However, in a rapidly changing wireless environment with limited-bandwidth, real-time negotiations for dynamic SuperPeer election, and subsequently re-election (in order to maintain a desired number of actively running SuperPeers in the network should SuperPeers drop out) are not desirable due to the increased overhead they present. Also, the negotiation overhead is dependent on the number of nodes that are participating in the negotiation process (i.e. the more nodes going through negotiation at one time, the more overhead); which means real-time negotiations may be subjected to scalability issues.

Although it *appears* that real-time negotiations may enable peers to determine the most "suitable" peer to become a SuperPeer (e.g. the most mobility-stable peer[8], the peer with most power… etc.) for carrying out a dedicated task, in a rapidly changing network, the "suitability" of a peer to be elected as a SuperPeer may change rapidly. For example, a relatively stable SuperPeer may run out of power after an extensive real-time negotiation (hence become unstable). In addition, if the membership of the virtual network continuously changes because of old members leaving and new ones joining, then the relative "stability" or "suitability" of the SuperPeer may no longer be valid because any of the new members may now be considered more stable/suitable, according to some relevant criteria. So, *each time* there is a change in network conditions, the (remaining) peers must conduct real-time negotiation again to assign dedicated tasks to each other i.e. SuperPeer re-election. The high overhead for frequent real-time negotiations to accommodate the rapidly changing environment should be avoided. Instead, the selection process should be self-organised to reduce management overhead. We will discuss further the SuperPeer approaches in later sections.

4.2.3 Requirements of Self-Organising Node Selection Process

This chapter suggests that to enable practical network context aggregation and subsequently aggregated context dissemination in ANs, the self-organising management

[7] By homogeneous network environment, we mean all nodes are equally capable of hosting and operating any type of VMB nodes

[8] Note that mobile stability is a relative term: a stable SuperPeer is of no use if its associated peers are dynamically changing (see later).

system should be: (a) *automatic and decentralised:* there is a need of a mechanism for dynamically and automatically assigning different network context aggregation tasks to different sets of peers in the network, and subsequently disseminating final aggregated network context to others, in order to achieve this automatically and to avoid a single point of failure; (b) *efficient*: the aggregation task assignation process should be conducted with least disturbance to existing members in the network; and the use of heavy-weighted real-time communications and negotiation should be limited (if not avoided); (c) *robust and flexible*: automatic and self-organising aggregation task re-assignation and task sharing should be supported, in order to accommodate failure of nodes (that are assigned with dedicated aggregation tasks) due to high mobility and heterogeneity in networks, and to avoid a single point of failure; (d) *dynamic and scalable*: no (set of) node(s) in the network should be permanently responsible for a particular aggregation task; instead, tasks should be automatically transferable to enhance *load balancing* i.e. different types of network context aggregation tasks should be evenly distributed to all nodes in the network, rather than relying on a few set of nodes (e.g. the SuperPeers) to carry out all aggregation; (e) *distributed processing*: the network context aggregation process should be conducted across nodes to minimise overhead on the node that is responsible for aggregation;

We have mentioned that our solution uses DHTs as the implementation technique. We have investigated and evaluated in [80] the use of DHT [19][89][90] techniques to structurally organise AN nodes and distributed network context into a P2P network, for easy location and retrieval of distributed network context in an AN. The scalability and robustness (i.e. self-organising and fault-tolerant) features of DHTs have been demonstrated to be beneficial in supporting distributed management applications [99][100][101]. Whilst DHTs were originally designed for the Internet [19][89][90] i.e. a relatively resource-rich environment, the use of DHTs in resource-limited environments such as wireless networks e.g. Mobile Ad-hoc NETworks (MANETs) [96] has also been explored [98]. Other existing literatures have presented solutions to enable efficient DHT establishment [91][92], routing and content locality optimisation [84][102], and scalable and efficient DHT merging/composition and maintenance [95][103], in order to support the use of DHTs in wireless networks. In the evaluation section, we will prove that our DHT-based solution is much more efficient and scalable than existing approaches that rely on real-time negotiations.

4.2.4 Simplified VMB Protocol

The following assumptions apply to both the simplified and standard VMB protocols. We have outlined that our approaches use DHT techniques. Because DHTs are used in AN for supporting distributed network context locating and retrieval [80], we assume that a DHT is pre-established in an AN. For simplicity, we assume one DHT per AN. We assume that network context monitoring devices have been deployed in the network. This is achieved through the use of a programmable approach. The Context-Coordinator (ConCoord) we presented in [80] provides a scalable architecture for AN nodes to locate distributed network context. We therefore assume network context are accessible. Readers are referred to [80] for more detail. Thus, *enhancing* existing DHT establishment and maintenance techniques for wireless networks is out of scope of this chapter. We have already discussed that the applicability of DHTs in wireless networks have been addressed in previous research work [84][91][92][95][102][103].

We will demonstrate our approach through a 2-dimensional DHT i.e. a Content Addressable Network (CAN) DHT [19]. Our approach is not restricted to a particular type of DHT implementation. The CAN-based DHT is used simply for enhanced visualisation. For simplicity, we assume that each keyspace portion in a CAN-DHT is owned by a different node. We assume routing locality is optimised in DHTs. We have presented and justified an efficient and scalable solution in [95] to enable optimised routing locality in DHTs.

4.2.4.1 Self-Organising Aggregation Task Assignment

As we mentioned earlier, each type of aggregated network context is identified by a unique UCI. We do not restrict how UCIs can be represented; they could be Naming objects, or numerical IDs… etc. For simplicity, their names are represented as Strings (UCI_x) in this chapter. Some examples are shown in Figure 58.

UCI_01:

The location(s) in the network where a particular MP3 file is located

UCI_02:

The top three heaviest flows in the network

Figure 58 – Example UCI definitions

When an AN node i.e. a new joining node first joins an AN (of which a DHT is established), the node either obtains a copy (from existing nodes), or dynamically generates a map similar to the one shown in Figure 59. This table maps each of the defined UCIs in the AN to their corresponding points in the keyspace as defined by the hash function of the DHT. The corresponding keyspace owner of an entry in the list is the node that is responsible for hosting the type of VMB nodes that is responsible for aggregating that type of network context, and subsequently disseminating, that particular type of aggregated network context. For example, according to Figure 59, the new joining node will know that the node that owns the portion of the keyspace containing the point aaaa[9] is responsible for hosting a VMB node for aggregating (and subsequently disseminating) the network context of type UCI_01.

HASH(UCI_01) → aaaa

HASH(UCI_02) → bbbb

HASH(UCI_03) → cccc

HASH(UCI_04) → dddd

Figure 59 – Example mappings between different UCIs and DHT keyspaces

At the same time of mapping, the new joining node joins the AN's DHT, and obtains its portion of the keyspace in the AN's DHT by using standard DHT protocols. By comparing the entries in the list (Figure 59) with its recently obtained portion of the keyspace, the new joining node will immediately know which type of VMB node that it is *expected* to host. For example, if the new joining node is given a portion of the keyspace that includes point cccc, it knows that it is expected to host the type of VMB node that aggregates, and subsequently disseminates UCI_03. Similarly, the change in DHT keyspace ownership (to accommodate the new joining node) will enable other existing nodes in the AN/DHT to directly or transparently locate the new node now responsible for hosting these particular type(s) of VMB nodes.

[9] These are example keyspace points that are used to simplify our discussion. For a 160-bit DHT, these keyspace points will refer to a 160-bit keyspace identifier.

To enable readers to understand the simplified VMB protocol, a scenario is described below. In Figure 60, suppose node C is the new joining node, and node A and B are existing members of the AN/DHT. A 2-dimensional CAN-DHT is used to show DHT keyspace ownership partitioning. By using the mappings between UCIs and keyspaces, node C knows (or expects, see later) that node A, which owns the keyspace portion containing the points aaaa and bbbb, is hosting the VMB node for aggregating (and subsequently disseminating) UCI_01 and UCI_02. Similarly, the change in DHT keyspace ownership will tell existing members, i.e. node A and B, that node C is expected to host the VMB node for UCI_03. So, the assignation and determination of "who to aggregate what" and "who is aggregating what" are completely automatic, and completed without the need of any real-time negotiations between peers. No additional overhead is incurred by the simplified VMB node protocol, except for the initial setup and maintenance cost for the AN's DHT (see later).

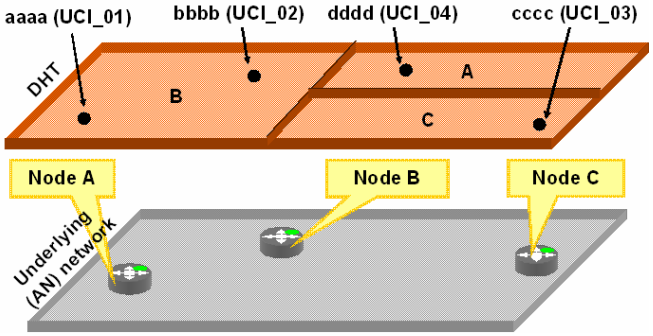


Figure 60 – A mapping between AN nodes and DHT keyspace ownership

The above example is a simplified scenario that involves only three nodes. In reality, in a large-scale AN, many more nodes are involved. Assuming random DHT keyspace partitioning, the more nodes in the AN, the more distributed the VMB nodes. In a dynamic environment, one may argue frequent computations on mapping may not be desirable. There is *no* need for real-time frequent mappings. This is because the mappings are re-usable as long as the DHT implementation is consistent. Thus, each node has to do the mapping once when it first joins the DHT (or else it may simply copy the table from an existing member, as described). Real-time computation for mappings between UCIs and keyspace would be needed only if a *new* UCI is defined and introduced to nodes in the AN.

4.2.4.2 VMB Node Activation

Once an AN node has found out which type of VMB node that it is expected to host, it is in the position to activate the corresponding VMB node to start network context aggregation. We have discussed that network context aggregation involves the collection of distributed network context, and analysis of the collected network context to compute the aggregated network context. We will discuss in this section the triggering factor(s) of network context aggregation.

The exact triggering factors of VMB node activation may vary: the VMB nodes can be started as soon as the AN node has verified which types of VMB nodes that it is expected to host; or the VMB nodes can start aggregation only after the AN node has received explicit client requests. The first approach may optionally include a fixed timeout, which allows re-instantiation of aggregation after a certain time to "re-fresh" the value of aggregated network context; or the re-instantiation can be started after a random delay time, to avoid busts of traffic at one time by many VMB nodes being (re)instantiated on the same AN node at one time. This approach makes results more readily available to clients, and is therefore preferable for situations where the type of aggregated network context is more popularly requested. The latter approach requires clients' requests. This approach is more ideal for situations where the type of network context being aggregated is less

common, or the network context aggregation process requires specific client inputs, e.g., the client must specify the name of the MP3 file that it wants to locate for UCI_01 in Figure 58.

4.2.4.3 Distributed Network Context Aggregations

Once VMB nodes are activated, they will start aggregating the type of context that they are responsible for. As we discussed, it is possible to have multiple VMB nodes in an AN that are simultaneously aggregating the same type of network context (that together they form a VMB). But for simplicity, in this section we assume only one VMB node for each type of network context in an AN at one time. We will discuss more on VMB nodes in the standard VMB protocol.

Due to space limitation, we will only discuss the concepts of an appropriate network context aggregation model. Readers are referred to [80][81] for more detail. The simplest way of aggregation is for each VMB node to query each (managed) peer in the domain directly [87], retrieve the distributed network context, and carry out local computation to determine the aggregated network context (e.g. a VMB node queries each AN node, collect all flow statistics on all AN nodes' interfaces, and calculate the top three heaviest flows in the entire network). However, this is a centralised approach, and therefore would not scale if the number of managed peers is large (see later).

We recommend a more scalable aggregation model (e.g. distributed) than the centralised approach. The echo-pattern-based [83] aggregation model presented in [81] is an ideal candidate. For example, to determine the top three heaviest flows in the network (i.e. the aggregated context), the VMB node will distribute a request to its immediate physical neighbours, which will then propagate the same request to their immediate physical neighbours, and so on, until the entire network is propagated. The request contains an execution command such as "determine the top three heaviest flows on the *local* node". During the request's propagation, a spanning tree defining the parent-child relationship between nodes, is created (with the VMB node being the root of the tree). The command in the request is executed when the request has reached the network's edges (i.e. the leaf nodes). Each leaf node determines *locally* the top three heaviest flows (i.e. the distributed network context), and returns the *initial aggregated network context*[10] to its parent node. The parent then carries out the same operation (i.e. aggregates its top three heaviest flows, and aggregate further with the initial aggregated network context returned from its child nodes), and returns the aggregated network context to its parents, and so forth. The process repeats until the VMB node has received results from all of its child nodes. The VMB node then aggregates the results from all of its child nodes and its locally collected network context to generate the final aggregated network context. Thus, unlike centralised approaches, the workload on the VMB node depends on the number of its immediate physical neighbours only, instead of the size of the entire managed domain. In this way, the VMB node achieves scalable and distributed aggregation processing.

4.2.4.4 Self-Organising Task Transfer & Recovery

One advantage of the (simplified/standard) VMB protocol is that no one node is permanently responsible for aggregating a certain type of network context, but the

[10] The initial aggregated context is an aggregation of all the *local* context that a node can collect *locally*. Thus, it does not present the *entire* network-wide view.

aggregation task (or aggregation responsibility) is *transferable*: this is known as *task transfer and recovery* in this chapter. The advantages of task transfer and recovery are: (a) a VMB node's assigned aggregation tasks may be automatically transferred to other nodes to enable more even load balancing; (b) task recovery handles situations when a VMB node fails to carry out its assigned aggregation tasks, without the need of real-time re-negotiation. Hence efficiency and robustness are enhanced.

Task transfer is triggered when a new node joins the network; and task recovery is triggered when a node leaves the network. Using Figure 60 as an example, suppose that before node C joins, the keyspace point cccc was originally owned by node A. In other words, before node C joins, node A hosted the VMB nodes for UCI_03 and UCI_04. When node C joins, node C takes up some of node A's original keyspace (which, say, includes the point cccc). As such, one of the node A's original tasks (i.e. to host the VMB node for UCI_03) is now dynamically *re-assigned* (or transferred) to node C in real-time. When node A hands over to node C, node A may also handover the current value of the final aggregated network context of UCI_03 to node C, as long as the value is still valid (e.g. has not expired). Thus, with task recovery, no one node is permanently responsible for aggregating or disseminating a particular type of network context. Assume random keyspace partitioning, aggregation tasks will be evenly distributed in the network, hence *load balancing* is achieved.

Task recovery enables the automatic and efficient recovery of aggregation tasks when a VMB node fails, and no explicit notification is needed to inform other peers in the network about the failure. To visualise this, imagine node C (Figure 60) suddenly drops out. Node C's keyspace, and its task (i.e. to host a VMB node for UCI_03) will be automatically taken up by the node that owns the keyspace portion that is immediate neighbouring to node C's original keyspace (there could be several neighbours, which neighbour to take over will depend on the underlying DHT recovery protocol). The "taking-over" node will automatically become the host for the (missing) VMB node(s). Thus, no real-time re-negotiation would be needed, and no notification is needed to notify other nodes in the network for task recovery. Thus, with task recovery, the (simplified/standard) VMB protocol is much more scalable and efficient.

4.2.4.5 Practicability of the Simplified VMB protocol

We have explained that the (simplified/standard) VMB protocol uses task transfer and recovery to enhance robustness with much less overhead (see later for evaluation). However, in a rapidly changing network, task recovery may still generate some overhead i.e. every time a node drops out, its roles must be transferred to its overlay neighbour. The "missing service" that was expected to be provided by the failing VMB node(s) would not be in service, until the unoccupied keyspace (that was left over by the dropping out node) has been re-occupied by another node. This is known as the *dynamicity issue* in this chapter. Also, so far we have assumed that a node, once determined its expected aggregation task through mapping between UCIs and keyspace points, is capable of hosting the type of VMB node that it is expected to host. In a heterogeneous environment, this would not always be the case, as some nodes would not be capable of hosting VMB nodes (e.g. due to low processing power, poor communication links, and others). This is known as the *heterogeneity issue* in this chapter. We present in the next section the standard VMB protocol, which addresses these issues.

4.2.5 Standard VMB Protocol

The standard VMBB protocol addresses the dynamicity and heterogeneity issues by requiring the neighbouring overlay nodes to a node that is expected to host a VMB node (i.e. the original VMB node) to host the same type of VMB node. In other words, the neighbouring overlay nodes host the *deputy* VMB nodes. For example in Figure 61a, suppose a node is expected to host, say, a VMB node for UCI_01 (the one with a dark circle)[11]. Each of the nodes that own the keyspaces that are immediate neighbours of the original node's keyspace (i.e. those marked with an orange triangle) must also host a VMB node for UCI_01. If the original VMB node is unable to carry out its aggregation task (due to broken links or simply not capable of) i.e. it is becoming an *incapable node*, the incapable node's immediate overlay neighbours (i.e. the deputy VMB nodes) will act as the automatic "backups" to the incapable node.

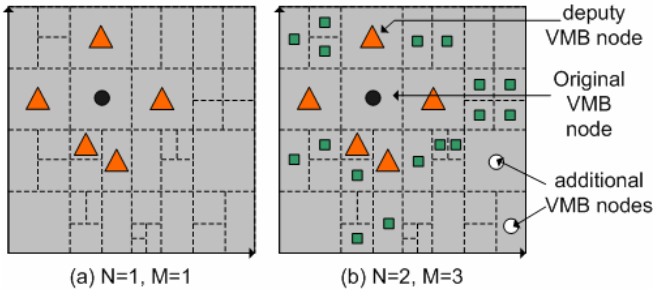


Figure 61 – The standard VMB protocol with different values of N and M

If a new joining node (after computing the type of VMB node that it is expected to host) knows that it is not capable of hosting the VMB node that it is expected to host, it will simply not host the VMB node. If this incapable node is queried by clients for the final aggregated network context, the incapable node will query its immediate overlay neighbours (which host the deputy VMB nodes) for the aggregated network context of interest, update its cache with the aggregated network context obtained from its neighbours, and respond to the client as if it was the source of the result. Optionally, the incapable node may also inform the client through its response that it is in fact an incapable node, and refer the client to one of its capable neighbours so that next time the client can avoid contacting the incapable node. The caching is useful because the incapable node can respond to other clients without having to query again its capable neighbours (as long as the value of the aggregated network context has not expired). The caching does not add additional overhead on the AN nodes because under normal circumstances, aggregated network context are cached until they expire anyway (as discussed). A small round trip delay between the incapable node and its overlay neighbours would be incurred. This time delay could be minimised if DHT routing locality is optimised (we have discussed in [95] how to optimise routing locality in DHTs). As such, we avoid totally the expensive re-negotiation process that would otherwise be needed. The scale of the query by an incapable node is limited because the number of immediate overlay neighbours to a node

[11] In the literature the deputy VMB nodes (i.e. immediate overlay neighbours) are shown using a 2-dimensional CAN-DHT (as explained). If a ring-based DHT was used (e.g. Chord), the deputy VMB nodes would be the immediate successor and the predecessor of the original VMB node. The physical hop between these nodes are limited if routing locality is optimised. We have presented in [95] an efficient and scalable way to enforce routing locality in DHTs.

on the DHT overlay is always limited (see later for evaluation). As such, the time needed by an incapable node for searching sequentially (or simultaneously) through its (limited number of) neighbours is minimal[12]. Another workaround would be to have an incapable node querying its overlay neighbours at random times even without receiving requests from clients (so that the results are always consistently available on the incapable node to serve clients' requests promptly); the query message can be included in the DHT maintenance messages sent by the incapable node to its neighbours (i.e. the "are you still alive" messages). This may create some additional overhead to the overall system; but we will show in the evaluation section that the magnitude of this additional overhead is negligible when comparing to existing approaches.

4.2.5.1 Controlling the Deployment of the Standard VMB protocol

We introduce two scales, known as the neighbourhood scale *N*, and the network scale *M*, to control the scale of deployment of the standard VMB protocol. Both *N* and *M* are introduced to enhance robustness by addressing network dynamicity and heterogeneity, by enabling efficient and robust recovery and backup processes; however, they are designed to accommodate different levels of routing locality. *N* increases the number of VMB nodes of the same type in the network by requiring neighbouring overlay nodes of the original VMB node to host the same type of VMB node, *when routing locality is not optimised*. *M* also increases the number of VMB nodes of the same type in the network, but by increasing additional VMB nodes at *random locations* in the network. *M* is used when routing locality *is* optimised. The reasons for having *N* and *M* will be discussed further shortly. *N* and *M* may be used simultaneously to control robustness (see later).

N determines the radius of the deployment scale of the standard VMB protocol: for example when *N*=0, no immediate overlay neighbours of the original VMB node are expected to host the same type of VMB node (i.e. the simplified VMB protocol). When *N*=1, the original VMB node's immediate overlay neighbours (i.e. those marked with an orange triangle in Figure 61a) are expected to host the same type of VMB node. These neighbouring nodes are known as the N^{th} *wave* (of a particular type of VMB node) in this chapter. Similarly when *N*=2, the immediate overlay neighbours to the nodes of the 1^{st} *wave* are expected to host the same type of VMB node (i.e. those marked with a green square in Figure 61b); these nodes are known as the 2^{nd} *wave*… and so on. As a peer in a DHT overlay does not keep states of peers beyond its immediate overlay neighbours (thus giving DHTs their inherent scalability); when *N*>1, nodes of the first *N* wave are responsible to inform their immediate overlay neighbours (i.e. nodes of the 2^{nd} *wave*) the type of VMB node that they should host and the value of *N*[13].

The network scale parameter *M* has a value of 1 when there is only one VMB node (of a particular type) in the network (i.e. the node marked with a dark circle in Figure 61a). *M*=3 when there are three VMB nodes (of the same type) in the network (i.e. three nodes in

[12] Note that each incapable node searches through its immediate overlay neighbours only. This rule applies not only to the (original) incapable node, but also to the deputy and "*additional"* VMB nodes that are failing (see later for additional VMB nodes). As such, the scale of the search is limited.

[13] This can be done through the DHT maintenance messages exchanged between neighbouring peers. The scale of maintenance on each node does not increase because each node is still only responsible to maintain state of its immediate overlay neighbours only.

Figure 61b are marked with either a dark or hollow circle[14]). The additional VMB nodes (s) when *M*>1 are determined by *multiple hashing* the UCI of the network context that the VMB node is responsible for, which returns other keyspace point(s) in the DHT (i.e. other than the keyspace point of the original VMB node) (Figure 62). The owner of the other keyspace point is the node that is expected to host the "additional" VMB node (of the same type).

When *M*=1:

HASH(UCI_01) → keyspace point of VMB node #1

When *M*=2:

HASH(UCI_01) → keyspace point of VMB node #1

HASH(HASH(UCI_01)) → keyspace point of VMB node #2

Figure 62 – Multiple hashing for M > 1

The values for *N* and *M* are determined by the *failure rate*, which in turn depends on the level of dynamicity and heterogeneity of the network. The more mobile the network, or the more heterogeneous the network, more peers are likely to be incapable of hosting the type of VMB node that they are expected to host. We therefore increase the value of *N* to create spheres of influence of *different sizes*: if the failure rate is high, *N* is increased, to maintain a constant number of capable nodes (that are capable of hosting the same type of VMB node) at one time. It should be noted that the number of actively running VMB nodes of the same type (known as *active VMB nodes* in the rest of this chapter) in the network at one time is dependent on *both* the value of *N and* the failure rate: for example, *N*=1 in Figure 61a, assuming 0% failure rate, the total number of nodes that are expected to host a (particular type of) VMB node would be six. But assuming a 20% failure rate, the (average) total number of active VMB node (of that particular type) would be 6*0.2 = 4.8. So, we can adjust our protocol using the value of *N* (and *M*), to accommodate different network environments (i.e. different failure rates).

Note that through *N*, we run duplicated VMB nodes on *overlay* neighbouring nodes, instead of physical neighbouring nodes. This arrangement is for two reasons: (a) according to the DHT protocol [19], an overlay neighbour is the next node that "takes over" when a node fails, which is not necessarily the incapable node's physical neighbour. By having neighbouring overlay nodes pre-running the same type of VMB node, should a VMB node fails, the pre-running VMB nodes (on the neighbouring overlay nodes) can come into immediate action; (b) to avoid uneven loading in the network (see later).

The network factor *M* serves a similar purpose but for a different reason. Note that with random DHT keyspace partitioning [19], immediate DHT overlay neighbours are unlikely to be physical neighbours on the physical network (i.e. the routing locality problem). Therefore if *N*>0, the set of VMB nodes should be (statistically) evenly distributed in the physical network. But if optimal routing locality is enforced (that the overlay network topology maps with the underlying physical network topology), and *N*>0, the set of VMB nodes will most

[14] For simplicity, the neighbouring VMB nodes to two of the three VMB nodes (i.e. the ones marked with a hollow circle) were not displayed in the literature. Note that in CAN-DHTs, neighbours may wrap around the edges of the key space.

likely be located in the nearby physical neighbourhood, possibly causing *uneven loading* in the network. This is why the network factor *M* is introduced: if the administrator would prefer not to set the value of *N* to too high (due to, say, optimised routing locality in the DHT overlay), the administrator can still control the robustness of the standard VMB protocol by adjusting the value of *M*. As shown in Figure 62, different values of *M* points to different points in the keyspace. Assuming the scale of network is large (i.e. large keyspace), the additional VMB nodes should therefore be located (statistically) evenly in the keyspace (hence located evenly on the physical network). Hence, uneven load balancing caused by the use of *N* with optimal routing locality is resolved. Note that increasing the value of *N* and *M* does *not* increase network traffic proportionally. This is because the values of *N* and *M* should be adjusted to accommodate *different* failure rates (i.e. the higher the failure rate the higher the values of *N* or *M*). We will discuss further the impact of *N* (and *M*) in network traffic in the evaluation section. Note that *N* and *M* may be used together to control robustness. For example, if routing locality has been optimised, *M* should be used to enable more VMB nodes in the network; but each additional VMB node should still be accompanied by some surrounding backup nodes for enhanced robustness should the VMB node fail. Therefore, the use of *N* and *M* are not mutually exclusive.

4.2.5.2 Enhanced Efficiency, Robustness, Scalability, and Decentralisation

The standard VMB protocol is designed for efficiency, robustness, and scalability. Under the standard VMB protocol, a client which needs access to one type of aggregated network context (say, UCI_01) need not access the original VMB node; as its request routes its way through the DHT to the original VMB node, the requests will be serviced by the first intercepting node that hosts the same type of VMB node. This enhances scalability in the sense that more VMB nodes of the same type are available in the network at one time to handle clients' requests; and it also enhances efficiency because a client may be serviced by the nearest VMB node.

Furthermore, we have explained earlier that if the incapable node drops out, the incapable node's keyspace will be automatically taken up by one of its immediate overlays' neighbours. But then the immediate overlay neighbour that is taking over the keyspace would have to instantiate the VMB node, and start aggregating information, and so on, thus, a frequent recovery process as such may add unnecessary overhead to the nodes in the network. However, with standard VMB protocol, the incapable node's immediate overlay nodes are already running the same type of VMB node, so should the original VMB node drop out, there is no need for real-time instantiation of VMB nodes. So, the overhead on the node "taking over" would be much less during the keyspace hand over process. This idea is similar to having distributed tasks running at random times to avoid a burst of traffic in the network if they were executed simultaneously. Furthermore, the take-over process is automatic without any form of re-negotiation. Thus, we enhance robustness without incurring additional overhead. Also, having more than one VMB node actively running in the network at one time provides a more decentralised system i.e. avoiding a single point of failure.

4.2.5.3 Loading on Nodes and VMB nodes

One may argue that the described process enhances the level of decentralisation and robustness at the expense of some efficiency because more than one node at one time is now carrying out (duplicated) distributed network context aggregation and dissemination. In a small size network and a stable environment (i.e. 0% failure rate), one node per one type of VMB node may be sufficient. But in a large network, more than one VMB node of the same type should be in the network to handle potentially large number of client requests (i.e. to avoid single point of failure or overloading).

There is no need to have the *same type* of VMB nodes (i.e. aggregating the same type of context) simultaneously *aggregating* network context *at the same time* as it would be a waste of resources, because the aggregations are duplicated. Instead, some form of co-

ordination is needed between VMB nodes of the same type in the network, to carry out aggregations in a sequential order to minimise the effect of aggregation processes. When instantiating the deputy VMB nodes, the original VMB node will calculate a random time delay for *each* deputy VMB node, and pass on the specific random time delay and timeout value of the aggregated network context (which is pre-determined a constant value that is specific for the type of aggregated network context) to the deputy VMB nodes respectively. Each deputy VMB node starts its own aggregation after the specific random time delay, and re-instantiates its aggregation process when the aggregated network context times out. In this case, we avoid simultaneous running aggregation threads. To minimise the overhead on other nodes (other than the VMB nodes) in the network due to the distributed network context aggregation processes that are instantiated by a set of decentralised VMB nodes (at random time), these other nodes (i.e. the child nodes explained in an earlier section) may cache the aggregated network context until the context expires (the time out value for the aggregated network context are either pre-determined, or obtainable when the child node first receives an aggregation request for that type of network context). The cached results can be re-used if the child node receives the same aggregation request from a different VMB node. Lastly, readers should also note that we have discussed how to control the number of actively running VMB nodes in the AN, by adjusting the value of *N* and/or *M*. If there are too many simultaneous aggregations going on and causes overloading or congestion, the number of actively running VMB nodes is adjustable through *N* and *M*.

Given that the VMB nodes are evenly distributed in the network, and the VMB nodes may intercept and respond to clients' requests, potential overloading on one VMB node is avoided. Furthermore, since distributed (or centralised) aggregation process takes time because of communication delays between nodes to pass on (distributed) aggregated results, having several simultaneously aggregating VMB nodes in the network will make aggregated network context *readily available* to clients.

4.2.5.4 Related Work

Although the aims of the standard VMB protocol and the SuperPeer approaches may be similar, i.e. both are trying to support (distributed) management operations in P2P networks, the two approaches attempt to solve the problem using entirely different methodologies. The reason is that the two approaches have different fundamental assumptions (to be discussed shortly). The fundamental differences between any protocols that involve negotiations and the VMB protocols are: (a) the approaches that require real-time negotiations attempt to filter out incapable (or less desirable) nodes *before* task assignment, but at the expense of incurring potentially high overhead for dynamic negotiations and renegotiations every time there is change in responsibilities. Furthermore, in a highly dynamic environment, the "suitability" of a node may change in no time; (b) under the standard VMB protocol, incapable nodes are filtered out automatically *after* they have automatically determined their "expected" aggregation tasks. No negotiation is needed at the first place and no re-negotiation is needed for task recovery. Furthermore, no notification is needed to announce new responsibilities.

Existing SuperPeer approaches either assume SuperPeer election or assume SuperPeer existence [86][88][94], or rely on real-time communications and (re)negotiations for SuperPeer election in order to identify and to locate more capable nodes in the network to carry out dedicated tasks [87]. This implicitly assumes that the network is more resource-rich (e.g. more stable, or has more available bandwidth, etc.) [86][87][88]. For example in [86], the need for SuperPeer election was addressed; but how the election was conducted was not discussed; only some general guidance such as election criteria such as connectivity quality, node processing power, security requirements. etc. was provided. In [94], a general theoretical model for determining the best peer in the network was presented; but the actual peer selection implementation technique was not addressed. In

[88], a SuperPeer election approach with a static bootstrapping was presented: the first group of nodes joining the network were automatically assumed to be the SuperPeers. It was described in [88] that new SuperPeers would be selected by existing SuperPeers (for the purpose of load balancing or to recover from previous SuperPeer failure) through a "black box" volunteer service (which maintained information about the capabilities and willingness of individual peers). But no detail was provided on how this black box volunteer service was implemented.

In [87], a small set of peers in the network were "converted" into SuperPeers, by measuring their response thresholds to broadcast messages. In [87], SuperPeers were more privileged peers, so (normal) peers were under direct control by their corresponding SuperPeers. Thus, the number of state maintenance on each SuperPeer is dependent on the number of its *directly managed peers* i.e. the size of its managed domain. The developers of [87] suggested that the number of SuperPeers is kept small in order to minimise the overhead incurred by the "competitions" between SuperPeers: within the same network, SuperPeers must compete with each other to gain control of more peers. The competition process involves *each* SuperPeer *broadcasting* its availability in the network. The drawback of [87] was that because state maintenance on each SuperPeer is dependent on the size of its managed domain, should the scale of deployment be large (i.e. many peers), having only a small number of SuperPeers would result in potential overloading of the SuperPeers i.e. centralised maintenance of peers. Although a workaround would be to reduce the size of each managed domain; having more SuperPeers would mean the competition would be more intense i.e. more broadcasting would be needed. Furthermore, broadcast was also used in [87] for client searching for SuperPeers. The use of broadcast is less desirable for bandwidth-limited wireless networks. On the other hand, the approach in [87] could be ideal for electing SuperPeers in a more static and resource-rich environment. The standard VMB protocol does not incur additional maintenance overhead except the initial cost for setting up. Network maintenance is also needed by any SuperPeer approaches that follow a structured P2P approach for dynamically electing SuperPeers[15] [86][87][88]. In addition the SuperPeer approaches incur *further* overhead for dynamically re-assigning roles between peers and SuperPeers should SuperPeers fail i.e. SuperPeer (re)election; whereas the standard VMB protocol does not incur any *additional* overhead for task re-assignation (see later).

In contrast, the standard VMB protocol has the following advantages: each peer maps the list of tasks to a list of keyspaces to determine its own tasks and other's tasks. Thus, network context aggregation tasks are *automatically* assigned to peers in the network, through which peers are made aware of their own tasks and also other's tasks, *without* the need of any form of real-time negotiations [86][87] or "black box" services [88]. Also, there is no need for any form of out-of-band notification channels to advertise task responsibility of each peer [87]. This makes the standard VMB protocol *automatic*, and much more *scalable* and *efficient*. We have discussed how the standard VMB protocol enables self-

[15] Peers must first form a P2P network, from which SuperPeers are elected. SuperPeers then manage a set of peers. The managed peers do not necessary to be the immediate P2P network overlay neighbours to the SuperPeer, but could be any peer in the network that wants to be managed by that SuperPeer (e.g. consistency between policies… etc.). Each node in the network must therefore maintain states of its immediate P2P network neighbours, *and* if the node is a SuperPeer, it must also maintain state of the peers that it is managing.

organising task recovery, again *without* any form of real-time (re)negotiation. As such, the standard VMB protocol achieves *robustness* at no additional overhead.

In contrast to existing SuperPeer approaches, in which a set of nodes would become more privileged than others, the standard VMB protocol is *truly decentralised*: each peer may host VMB nodes. Also, unlike a SuperPeer, which maintains states of its immediate P2P neighbours and peers that are directly under its control, a VMB node maintains states of its immediate overlay neighbours only. Unlike the approach in [87], which limits the number of SuperPeers to a small number, the standard VMB protocol has no limitation on the number of VMB nodes in the network, and multiple VMB nodes of the same type form a VMB that provides clients with better access to aggregated network context, which makes the standard VMB protocol more *flexible* and *decentralised* and with *more even load balancing* across the peers. In contrast to [87], where state maintenance on each SuperPeer is related to the size of its *entire* managed domain, (but the size of managed domain grows as the size of network grows), the *only* form of state maintenance on each peer under the standard VMB protocol is DHT maintenance (i.e. to ensure one's neighbours are still alive). Thus, under the standard VMB protocol, state maintenance on each peer is always restricted to the number of its *immediate* neighbours on the DHT (we will see later that this number is strictly limited even when the scale of the DHT overlay is large). This makes the standard VMB protocol more *scalable*. The following table (Table 7) summaries a list of characteristics of different approaches.

	Standard VMB	SuperPeer approaches [87]
Initial setup cost	P2P network setup through DHT	P2P network setup
Overhead for task assignation/SuperPeer election	Low: automatic and no real-time negotiations	High: real-time negotiations through broadcast
State maintenance per VMB node/SuperPeer	Low: depends on the no. of the VMB node's immediate DHT overlay neighbours, conducted using standard DHT maintenance messages	High: depends on the no. of the SuperPeer's immediate P2P network neighbours & no. of peers in the managed domain, in addition to the overhead of standard P2P network maintenance
Overhead for VMB node/SuperPeer failure management	Low: automatic recovery through task recovery, no real-time re-negotiation	High: real-time re-negotiations/communication through broadcast
Load balancing	Completely decentralised, thus (statistically) evenly distributed in the physical network	Uneven loading due to some level of centralisation caused by the use of SuperPeers to directly manage peers in a domain
Level of robustness to tackle different levels of dynamicity and heterogeneity	Flexible: dynamically adjustable by *N* and *M*	Not addressed

Deployment environment	Ideal for dynamic and heterogeneous environments with limited-bandwidth with low(er) overhead	For relatively resource-rich(er) and more static environments with a high(er) overhead involved

Table 7 – A summary between different approaches

4.3 Policies for AN Underlay/Overlay Optimisation

This chapter introduces the concept of policy-based hierarchical overlay structures.

4.3.1 Hierarchical overlay structures

Hierarchical overlay structures consist of overlays arranged in multiple layers in a tree-like structure. Overlays on the bottommost level are clusters of nodes. Overlays on higher layers on the other hand are clusters of lower level overlays. In the AN setting nodes correspond to the ANNs (AN Nodes). The overlay structure itself can be considered as a set of overlays above an underlay formed by the ANs.

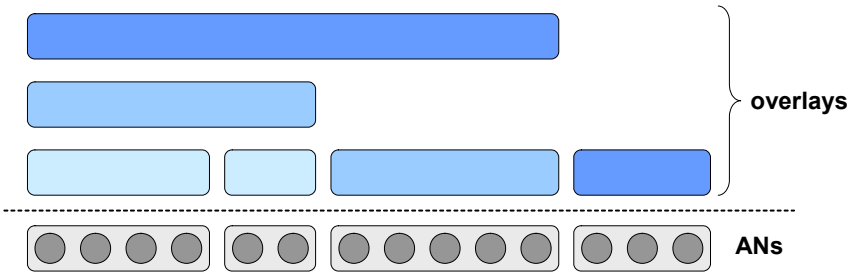


Figure 63 – Hierarchical overlay hierarchy

Many types of tasks in the network benefit greatly from the existence of a hierarchical overlay structure in a number of ways, most notably robustness and scalability. Moreover different tasks can create their own overlay structures that fulfil their needs especially. Of course these hierarchies also have to take into account different constraints regarding e.g. security domains.

4.3.2 Applications

An example would be the collection of status information from services in the nodes for monitoring purposes. In each layer the information is aggregated until a complete snapshot of service status within the network is formed in the uppermost layer.

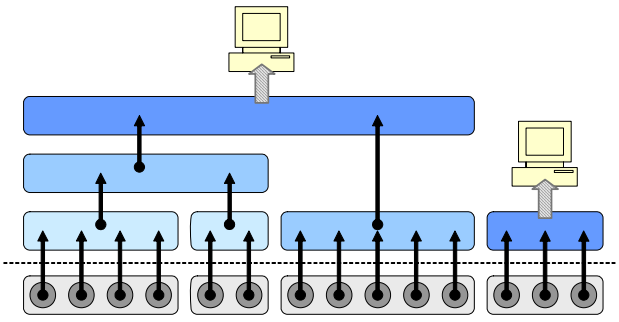


Figure 64 – Example: monitoring

The overlay structure does not necessarily have a tree topology. The topology is determined by a number of requirements, including the structure of the task in question, security domains, management decisions, etc. An additional factor influencing the topology

is the fact that the structure of the underlay (i.e. that of the ANs themselves) changes though composition and decomposition.

The goal is to be able to build and maintain a hierarchical overlay structure optimal or close to optimal for a given task, while taking the above requirements and composition into account.

4.3.3 Policy-driven structure

The hierarchical overlay structures introduced in this chapter use a set of policies to achieve the required goals, which are continuously updated as necessary. This approach allows the structure to be self-organizing through structural changes dictated by policy changes or AN composition events. Policies are changed in response to changes in requirements. To minimize the overhead associated with overlay structure rearrangement, such rearrangements must be focussed as tightly as possible.

Policies in the hierarchy are assigned as follows. Each ANN has set of policies relevant to the hierarchical overlay structure. ANNs that have compatible policies are members of the same overlay on the bottommost layer. Overlays on the bottommost layer may be combined into an overlay on a higher level, by extracting policy elements common to all bottommost overlays in question. As a consequence different hierarchy layers represent different levels of detail. The bottommost layer contains highly detailed policy information. Higher layers are decreasingly specific. Theoretically it is always possible to create an uppermost layer with a single overlay (i.e. a tree structure instead of a forest). However, in typical cases this overlay has an empty policy.

4.3.4 Policy maintenance

The maintenance of such a hierarchical structure is based on two primitives: joining and splitting. When two or more such structures come into contact with each other (for example due to a change in the underlying network topology), they must be somehow merged into a single structure. An algorithm for performing this task is the following:

- The two structures come into contact with each other through two nodes on the bottommost level discovering each other.
- Starting from the bottommost level, move upwards until the uppermost level of one of the structures is reached.
- Test the policies of the two overlays for compatibility.
- If they are mutually compatible, the two overlays can be joined, forming a single overlay, whose set of children is the union of the sets of children of the original overlays.
- If mutual compatibility does not hold, perform the splitting operation. The splitting operation involves performing a clustering on the sets of policies of the children of both overlays. The result of the clustering defines a number of overlays each with consistent policies. The resulting overlays have their own set of children from which the overlay's joint policy has been formed.
- Create a higher level with a number of overlays consisting of the overlays resulting from the clustering process.

The algorithm described above may be executed at any time as a maintenance operation, for automatically optimizing the structure for new requirements in response to policy changes.

4.3.5 Conclusion

Using this structure and the algorithm a hierarchical overlay structure optimised for a given task can be maintained automatically. This approach takes advantage of the multiple overlay levels interacting with the underlay reflecting the physical structure of the network.

4.4 Security Domain Management in ContextWare

An appropriate security approach is needed in the ContextWare (CW) for protecting and for enabling negotiated inter-operator access to and dissemination of context information. In general the following main security issues should be considered:

Access control: Only authorised entities should be granted access to the context information in accordance with the agreed user privacy and network secrecy policies.

Authorisation: Use, processing (e.g. aggregation, inferring) and dissemination of context information need to be controlled and the corresponding policies enforced.

Integrity and confidentiality protection: The authenticity and integrity of context information should be ensured in order to prevent unauthorised manipulation during context transfer or storage. Depending on the context type (and the corresponding policy), end-to-end encryption of context information may be required.

Performance: A good balance between the level of provided security and the overall ContextWare performance should be found, for example, in conjunction with the Quality of Context (QoC) to be achieved.

During AN phase 1 a privacy-sensitive authorisation framework based on the IETF GeoPriv framework has been proposed [44]. The access control and authorisation should be carried out at two levels by different CW entities: The basic access control and authorisation will be performed by the Context Coordinator (ConCoord), and a finer grained access control by the Context Manager (CM) or Context Source (CS). The applicability of the security domain concept[16] on supporting this approach has been investigated in AN phase 2.

4.4.1 Security Domain Concept

A security domain is a natural grouping of resources or nodes which follow the same set of security policies [45]. Every security domain is controlled by a member entity called the *domain manager*. The manager issues public key certificates, wherein the member's identifier (e.g. Node ID) is signed with the manager's private key. These certificates will mainly be used by the members for authentication and proof of domain membership. Access requests from the domain manager are controlled by the *meta policies* (denoted by p_0), while the *member policies* (denoted by p_1) set the rules for requests from members of the security domain. The security domains will be created and members enrolled during bootstrapping, network attachment and composition phases. The impacts of the resulted security domain memberships of different CW entities on the access control mechanisms in ContextWare are discussed in the following subsections.

4.4.2 Access Control in ContextWare

There is the need to strike a balance between the speed of retrieval and the level of guaranteed security. We thus anticipate that the use of security domain can be justified

[16] As identified as one technical issue by WP-A and described in the AN System Description document.

only when the complexity of its setup and having entities enrolled into it is offset by the achieved gain. We aim to have a huge number of queries automatically secured without the need for individual query authorisation which would otherwise incur in added delay and longer retrieval time. The security domain approach is therefore envisaged for a coarse grained access control as explained hereafter.

In order for a certain AN's Context Source to use the services of that particular AN's ConCoord, the source must be a member of the same security domain as the ConCoord and the Context Clients. Schematically, this assumption can be seen in Figure 65 where Context Source (CS#1 and CS#2) membership to the security domain β identifying, say, AN_X means that the context sources own "AN_X public" information that can be disclosed without any further checks to the clients (CC#1) members of β (according to its member policy $p_0(\beta)$). Each Context Source can then decide to further restrict access to the information it owns and run the corresponding finer grained (private) policies $p_{priv,1}$, $p_{priv,2}$ etc. outside the security domain agreements. As indicated in the figure each security domain member has the domain manager's public key which enables it to verify the domain membership (i.e. member certificate) of any other members[17]. The exchange of such public keys has already taken place beforehand, e.g. during network bootstrapping and configuration phase. Note that every security domain related task, e.g. verifying domain membership or checking member policies, will normally be carried out by the appointed *node manager* entity. Member certificates are issued to node managers identified by their Node ID. One possible configuration depicted in Figure 65 is that the Context Client, Context Source and ConCoord are implemented in different nodes, but belong to the same ambient network.

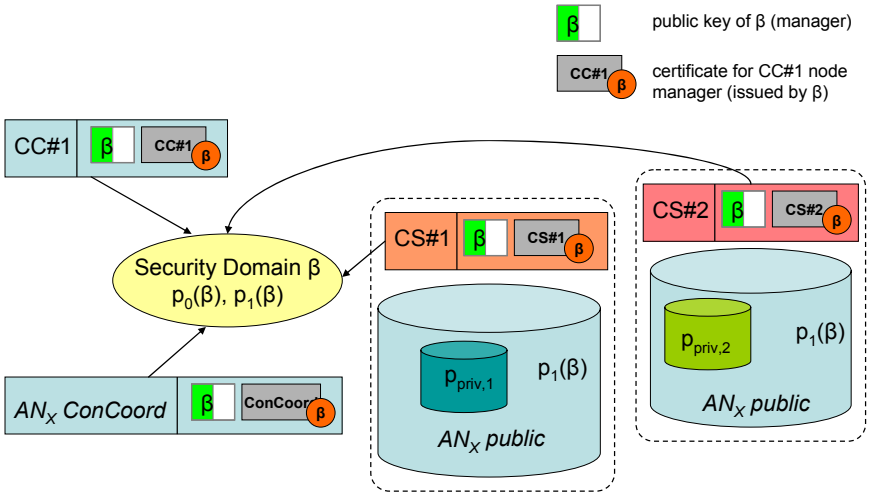


Figure 65 – Security domain management for ContextWare objects

Figure 66 sheds more light on the sequence of events taking place after a client CC#1 wants to resolve a particular UCI_1:

- CC#1 sends AN_X ConCoord a request to resolve the context source address of UCI_1.

[17] Note that the domain manager itself is not shown in the figure.

- ConCoord verifies CC#1 identity and its membership in the security domain β.
- On successful verification ConCoord resolves UCI_1 by consulting its DHT network.
- ConCoord provides the UCI_1 source address (i.e. CS#1) to CC#1.
- CC#1 sends the request for context information (identified by UCI_1) to CS#1.

CS#1 checks if the requested context is "AN_X public", i.e. accessible by members of security domain β[18]. If CS#1 had assigned a specific set of (private) policies to UCI_1:

- CS#1 runs the corresponding policy $p_{priv,1}$ with regard to access by CC#1.
- If the policy permits the access, CS#1 can continue the context provisioning to CC#1.
- CS#1 provides the context information (e.g. an SNR value of 20db) to CC#1.

As shown in the example above, the benefit of employing the security domain approach is that CS#1 needs not to run the corresponding access policy sets anymore if the requested context had been declared public to AN_X or security domain β. In this case CS#1 node only needs to make sure that the Context Client node (i.e. CC#1) is a member of the security domain.

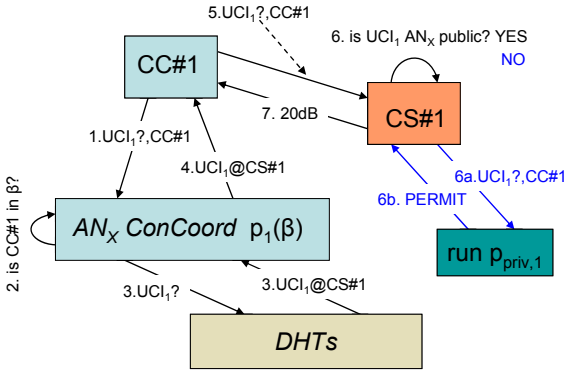


Figure 66 – ContextWare access control within a single AN (security domain)

4.4.3 Inter-AN ContextWare Access Control

In order to share resources such as context information, two (or more) ambient networks may compose and interwork according to the composition agreements. The agreements comprise the negotiation of access policies for requests from members of the other (security) domains. Figure 67 depicts an example of context information sharing between two composed ambient networks AN_X and AN_Y which consist of the security domains β and φ, respectively. Both security domains (i.e. their managers) have agreed on the set of member policies $p'_1(\beta)$ and $p'_1(\varphi)$ to control the access requests from the other domain; for example: $p'_1(\beta)$ are the policies for requests from members of domain β for accessing "AN_Y public" context information. Note that each domain member may have access to the manager's public key of the other domain which allows it verify other domain's

[18] CS#1 may wish to verify the security domain membership of CC#1 beforehand, but this may be skipped if CS#1 assumes that the verification already took place at AN_X ConCoord.

membership. The following sequence of events may take place after a client CC_X#1 wants to resolve a particular UCI_1:

- CC_X#1 sends AN_X ConCoord a request to resolve UCI_1.
- AN_X ConCoord verifies CC_X#1 identity and its membership in the security domain β.
- On successful verification AN_X ConCoord locates the UCI_1 source in AN_Y; it then sends a request to the DNS server to resolve the IP address of AN_Y ConCoord.
- The DNS server provides the IP address of AN_Y ConCoord to AN_X ConCoord.
- AN_X ConCoord forwards the request to AN_Y ConCoord (member of security domain φ).
- According to the agreed member policy $p'_1(\beta)$, AN_Y ConCoord may need to verify the membership of AN_X ConCoord in the security domain β.
- On successful verification, AN_Y ConCoord provides the UCI_1 source address (i.e. CS_Y#1) to AN_X ConCoord.
- AN_X ConCoord forwards the UCI_1 source address to CC_X#1.
- CC_X#1 sends the request for context information (identified by UCI_1) to CS_Y#1.
- CS_Y#1 checks the membership of CC_X#1 in security domain β; if the verification fails CS_Y#1 denies the access.
- CS_Y#1 then checks if the requested context information is "AN_Y public", i.e. accessible by members of security domain β in accordance with the policy set $p'_1(\beta)$. If CS_Y#1 had assigned a specific set of (private) policies to UCI_1:
- CS_Y#1 runs the corresponding policy $p_{priv,1}$ with regard to access by CC_X#1.
- If the policy permits the access, CS_Y#1 can continue the context provisioning to CC_X#1.
- CS_Y#1 provides the context (e.g. an SNR performance of AN_Y of 20db) to CC_X#1.

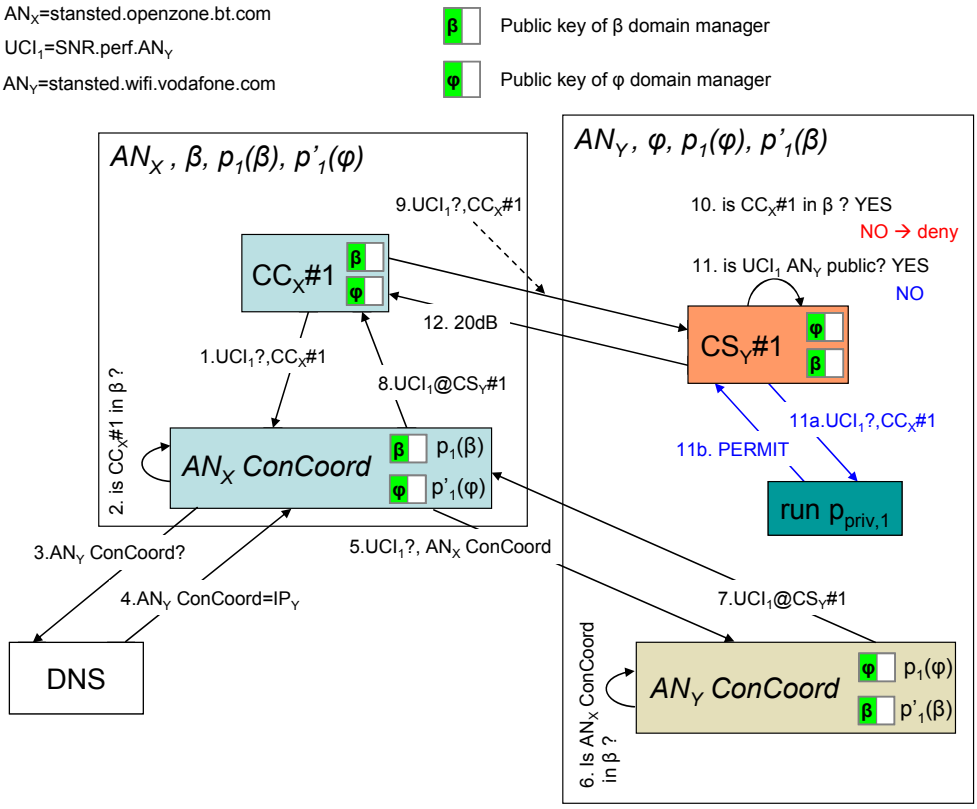


Figure 67 – Inter-AN ContextWare communication and access control

5 Conclusions and Next Steps

Some of the characteristics of Ambient Networks are the high degree of heterogeneity and dynamicity in composition, in combination with the introduction of multitude at all levels: we will have multiple different link technologies to choose from, we will have multiple different network technologies, we will have multiple different networks that will compose and decompose, we will have multiple different operators and service providers that will cooperate and compete in a dynamic fashion, we will have a multitude of different applications that will run on a multitude of different terminals with different capabilities. All this provides the user with the best connectivity possible based on the users' requirements at that time and the current capabilities of the networks available.

In order to manage networks and network nodes in such an environment, radical changes have to be made to the management paradigms of today. This will not be feasible using a centralised, hierarchical solution with a high degree of hands-on control of networks and nodes. Instead new network management systems are required. These must be distributed over the different networks, highly autonomous and self-managing, and only require interaction from operators and specialists if the system itself cannot resolve conflicts.

An integration of different management applications is one on the target of WP-D in the AN2 project.

Our conclusions are provided in the following sections and they include:

An analysis of the AN Management functionality

WP-D Results of the work in the first year of the AN Phase II project

Work plan refinement for year 2

5.1 Analysis of the AN Management Functionality

Extending the Management Functionality

Due to the limited WP-D size and work scope the full functionality AN management system is not an objective in the AN phase II project. As such we can enumerate (sub)systems that are currently missing in order to build a full management system and these systems are outside of the scope of the WP-D work.

Support for Charging

We have not yet investigated how our system can support billing and charging. In a dynamic environment, with ad-hoc compositions between possibly competing operators and providers, how will every participant be properly rewarded for their effort?

Support for Fault Management

How will be able to identify root causes for alarms being sent to and from different nodes and networks and domains? When we have a distributed system there will not be a single point, which will see all alarms. How will be able to differentiate between what should be reported as alarms and what should be reported as regular reconfigurations? Which parts can be resolved by the network(s) themselves, and what parts must be resolved by an operator? Which operator shall be contacted if there are multiple operators available? How will an operator be able to get hands-on control over (parts of) the network if she finds it necessary?

Support for Security Management

We have not yet investigated the effects that different aspects of security will have on these management systems. With an increased autonomy and inter-network/inter-domain interaction this will put completely new requirements on how nodes identify each other,

how they can trust each other, how they will authorise actions initiated in perhaps other networks, and how different instructions and requests can be saved in an unhampered way to be able to trace what actually happened, and in what order, at a later stage.

These and many other aspects must be identified, verified and incorporated into a full management system in order to provide a management system that is complete and efficient. The AN management systems should be modular, but some parts must be present in all systems, and we need to show how different modules will work together in an efficient way.

AN Network Management Interworking with ACS Functions

AN Network management systems must interact with all other parts of the ACS if it is to fulfil its work in an efficient way. This means that to enable network management system to work, these interactions must be identified, the work division between the network management system and the other parts must be resolved and agreed upon, and the solutions must be tested and verified. Some of the interactions we are presenting in section 2.8. Context management and Network management are and will be closely interleaved with each other. Context management provides the means for distributed NMSs to get access to policies and information, and context management will require interaction from NMSs to know how to update, synchronise and distribute their information within and between networks and domains.

5.1.1 Overall Properties of the AN Management Systems

One of the main drivers behind AN autonomous management systems is that the industry is finding that the cost of technology is decreasing, yet IT costs are not (i.e. *IT gap*). Also, as systems become more advanced, they tend to become more complex and increasingly difficult to maintain. To complicate matters further, there has been, and will be for the foreseeable future, a scarcity of IT professionals to install, configure, optimise and maintain these complex systems.

One other important driver behind AN autonomous management system is the increasing network operational costs and the cost of introducing new services (i.e. *Service Gap*). The rapid deployment of Information Technology infrastructure technology and hardware, and its accelerating increase in performance (each generation improves by a factor of 1.5 – 2 per annum), gives rise to new challenging problems. On the one hand, the availability of computation performance and network bandwidth at a reasonable cost stimulates demand for products with more functionality and the demand for increasingly powerful services. On the other hand, developers cannot keep up with the demand for software, resulting in an ever-increasing Service Gap: the time lag and high cost of adapting/engineering end system software, and the diversion of highly qualified staff to 'administrative' functions.

The main reason for this Service gap, and the mismatch between technological potential and its realisation as software and services, is to be found in the management separation between the computational and communication resources and the lack of focus on complexity. Complexity is currently the main factor preventing highly qualified workers from being productive, since many of them are occupied with system administration, configuration and maintenance, resulting in costs but no return of investment. At the same time, complexity limits developers' productivity since it increases project timescales, and encourages specialisation and inflexibility. It is widely agreed that the high costs of the initial project stages and its associated administrative overheads, inhibit the future growth of IT and its advance into new areas. If software and services are to progress at a faster pace than computational performance and bandwidth, which is the only way to narrow the Service gap, managing complexity becomes the key issue. In part because of the Service Gap these deficiencies remain to be addressed, and their solution using traditional technology is not getting nearer.

Therefore, the aim of AN autonomic management systems is to reduce the amount of maintenance and management needed to keep systems working as efficiently as possible, as much of the time as possible i.e. it is about making systems self-managing for a broad range of activities.

The main aim of the AN Management autonomous systems is that they exhibit self-awareness properties, in particular self-contextualisation, programmability and self-management (i.e. self – optimisation; – organisation; -policing; -configuration; -adaptation; – healing; – protection) as depicted in Figure 68.

Self-Contextualisation – Context is any information that can be used to characterise the situation of an entity (a person or object) that is considered relevant to the interaction between a user and an application. A context-aware system is capable of using context information ensuring it successfully performs its expected role, and also maximises the perceived benefits of its use. Nevertheless, this is a user-centric view and reflects the fact that most research on context and context-awareness up to now has been focused on "user context". In contrast a new generation network gives context a much broader scope and renders it universally accessible as a basic commodity provided and used by the network. In this way context becomes a decisive factor in the success of future AN autonomous rule-based systems adaptive to changing conditions. As such contextualisation is a service/software property. Self-contextualisation means that a management service/software component autonomously becomes context-aware. It represents the ability of a management system to describe, use and adapt its behaviours to its context. Network context for supporting service/software components should be made available, so that multiple service/software components may take advantage of the available network context. In order to do so in the complex environment of the large and heterogeneous Internet, the service/software component must be equipped with certain self-management capabilities. Once a management service/software component becomes context-aware it can make use of context information for other self-management tasks that depend on context information.

Programmability – Recent research on distributed systems and network technologies has focused on making service networks programmable. The objectives of programmable service networks are to take advantage of network processing resources and to promote new service models allowing new business models to be supported. The resulting service models do not, however, target development of services, but, rather, their deployment. Dynamic service programming applies to executable service code that is injected into the AN autonomic systems elements to create the new functionality at run-time. The basic idea is to enable third parties (users, operators, and service providers) to inject application-specific services into the Autonomic Systems platform. Applications and services are thus able to utilise required network support in terms of optimised network resources and as such they can be said to be network-aware i.e. a service-driven network. This means that network programmability is following autonomous flows of control triggered and moderated by network events or changes in network context The network is self-organised in the sense that it autonomically monitors available context in the network, and provides the required context and any other necessary network service support to the requested services, and self-adapt when context changes.

Self-Management – Management is an essential topic when dealing with utilisation of network context information for supporting services and contextualised services. Managing context from the perspectives of the context information provider means dealing with a number of processes related to manipulation of network context information. For instance, the creation, composition and inference of context of diverse quality; also Quality of Context (QoC)-based storage, distribution and caching are relevant. Clearly, a context source must be trustworthy and the information it provides must be sufficiently precise for

the task at hand, and, in this way, the new concept of Quality of Context (QoC) becomes important.

Among AN autonomous capabilities self-optimisation, self-organisation, self-configuration, self-adaptation are highly relevant. Moreover, the management of network context information must be addressed in the framework of the autonomous computing paradigm. This means that a key element for contextualisation is the addition of intelligence and self-management capabilities to facilitate network context self-management and thus eliminating unnecessary multi-level configurations as in conventional hierarchical management systems. With such embedded intelligence, it is only necessary to write or specify the high-level design goals and management constraints, so that the network and service overlay should make the low level decisions on its own. The system should reconfigure itself according to changes in the high-level requirements (i.e. use of cognitive and knowledge networks principles). This requires the ability to express rules within each configuration level and also between levels.

Currently, network management faces many challenges: complexity, data volume, data comprehension, changing rules, reactive monitoring, resource availability, and others. AN Self-management aims to automatically meet these challenges, in the following ways:

Self-optimisation – In the large and heterogeneous Internet, heterogeneous and distributed network management and context information and resources and their availability are rapidly changing. There is a need for an AN autonomous management tool for consistent monitoring and control of network management and context information and resources, so that service/software components may be executed or deployed in the most optimised fashion. AN Autonomic systems must seek to improve their operation every time. They must identify opportunities to make themselves more efficient from the point of view of strategic policies (performance, cost, etc).

Self-organisation – Network elements and context information and resources are distributed across heterogeneous networks. In order for services to make use of these distributed information and resources, they must be structured or referenced in an easy-to-access-and-retrieve structure in an automatic fashion. All these network context information and resources must be autonomously organised and reserved through a service layer. The autonomous structuring of network context information and resources is the essential work of self-organisation. Also from a network management point of view self-organisation must be introduced and used in order to cope with the frequent changes in the network, as well as to enable self-healing and to hide potential faults as far as possible. This is a continuous process.

Self-policing – Because of the ability of Ambient Networks to adapt to evolving situations and conditions, new network resources can become active, policies can change and the business needs and models can vary accordingly. These changes are actuated through high-level policies that need to get appropriately translated into low-level localised actions allowing achievement of overall system goals.

Self-configuration/Self-adaptation – Autonomous structuring of network management and context information and resources makes them available to services. User services and the underlying supporting services must be re-configured in order to make use of new network management and context information and resources. The new network management and context information and resources also trigger changes such as re-configuration in the network context-aware overlay. AN Autonomic systems must configure themselves in accordance with high-level policies representing service agreements or business objectives, rules and events. When a component or a service is introduced, the system will incorporate it seamlessly, and the rest of the system will adapt to its presence. In the case of components, they will register themselves and other components will be able of use it or modify their behaviour to fit the new situation.

Self-healing – AN Autonomic systems will detect, diagnose and repair problems caused by network or system failures. Using knowledge about the system configuration, a problem-diagnosis embedded intelligence would analyse the monitored information. Then, the network would use its diagnosis to identify and enforce solutions or alert a human in the case of no solutions available.

Self-protection – There are two ways in which an AN autonomic system must self-protect. It must defend itself as a whole by reacting to, or anticipating, large-scale correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures.

The realisation of self-awareness properties revolves around increasing the level of automation of the intelligent control loop described as *'Collect-Decide-Enforce'*.

- 'Collect' is about monitoring that allows construction of a picture about the surrounding environment in order to build self-awareness
- 'Decide' involves inference and planning. The former refers to a process whereby the problem is diagnosed based on the collected information while the latter refers to the process whereby a solution is selected.
- 'Enforce' comprises deployment, which adds functionality by means of new components, and configuration, which changes the existing functionality by means of programmability of networks.

The realisation of this intelligent loop eventually leads to a distributed, adaptive, global evolvable system capable of fostering continuous changes as it depicted in Figure 68.

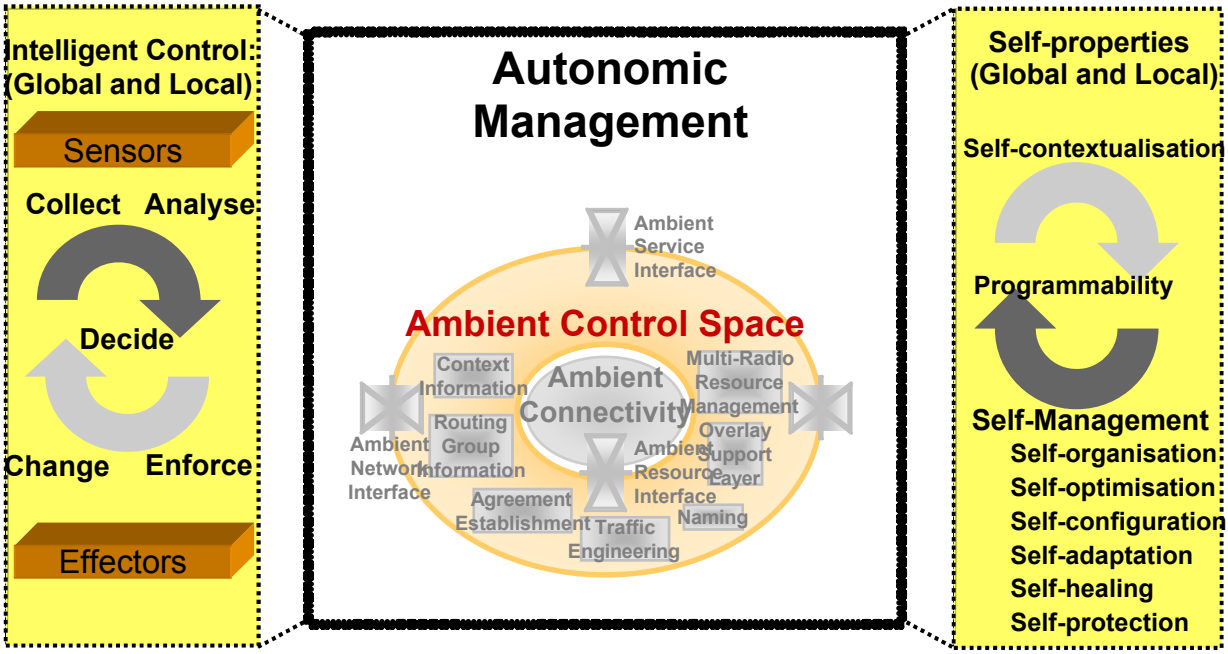


Figure 68 - AN Autonomic Management Systems

5.2 Main Results

The main WP-D results in the first year of phase 2 are:

- *AN ManagementWare model* (WP-D) - the development of this model underpins all WP-D design and implementation work. It includes all ManagementWare FEs: ConCoord FE, Context Management FE, Monitoring FE, Composition Management FE, Self-configuration FE, Registries Management FE.
- *Context Management System design and implementation (D1)* – Full design and development of a novel network-level context management system. It has the role of managing the context information in the Ambient Control Space, including its distribution to *context clients/consumers*. Context clients are context-aware services, either user-facing applications/services or network services, which make use of or/and adapt themselves to context information. Network services in the Ambient Networks project are described as the services provided by a number of *Functional Entities (FE).* The Context Management functionality helps to make the interactions between the different *context sources* and *context clients* simpler and more efficient. It acts as a mediating unit and reduces the numbers of interactions and the overhead control traffic. A model based evaluation of the performance for context distribution was developed.
- *ConCoord System design and implementation (D1*) – Full design and development of the ConCoord, which corresponds to a distributed registry that maps Universal Context Identifiers (UCIs) to the location of context information objects. The ConCoord maintains this registry by receiving REGISTER requests from context sources (entities providing one or more context objects). The ConCoord FE is the first point-of contact for context clients. When Context clients want to access context information, these clients send RESOLVE requests, which contain one or more UCIs to the ConCoord. The ConCoord then responds by returning the locations of the corresponding context objects. Universal Context Identifiers (UCIs) are a new type of Uniform Resource Identifiers (URI). UCIs uniquely identify a given context object, but not its location within the network. Then clients contact the located context sources directly to GET the information, or to SUSBSCRIBE to context change events by receiving NOTIFICATIONs. The REGISTER and RESOLVE primitives with which the ConCoord communicates with the context sources and clients is a SIMPLE-inspired protocol. The ConCoord is implemented using Bamboo DHTs.
- *Context-aware Service Management design (D1)* – In cooperation with WP-F the network underlay ContextWare work has being extended and integrated with the service overlay (SATOs). By overlay context-awareness, we refer to the capability of a SATO to use network context information for self-adaptation, or in the provision of services. The provision of network context-awareness is needed in SATO in order to aid in distributed service management in heterogeneous wireless environments, in response to the ever-changing network context. As a summary, the context-aware functionality of SATO service management includes: i. support efficient deployment, activation, and (re)configuration of (new) Context Sensors in a scalable and distributed manner on (potentially) large numbers of AN nodes; in order to monitor and collect specific pieces of network context that are needed to support service management; ii. Provision of a scalable, distributed mechanism for locating (the collected) distributed network context within an AN; in order to support subsequent network context retrieval by AN service managers for service management.

- *Access Control and Authorisation in ContextWare (D1)* – Security domains will be created during AN bootstrapping and (re-)configuration phases, and nodes (or resources) will be enrolled into these domains in accordance with their security policies. The implication of security domain memberships of CW nodes on the mechanisms for accessing/retrieving context information from context sources (or managers) has been investigated. Our approach is to perform coarse-grained access control (at ConCoord) through security domain management and to run further (private) access policies at the context sources, if necessary.
- *Adaptive Decentralized Real time Monitoring: design, evaluation and implementation (D2)*- The *decentralized real time monitoring FE* provides other FEs and network administrators with network-wide state variables in real time. The client FE (or network administrator) specifies: (i) the network-wide variable it is interested in (e.g., the number of VoIP flows in the domain), (ii) the monitoring technique (e.g., periodic sampling or continuous) and (iii) the required accuracy. The monitoring FE provides the client with an estimation of the variable with the required accuracy and minimal overhead. (Monitoring distributed systems involves the fundamental trade-off between accurate estimation of a variable and the generated overhead). The monitoring FE creates a self-organizing layer that interconnects management processes on the network devices. This self-organizing layer is scalable, robust and adaptive to networking condition changes. The monitoring FE includes a number of protocols we have developed (e.g., Echo, GAP, A-GAP, TCA-GAP). Echo provides distributed polling. A-GAP provides continuous monitoring and can reduce the generated overhead by almost two orders of magnitude for allowing small accuracy errors. A prototype has been developed and is currently under evaluation. It will be integrated with the ACS framework in 2007.
- *Self-configuring management design (D2)* – Self-management is the dominant approach in Ambient Networks and different configuration tasks are performed independently by different FEs without the control of a central component. On the other hand, decentralized configuration requires in general some mechanisms to accomplish correctly their configuration tasks. Stability and oscillations avoidance are the major issues that must be tackled in a distributed scenario. An architecture for plug-and-play base stations has been designed and evaluations have been conducted to investigate the behaviour of some of the control algorithm underneath. So far, stability has been enforced with a simple mechanism that blocks contradictory actions. With this experience, a specific FE has been proposed to support more advanced services to guarantee stability of distributed configuration tasks. This new self-configuring FE basically relies on instruments of control theory to mitigate the enforcement of configuration actions over time.
- *Management of Composition – design and implementation (D2/D4)* – A set of P2P hierarchical self-organization models have been developed along with a formal description and evaluation methodology to analyse maintenance of a logical network structure on top of a dynamically changing physical network topology. The Composition Management FE has been designed based on experience with the above self-organization models. It cooperates with the ACS framework, the Composition FE, and the Policy Management System to manage the logical structure of the ACS as well as FE instances in this structure. A first prototype was developed and is currently under integration with the ACS framework.
- *Policy Framework design (D3)*- Policies are used in a number of different areas within AN. Still all FEs in the ACS need to use the common AN policy framework to ensure consistency between the different policy sets. When composing there is a need to check that the policies remain consistent also after the composition. The

policy framework developed by D3 was used for self-management and security domains in AN.

A number of management interoperability and integration techniques and enables were investigated and developed including:

- *Registry Management Design* (D2/D4) – The goal of a common distributed registry management FE is to exploit synergies and provide a unified distributed registry functionality for all functional entities. As such the registry management enable interworking and integration between all ACS FEs and in particular the management FEs. While scalability and robustness is achieved through the use P2P technology, fault tolerance and low maintenance overhead is ensured by bypassing the storage subsystem whenever it is possible. The design of this novel registry is based on DHTs, which are applicable to AN (i.e. they follow (de)composition of the ANs and efficient stochastic routing maintenance mechanisms guarantee low maintenance overhead in dynamic AN environments).
- *Dynamic contextualisation and adaptation; instantiation of management functionality* (D4) – a modular and scalable system facilities, called DINA, that enables the dynamic deployment, control and management of functionality (i.e. programmable sessions over network entities) was redesigned for AN project, including porting to FreeBSD. These facilities are aimed at use in year 2 work for on-demand dynamic instantiation and activation of new execution environments, management functionality, new network sensors, or for dynamic adaptation and self- contextualisation. DINA was used for the ContextWare prototype of WP-DF and it also used in the WP-F work.
- *Cross-layers Management approaches (D4)*- Cross-layer design enhances the traditional design, where each layer of the protocol stack operates independently by facilitating information exchange between layers and by optimising end-to-end performance of different problems. We are assuming that there is certain cooperation possible between AN underlay and Service overlay layers. Information and /or policies at the overlay could be provided at underlay and vice versa in order to better solve an overlay/underlay problem, including cross-layers optimisation. Joint work WP-F/ WP-E/WP-D has focussed on identifying the main principles for cross-layers design and their used in solving different types of cross-layers interactions or optimisation problems, including: i. Interactions between AN bearers; ii. Underlay transport protocol performance optimisation; iii. Re-routing in the overlay based on congestions status in the underlay; iv. Underlay/overlay QoS interactions; v. Collecting and storing distributed network context information, to support service management in SATOs; VI. vii. Underlay/overlay policies' interactions. Cross-layer approaches are one of the most promising design models to serve as a blueprint for dynamic network architectures.
- *ASI Management Primitives Design (D4).* ASI framework includes the specification and design of the ASI primitives ASI primitives, related to particular FE: ContextWare, Network Management, Policy Management have being developed in cooperation with WP-F.
- *Self-organisation Management Design (D4)* – a self-organising management system is needed to automatically and dynamically determine (or to select) a set of nodes in the underlay and overlay, to be responsible for different context and management aggregation processes. Existing approaches rely on real-time negotiation(s) for node selection. However; real-time negotiations are expensive for bandwidth-limited environments such as ANs. A novel AN Virtual Management Backbones (VMBs) was developed. Each VMB has a dedicated (distributed) context aggregation task, and consist of a set of dynamically selected VMB nodes,

where aggregation is carried out. The selection of VMB nodes is conducted without heavy-weighted real-time negotiations. Our solution achieves this goal by utilising the scalability and dynamicity advantages of Distributed Hash Tales (DHTs).

- *Dynamic ACS Orchestration Engines (D4)* – ACS orchestration describes: i. required patterns of interaction between FEs, and templates for sequences of interactions (i.e. orchestration scripts /policies); ii. control and data-flow using processing steps with activities, sequencing rules/policies, flows with data and/or control links; iii. execution of the order of the interactions between FEs. iv. execution of task-driven orchestrations (e.g. management tasks, mobility tasks, security tasks, connectivity tasks, etc.). Dynamic Orchestration Engine is an intelligent intermediary between ACS FEs that alters the flow of orchestration based on run time information and it executes task-driven orchestration scripts. ACS may require multiple types of orchestration engines, which are task -optimised. We envisage orchestration engines, which are management-task optimised. The management tasks include: initialisation, dynamic reconfiguration, adaptation and contextualisation, FCAPS functionality, dynamic service deployment support, optimisation, organisation of ACF FEs. The management orchestration is for further study in year 2.

The interaction with other work packages (WPs) has been driven by a number of related topics being addressed separately in different WPs. Links with some WPs have been stronger than with others and more details about it can be found in the Common milestone report WP-D/E/F on cross layer design, interactions, optimisations and enhancements. Worth mentioning here the collaboration with WP-B leading to a number of publications and to a standard contribution on a Network Information Service Infrastructure, as well as the collaboration with WP-F on making the SATOs context-aware. By overlay context-awareness, we refer to the capability of a SATO to use network context information for self-adaptation, or in the provision of services.

Interactions with WP-A and contribution to the AN System Description (SD) report were performed for all quarterly SD releases.

Interactions with WP-H for the AN prototype requirements and S/W licence were made. The following first prototypes of the ContexWare & ConCoord systems and the Management of Composition have been built and integrated into the common Ambient Network prototype. In addition DINA platform (i.e. dynamic management adaptation and dynamic deployment, control and management of functionality) and XCAM Policy Wrapper have beign developed for FreeBSD and made available for WPs use.

Collaboration with the other WWI projects SPICE, E2R2, WINNER, MOBILLIFE on the integration of the main context-aware components part of an overall WWI System architecture resulted in joint paper presented at the World Wireless Research Forum in Nov 2006

5.3 Next Steps

For the 2nd year of Ambient Networks Phase 2 there is no change of work objectives as defined in the AN Technical Annex 1.

The main goal is to progress and complete the design work for all ManagementWare FEs (see section 1.2) and to implement the results in a 2nd version of the prototype, which would include all Management FEs.

In addition work towards finding specific solutions for the following open issues is envisaged:

- (D1) develop the concept of Quality of Context (QoC) and its integration in the ContextWare.

- (D1) optimise and evaluate the scalability of the network-based ContextWare.
- (D1) develop interworking scenarios between network-based and service-based ContextWare systems.
- (D2) elaborate and evaluate the management techniques in the context of dynamic networks with volatile topology, in particular the stability and integrity procedures for configuration FE.
- (D2) extend the functionality of the Monitoring FE to support (i) other aggregation functions, such as “top-k” queries, and (ii) other control parameters, such as confidence intervals, whereby the monitoring error must not exceed a given maximum error with a given probability.
- (D2) evaluation of centralized vs decentralized approaches for the Monitoring FE.
- (D2) integrate network sensors developed Y1 into the monitoring and configuration process.
- (D2) design and evaluate generic stability algorithms for self-configuring tasks.
- (D3) elaborate scenarios for coordinating policy decisions as applied to AN (i.e. orchestration of the policy decision points) in order to maintain high level of integrity and stability of the AN.
- (D3) extend the policy framework in order to include orchestration engines
- (D2/D4) further elaborate the specification and design, initial implementation implement of the Registry Management FE as a distributed repository of information and operands with emphasis on (de) composition.
- (D4) Updates and optimisation of the AN Management model.
- (Cross and common D1, 2, 3, 4 work): elaborate and develop:
    - Dynamic instantiation and activation of management functionality
    - Dynamic adaptation of management functionality to changes in the network topology (i.e. due to continuously (de) composing AN), network mobility or network context
    - Self-optimisation in each FE
    - Consolidate the Self-organisation approaches developed in Y1
    - Orchestration of Management FEs within the ACS including the dynamic reconfiguration of FEs.
    - Relationship between AN Connectivity abstractions and AN ManagementWare

WP-D plans also to contribute its consolidated results to the WP-A AN System Description and of an AN System Primer document.

Building on the Y1 activity (i.e. 26 paper submissions) WP-D group would contribute to the dissemination of project results by actively and continuously submitting collaborative research papers to relevant conferences and journals. Prime topics for disseminations would results related to the above work topics.

The IEEE 802.21 standardization on dynamic information services infrastructure work commenced in 2006 will be pursued further.

Abbreviations

3GPP	Third Generation Partnership Project
ACS	Ambient Control Space
AN	Ambient Network
A-GAP	Accuracy Generic Aggregation Protocol
ANI	Ambient Network Interface
ANN	Ambient Network Node
AP	Access Point
API	Application Programming Interface
ARI	Ambient Resource Interface
ASI	Ambient Service Interface
AVP	Ambient Virtual Pipe / Ambient Virtual Network
CA	Composition Agreement
CAN	Content-Addressable Network
CLA	Context Level Agreement
CM	Context Management
CMIP	Common Management Interface Protocol
ConCord	Context Coordination
CSC	Context Sensitive Communication
CIB	Control Information Base
DEEP	Destination Endpoint Exploration Protocol
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
DINA	DINA programmable network platform
DN	Distinguished Name
DNS	Domain Name System
FE	Functional Entity
FCAPS	Fault-management, Configuration, Accounting, Performance, and Security
GANS	Generic Ambient Network Signalling
HIT	Host Identity Tag
IETF	Internet Engineering Task Force
IP	Internet Protocol
LDAP	Lightweight DAP
L1, L2, L3	OSI Layer 1, 2, 3
LAN	Local Area Network

MIB	Management Information Base
OCI	Overlay Control Interface
OM FE	Overlay Management Functional Entity
ON	Overlay Network
ONode	Overlay Node
OSL	Overlay Support Layer
P2P	Peer-to-Peer
PAN	Personal Area Network
PDA	Personal Digital Assistant
OSI	Open Systems Interconnection
OWL	Ontology Web Language
QoS	Quality of Service
QoC	Quality of Context
SIMPSON	Simple Pattern Simulator for Large Networks
SLA	Service Level Agreement
SATO	Service Adaptive Transport Overlay
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
UCI	Universal Context Identifier
UDDI	Universal Description Discovery and Integration
URI	Universal Resource Identifier
UML	Unified Modelling Language
TCP	Transport Control Protocol
TMN	Telecommunications Management Network
TTL	Time-To-Live
VPAN	Virtual Private Ambient Network
VPN	Virtual Private Network
WEP	Wired Equivalent Privacy
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network
WP	Work Package
WSDL	Web Services Description Language
W3C	World Wide Web Consortium
XDSL	Digital Subscriber Loop (e.g. ADSL)

References

[1] Ambient Networks Phase 2 Annex I – Description of Work, 2005-10-17

[2] M. Dam, R. Stadler, “A Generic Protocol for Network State Aggregation”, Radiovetenskap och Kommunication (RVK), Linkoping, Sweden, 14-16 June 2005.

[3] A. Gonzalez Prieto and R. Stadler, “Distributed Real-time Monitoring with Accuracy Objectives”, KTH Technical Report, May 2006. Available at: http://www.ee.kth.se/~gonzalez

[4] A. Gonzalez Prieto, R.Stadler "Adaptive Distributed Monitoring with Accuracy Objectives", to appear in the Proceedings of ACM SIGCOMM workshop on Internet Network Management (INM 06), Pisa, Italy, September 11, 2006

[5] A. Gonzalez Prieto, R.Stadler "Distributed Real-Time Monitoring with Accuracy Objectives", in the Proceedings of IFIP Networking, Coimbra, Portugal, May 15-19, 2006

[6] K. Lim and R. Stadler. SIMPSON — a SIMple Pattern Simulator fOr Networks. http://www.comet.columbia.edu/adm/software.htm, 2006.

[7] N. Spring, R. Mahajan, and D. Wetherall, “Measuring ISP topologies with Rocketfuel”, ACM/SIGCOMM, 2002, Pittsburgh, USA, August 2002.

[8] C. Olston, B. T. Loo and J. Widom, “Adaptive Precision Setting for Cached Approximate Values”, ACM SIGMOD 2001, Santa Barbara, USA, May 2001.

[9] C. Olston and J. Widom, ”Efficient Monitoring and Querying of Distributed, Dynamic Data via Approximate Replication”, IEEE Data Engineering Bulletin, March 2005

[10] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos, “Hierarchical In-Network Data Aggregation with Quality Guarantees”, EDBT 2004, Crete, Greece, March 2004

[11] A. Gonzalez Prieto and R. Stadler, “Distributed Real-time Monitoring with Accuracy Objectives”, KTH Technical Report, May 2006. Available at: http://www.ee.kth.se/~gonzalez

[12] R. van de Meent, A. Pras, Traffic Measurement Data Repository, University of Twente, http://traffic-repository.ewi.utwente.nl/, May 2006

[13] G. Cormode et al., “Holistic aggregates in a networked world: distributed tracking of approximate quantiles”, In ACM SIGMOD International Conference on Management of Data, Baltimore, USA, June 2005.

[14] M. A. Sharaf et al, “Balancing energy efficiency and quality of aggregate data in sensor networks”, ACM In-ternational Journal on Very Large Data Bases, 13(4):384–403, December 2004

[15] A. Boulis, S. Ganeriwal, and M. B. Srivastava, “Aggregation in sensor networks: an energy – accuracy tradeoff”, Elsevier Ad-hoc Networks Journal (special issue on sensor network protocols and applications), pages 317–331, 2003.

[16] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, “Impact of network density on data aggregation in wireless sensor networks”, 22nd International Conference on Distributed Computing Systems, Vienna, Austria, July 2002.

[17] IEEE. ANSI/IEEE Std 802.1D, 1998 Edition. IEEE, 1998.

[18] H. Pucha, S. Das, Y. Hu, “Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks”, 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA) 2004.

[19] T. Heer, S. Gotz, S. Rieche, K. Wehrle, “Adapting Distributed Hash Tables for Mobile Ad Hoc Networks”, 4th IEEE International Conference on Pervasive Computing and Communications Workshop (PERCOMW) 2006, pp. 173-178.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, “A Scalable Content-Addressable Network”, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM) 2001, pp. 161-172.

[21] R. Ocampo, L. Cheng, K. Jean, A. Prieto, A. Galis, Z. Lai, “Towards a Context Monitoring System for Ambient Networks”, in submission to the Chinacom (2006), temporarily available at http://www.ee.ucl.ac.uk/~lcheng/Sections/CHINACOM_2006.pdf

[22] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", IFIP/ACM Middleware, Heidelberg, Germany, pages 329-350, November, 2001, http://research.microsoft.com/~antr/PAST/pastry.pdf

[23] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", ACM SIGCOMM 2001, San Diego, August 2001, http://pdos.csail.mit.edu/sections/chord:sigcomm01/

[24] R. Campos, C. Pinho, M. Ricardo, J. Ruela, P. Poyhonen, C. Kappler, “Dynamic and Automatic Interworking between Personal Area Networks using Composition”, the 16th IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC), 2005.

[25] A. Galis, L. Cheng, M. Brunner, S. Schuetz, G. Nunzi, E. Asmare, J. Nielsen, A. Gunnar, H. Abrahamsson, R. Szabo, S. Csaba, M. Erdei, P. Kersch, Z. Kis, B. Kovács, R. Stadler, A.Wágner, K. Molnár, A. Gonzalez, G. Molnar, J. Andres, and M. Callejo. Ambient Network Management – Technologies and Strategies. Deliverable *Report D-8-1, Ambient Networks Project*, December 2004. Document number IST-2002-507134-AN/D8-1, available at http://www.ambient-networks.org/publications/D8-1_PU.pdf

[26] R. Giaffreda, J. Dang, R. Glitho, M. E. Barachi, F. Belqasmi, J. Mattam, T. Kanter, C. Reichert, M. Smirnov, A. Karmouch, D. Balakrishnan, H. Harroud, A. Karlsson, H. Laamanen, M. Laukkanen, R. Ocampo, K. Jean and A Galis. Ambient Networks ContextWare: Second Section on Context-Aware Networks. *Deliverable Report D-6-3, Ambient Networks Project,* December 2005. Document number IST-2002-507134-AN/WP6/D63, at http://www.ambient-networks.org/publications/D6_3_Ambient_Networks_ContextWare_Second_Section_on_Context-Aware_Networks_PU.pdf

[27] R. Ocampo, L. Cheng, Z. Lai and A. Galis. ContextWare Support for Network and Service Composition and Self-Adaptation. *2nd International Workshop on Mobility Aware Technologies and Applications (MATA 2005)*, Montreal, Canada, October 2005.

[28] R. Ocampo, A. Galis, H. De Meer and C. Todd. Flow Context Tags: Concepts and Applications. *IFIP TC6 Conference on Network Control and Engineering for QoS, Security and Mobility (NetCon'05)*, Lannion, France, November 2005

[29] R. Ocampo, A. Galis and C. Todd. Triggering Network Services Through Context-Tagged Flows. *Proceedings of the 5th International Conference on Computational Science (ICCS'05),* Atlanta, Georgia, USA, May 2005.

[30] R. Ocampo, A. Galis, H. De Meer and C. Todd. Implicit Flow QoS Signaling Using Semantic-Rich Context Tags. *Proceedings of the 13th International Workshop on Quality of Service (IWQoS 2005)*, Passau, Germany, June 2005.

[31] R. Ocampo, A. Galis, H. De Meer and C. Todd. Supporting Mobility Adaptation Through Flow Context. *2nd International Workshop on Mobility Aware Technologies and Applications (MATA 2005)*, Montreal, Canada, October 2005.

[32] R. Ocampo, A. Galis, H. De Meer and C. Todd. Towards Context-Based Flow Classification. To appear, *International Conference on Autonomous and Autonomic Systems (ICAS'06)*, July 2006, San Jose, California, USA.

[33] Peter Kersch: *Formalism to describe hierarchical self-organization algorithms*, https://bscw.ambient-networks.org/bscw/bscw.cgi/361421

[34] G. Karypis, E. H. Han, V. Kumar: CHAMELEON: *A Hierarchical Clustering Algorithm Using Dynamic Modelling*, IEEE Computer: Special Issue on Data Analysis and Mining

[35] Zoltán Lajos Kis, László Harri Németh: *Promise-based Autonomous regulation in Ambient Networks,* https://bscw.ambient-networks.org/bscw/bscw.cgi/361421

[36] Mark Burgess: An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation, Lecture Notes on Computer Science, 2005

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

[38] P. Mockapetris, P., "Domain Names – Concepts and Facilities," RFC 1034, IETF Nov. 1987.

[39] D. Ratajczak and J. Hellerstein, "Deconstructing DHTs," Intel Research Berkley, IRB-TR-03-042, Intel Research, Nov, 2003.

[40] Bamboo DHT website, http://bamboo-dht.org.

[41] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," Lecture Notes in Computer Science, vol. 2218, pp. 329–350, Springer 2001.

[42] I. Stoica et al, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications," ACM SIGCOMM, San Diego, CA, USA, Aug. 2001.

[43] L. Cheng, K. Jean, R. Ocampo, A. Galis, "Service-aware Overlay Adaptation in Ambient Networks", International Multi-Conference on Computing in the Global Information Technology (ICCGI) 2006, http://www.ee.ucl.ac.uk/~lcheng/Sections/ICCGI_2006_01.pdf

[44] R. Giaffreda, H. Tschofenig, T. Kanther, and C. Reichert, "An Authorization and Privacy Framework for Context-aware Networks," MATA 05, Oct. 2005.

[45] A. Méhes and G. Selander, "Identifiers and keys for management and composition of Ambient Networks", Annex 3 of Deliverable Report D7.2 "Ambient Network Security Architecture", Ambient Networks Project, December 2005, Document number IST-2002-507134-AN/WP7/D02, available at: http://www.ambient-networks.org/publications/D%207_2_ANNEX_3_PU.pdf.

[46] R. Moskowitz et al.: Host Identity Protocol – Internet Draft, IETF June 2006, available at: http://www.ietf.org/internet-drafts/draft-ietf-hip-base-06.txt.

[47] http://jena.sourceforge.net/, Jena – A Semantic Web Framework for Java Home Page

[48] http://protege.stanford.edu/, Protégé ontology editor and knowledge-base framework Home Page.

[49] J.de Bruijn, M. Ehrig, C. Feier, Ontology mediation, merging and aligning, May 2006

[50] http://www.w3.org/TR/owl-ref/ OWL Web Ontology Language Reference

[51] G.N. Stone,B. Lundy, G.N. Xie,"Network policy languages: a survey and a new approach", IEEE Network, Jan-Feb 2001, V.15,N. 1, pp 10-21.

[52] "Specification, design and implementation of the necessary components for the enhancement of an active platform for the validation of the project approach", CONTEXT-WP4-UPC-D4.2-101204 Del, IST – 2001 – 38142 – CONTEXT, 10.12.2004

[53] Konstantinos Psounis, "Active networks: applications, security, safety, and architectures", IEEE Communications Survey. First Quarter 1999, Vol. 2, No. 1.

[54] Jonathan M.Smith, Scott M.Nettles, "Active Networking: One view of the past, present and future", IEEE Transactions on Systems, Man, and Cybernetics, Part C, Volume 34, Issue 1 Page(s):4 – 18, Feb. 2004

[55] S. Vrontis, I. Sygkouna, M. Chantzara and E. Sykas, "Enabling distributed QoS management utilizing active network technology", 2003 IFIP-IEEE International Conference on Network Control and Engineering (Net-Con 2003), 11-15.10, 2003, Muscat, Oman.

[56] Kai Zimmermann, Lars Eggert, Simon Schütz and Marcus Brunner, "Self-Management of Wireless Base Stations", 15th IST Mobile & Wireless Communications Summit, Myconos, Greece, June 4-8, 2006.

[57] Y. Bejerano, S. Han, "Cell Breathing Techniques for Load Balancing in Wireless LANs", IEEE INFOCOM 2006.

[58] G. Nunzi, S. Schuetz, M. Brunner, "Design and Evaluation of Distributed Selfconfiguring Load-Balancing", IFIP/IEEE IM 2007.

[59] D 1.5 – AN Framework Architecture Annex / www.ambient-networks.org

[60] J. M. Kleinberg, "The Small-World Phenomenon: an Algorithmic Perspective", In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000

[61] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer networks," in Proc. of ACM PODC, Jul. 2002

[62] WP-D Publications/ Papers accepted for Publications to date

[63] Börje Ohlman (EAB), Kerry Jean (UCL), Alex Galis (UCL), Ian Herwono (BT), Johan Nielsen (EAB) -"Requirements for Policy Framework for Ambient Networks", WWRF16, 26-28 April 2006, Shanghai, China

[64] Gonzalez Prieto (KTH), R. Stadler (KTH)-"Distributed Real-time Monitoring with Accuracy Objectives" -, IFIP Networking 2006, Coimbra, Portugal, May 2006

[65] Simon, C. (BME), Wagner, A. (BME), Kersch, P. (BME), Erdei, M. (BME), Katona, T. (BME), Benk, B. (BME), Szabó, R. (BME), Galis, A. (UCL), Cheng, L. (UCL), Jean, K. (UCL), Lai, Z. (UCL), -"Peer-to-Peer Management in Ambient Networks" poster and demonstration at IST Mobile and Wireless Communications Summit, Myconos, 4-8 June 2006,

[66] Kai Zimmermann (NEC), Lars Eggert (NEC), Simon Schütz (NEC), Marcus Brunner (NEC). – "Self-Management of Wireless Base Stations"- 15th IST Mobile & Wireless Communications Summit, Myconos, Greece, June 4-8, 2006.

[67] Csaba Simon (BME), Petteri Pöyhönen (Nokia), Martin Johnson (Ericsson), Di Zhou (Siemens) -"Controlling and Managing Compositions in Ambient Networks" -, poster at 15th IST Mobile & Wireless Communications Summit, Myconos, Greece, June 4-8, 2006.

[68] Johan Nielsen (Ericsson), Zoltan Lajos Kis (BME), Marcus Brunner (NEC), Alberto Gonzalez (KTH), Rolf Stadler (KTH), -"Pattern-based Network Management within Ambient Networks", poster at IST Mobile Summit 2006, Myconos, Greece, June 4-8, 2006.

[69] Theo Kanter (Ericsson EAB)- "The ACAS and Ambient Network Approaches to Context-Aware Networks"- Second Workshop on Context Awareness for Proactive Systems (CAPS 2006), Kassel, Germany,12-13 June 2006.

[70] Roel Ocampo (UCL), Alex Galis (UCL), Chris Todd (UCL), Hermann De Meer (PU) – "Towards Context-Based Flow Classification"- – International Conference on Autonomic and Autonomous Systems (IEEE ICAS'06), Silicon Valley, USA, July 19-21, 2006.

[71] Kerry Jean (UCL), Lawrence Cheng (UCL), Roel Ocampo (UCL), Alex Galis (UCL) – "Contextualisation of Management Overlays in Ambient Networks" – IEEE International Multi-Conference on Computing in the Global Information Technology, ICCGI'06, Bucharest, Romania, August 1-3, 2006.

[72] Gonzalez Prieto (KTH), R. Stadler (KTH),- "Adaptive Distributed Monitoring with Accuracy Objectives" ACM SIGCOMM workshop on Internet Network Management (INM 06), Pisa, Italy, September 11, 2006

[73] Lawrence Cheng (UCL), Kerry Jean (UCL), Roel Ocampo (UCL), Alex Galis (UCL) - "Towards Flexible Service-aware Adaptation Policy Management in Ambient Networks" – IEEE International Conference on Networks 2006, Singapore, Sept 13-15, 2006

[74] A. Gonzalez Prieto, R. Stadler, "Adaptive Continuous Monitoring in Large-scale Networks with Accuracy Objectives", Fourth Swedish National Computer Networking Workshop (SNCNW 2006), Luleå, Sweden, October 26-27 2006

[75] Villy Bæk Iversen: "Teletraffic Engineering Handbook" ITU-T Study Group D. Geneva, 2005

[76] J. Rosenberg, "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856 Internet Engineering Task Force, August 2004. URL: http://www.ietf.org/rfc/rfc3856.txt

[77] Galis, A., Denazis, S., Brou, C., Klein, C. (ed) –"Programmable Networks for IP Service Deployment" ISBN 1-58053-745-6; pp450, June 2004; Artech House Books; www.artechhouse.com/Default.asp?Frame=Book.asp&Book=1-58053-745-6

[78] The Ambient Networks (ANs) Project, *http://www.ambient-networks.org*

[79] N. Niebert, et al, "Ambient networks: An architecture for communication networks beyond 3G," IEEE Wireless Communications, vol. 11, pp. 14-22, IEEE, Apr. 2004.

[80] R. Ocampo, L. Cheng, K. Jean, A. Prieto, A. Galis, "Towards a Context Monitoring System for Ambient Networks", to appear in the Proceedings of the 1st International Conference on Communications and Networking in China (Chinacom), Beijing,

China, Oct 2006:
http://www.ee.ucl.ac.uk/~lcheng/Chapters/Published/2006/CHINACOM_2006.pdf

[81] K. Lim, R. Stadler, “Read-time Views of Network Traffic using Decentralised Management”, in Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management, Nice, France, May 2005, pp. 119-132.

[82] R. Renesse, K. Birman, W. Vogels, “Astrolabe: A robust and scalable technology for distributed System Monitoring, Management, and Data Mining”, in Proceedings of ACM Trans. On Comp. Syst., Vol. 21, no. 2, May 2003.

[83] E. Chang, “Echo Algorithms: Depth Parallel Operations on General Graphs”, in Proceedings of IEEE Trans. On Softw. Engr., Vol. 8, no. 4, July 1982, pp. 391-401.

[84] S. Zhou, G. Ganger, R. Steenkiste, “Balancing Locality and Randomness in DHTs“, Technical Report, CMU-CS-03-203, Nov 2003, http://www.pdl.cmu.edu/PDL-FTP/stray/CMU-CS-03-203.pdf

[85] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, “A Scalable Content-Addressable Network”, in Proceedings of the 2001 ACM conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), San Diego, CA, USA, Aug 2001, pp. 161-172.

[86] M. Kleis, E. Lua, X. Zhou, “Hierarchical Peer-to-Peer Networks using Lightweight SuperPeer Topologies“, in Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC), Cartagena, Spain, Jun 2005, pp. 143-148.

[87] G. Jesi, A. Montresor, O. Babaoglu, “Proximity-Aware SuperPeer Overlay Topologies”, in Proceedings of the 2nd IEEE International Workshop on Self-Managed Networks, Systems & Services (SelfMan), Dublin, Ireland, Jun 2006.

[88] A. Mizrak, Y. Cheung, V. Kumar, S. Savage, “Structured SuperPeers: Leveraging Heterogeneity to Provide Constant-Time Lookup”, in Proceedings of the IEEE Workshop on Internet Applications (WIAPP), San Josc, USA, Jun 2003.

[89] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", in Proceedings of IFIP/ACM Middleware, Heidelberg, Germany, Nov 2001, pp. 329-350.

[90] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", in Proceedings of the 2001 ACM conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), San Diego, USA, Aug 2001.

[91] H. Pucha, S. Das, Y. Hu, “Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks”, in Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), English Lake District, UK, Dec 2004.

[92] T. Zahn, J. Schiller, “MADPastry: A DHT Substrate for Practicably Sized MANETs”, in Proceedings of the 5th Workshop on Applications and Services in Wireless Networks (ASWN), Paris, France, Jun 2005.

[93] K. Lim, R. Stadler, “A Navigation Pattern for Scalable Internet Management”, in Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM), Seattle, Washington, USA, May 2001.

[94] M. Adler, R. Kumar, K. Ross, D. Rubenstein, T. Suel, D. Yao, “Optimal Peer Selection for P2P Downloading and Streaming“, in Procceesings of IEEE Infocom, Miami, FL, March 2005.T. Zahn, J. Schiller, “MADPastry: A DHT Substrate for

Practicably Sized MANETs", in Proceedings of the 5th Workshop on Applications and Services in Wireless Networks (ASWN), Paris, France, Jun 2005.

[95] L. Cheng, R. Ocampo, K. Jean, A. Galis, C. Simon, R. Szabo, P. Kersch, R. Giaffreda, "Towards Distributed Hash Tables (De)Composition in Ambient Networks", to appear in the Proceedings of the 17th IFIP/IEEE Distributed Systems: Operations and Management (DSOM), Dublin, Ireland, Oct 2006.

[96] Mobile Ad-hoc NETworks (MANETs), http://www.ietf.org/html.charters/manet-charter.html

[97] OASIS eXtensible Access Control Markup Language (XACML) TC http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[98] W. Kellerer, R. Schollmeier, K. Wehrle, "Peer-to-Peer Systems and Applications", chapter Peer-to-Peer in Mobile Environments, pp. 401-417, LNCS 3485, Springer, Sep 2005.

[99] F. Dabek, M. Kaashoek, D. Karger, R. Morris, I. Stoica, "Wide-area cooperative storage with CFS", in Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Chateau Lake Louise, Banff, Canada, Oct 2001.

[100] A. Rowstron, P. Durschel, "PAST: A lagre-scale, persisent peer-to-perr storage utility", in Proceedings of the 18th ACM Symposium on Operating Systems Principles, May 2001.

[101] A. Rowstron, A. Kermarrec, M. Castro, P. Durschel, "SCRIBE: The design of a large-scale event notification infrastructure", in Proceedings of Networked Group Communications (NGC), Nov 2001, pp. 30-43.

[102] R. Winter, T. Zahn, J. Schiller, "Random Landmarking in Mobile, Topoloy-Aware Peer-to-Peer Networks", in Proceedings of the 10th International Workshop on Futuer Trends in Distributed Computing Systems (FTDCS), 2004.

[103] T. Heer, S. gotz, S. Rieche, K.Wehrle, "Adapting Distributed Hash Tables for Mobile Ad Hoc Networks", in Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communicatinos Workshops (PERCOM), Pisa, Italy, March 2006.

[104] A. Jonsson, et al., "Ambient Networks ContextWare: First Chapter on Context-Aware Networks", Deliverable Report D-6-1, Ambient Networks Project, January 2005. Document number IST-2002-507134-AN/WP6/D61, available at: http://www.ambient-networks.org/publications/D6-1_PU.pdf

[105] PONDER Policy Language, http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml

[106] Web Services Policy Framework (WS-Policy) http://msdn.microsoft.com/ws/2004/09/policy/

[107] University of Murcia (UMU), Spain:Java-based XACML editor, http://xacml.dif.um.es/

A1

A2 Appendices

A3 Models for Self-organization

Network self-organization is an important network management task in dynamic environments such as ad hoc or ambient networks. We define network self-organization as the process of creating and maintaining a logical network structure on top of a dynamically changing physical network topology. This logical network structure can be used as a scalable infrastructure by various functional entities like address management, routing, service registry, media delivery, etc.

A formal method to describe static and dynamic behaviour of hierarchical self-organization models has been described in [33], but it doesn't specify the actual algorithm and the logics behind. This section presents three different self-organization algorithms that could be used to help managing dynamic networks. All of them are described using the same formalism specified in [33] but are driven by different logics. The first is an algorithm is based on grouping nodes with the largest common subset of properties, the second one groups nodes with most compatible policies while the third one is based on cooperation of autonomous systems using promise theory. The first and second algorithms are quite similar therefore they are presented together under the same sub-section A3.1. The use of the promise theory is instead discussed in sub-section A3.2.

A3.1 Property set and policy based self-organization models

The property set based model tries to create a hierarchy by grouping together nodes with most possible common properties. Each physical node has a set of properties while property set of upper level tree nodes in the hierarchy is created as the intersection of property sets of their child nodes (thus consists of properties common to all children nodes).

A3.1.1 Property sets

The abstract term property denotes an arbitrary key-value pair. A property can represent multicast membership, preference to use or provide a specific service, QoS requirements, the fact of being part of security or administrative domain, etc…

A3.1.2 Benefits

Most of network related properties require aggregation of nodes having the same property into one group (ex. create a multicast group or form a security domain with nodes belonging to the same authority, etc.). The most important benefit of using property set based self-organization algorithms consists of providing a uniform algorithm to create and maintain these groups. An other benefit is the implicit creation of hierarchical group structure providing and maintaining a scalable infrastructure for functional entities. Furthermore, evaluation in section A3.3 proves that in many cases, common maintenance of these groups by one uniform algorithm can significantly decrease group maintenance cost compared to the total cost of per group maintenance.

A3.1.3 Property set vs. policy based model

The policy based self-organization model is a generalization of the property set based model. In policy-based algorithms, property sets are replaced by more generic policies allowing specifying rules and dependencies between these rules. The concept is very similar, but instead of maximizing the number of common properties inside one group, the aim is to group together nodes with most compatible policies.

Flexibility of the policy based model fits better the needs of ambient networks, however, for the sake of simplicity, functionality of the property set/policy based self-organization model will be presented using the property set based model.

A3.1.4 Algorithm specification

The rational behind the hierarchical self-organization algorithm for the property set based model can be best understood in two steps:

As a first approximation, we ignore the dynamism of the network, take a snapshot of the physical network topology and calculate the optimal (see definition of optimality in section A3.3.1) hierarchical structure for the given physical topology. It is possible to recalculate the optimal hierarchy every time that there is a change in the network topology. However, this would be an extremely expensive and resource consuming task and is merely impossible in large and fast changing networks. Thus there is a need to trade off between performance and optimal network structure. Therefore the final distributed algorithm uses heuristics instead of seeking the exact optimum.

Static centralized approximation

The first approximation of the algorithm (static centralized version) is based on hierarchical clustering widely used in data analysis and data mining [34]. Initially, each node has its own cluster and all of these clusters are member of a working cluster set. At each round, we select from this set the subset of neighbouring clusters with the largest number of common properties. If there are multiple subsets with the same largest number of common properties, then we choose the largest one from these subsets. Clusters of the selected subset are removed from the working cluster set and are assigned a new parent cluster. This parent cluster is inserted into the working cluster set, and the cycle restarts until either the working cluster set consists only of one single cluster or there are no more clusters with common properties. Thus the output of the algorithm is a cluster hierarchy with one or more root nodes.

Dynamic distributed algorithm based on heuristics

In a dynamic network, new nodes can join or leave the network, and new connections can be established or broken between nodes as a result of node mobility. Overlay hierarchy should adapt to these changes.

Keeping an optimal overlay hierarchy after changes in the physical network topology would often require many changes in the hierarchy resulting in a large communication overhead and very unstable hierarchy. Therefore a trade-off is needed between optimality and stability of the hierarchy and instead of maintaining an optimal hierarchical structure, the goal is to maintain a near optimal and only slowly changing hierarchy.

Another characteristic of this algorithm is its distributed nature: there isn't any central coordination when adapting hierarchy to changes in physical network topology. Everything is performed via peer-to-peer self-organization operations between nodes and overlays. These operations are always triggered by changes in the physical network topology, and changes may spread upwards in the hierarchy following a bottom-up principle. This approach ensures that changes are handled locally, and most of these operation series will terminate at lower hierarchy levels. A basic self-organization operation has been defined for all triggers related to changes in physical network topology. There are four possible triggers:

- new connection between two nodes (new edge in the topology graph)
- new node power up (new node in the topology graph)
- lost connection between two nodes (edge deleted in the topology graph)
- node fails or power down (node deleted from the topology graph)

The first two triggers will invoke a composition operation while the second two triggers invoke a decomposition operation. Composition is an active process trying to create a more optimal logical network structure taking advantage of new network connections.

Decomposition is in contrast a passive process trying to “correct” the overlay network hierarchy when some links are broken.

Composition operation

When a new link is established between two nodes, this triggers a composition operation between them, which may then spread upwards in the overlay hierarchy. Bootstrapping of a new node can be considered in a similar way since this also results in new links to its neighbouring nodes. Given two overlays (or nodes) A and B, the outcome of the composition operation depends on the property sets (S_A and S_B) of the two nodes and the property sets of their parent overlays ($S_{p(A)}$ and $S_{p(B)}$)). Depending on relationship between these property sets, the following operations are possible (see basic graph operations in [33]):

- merge(A, B)
- merge(B, A)
- join(A, p(B))
- join(B, p(A))
- expand(A, p(A)) + expand(B, p(B)) + merge(p(A), p(B))

or alternatively, the to overlays (nodes) may decide to forward the composition process to the upper level overlays (either A or B or both) or abort the composition process.

A and B proceed using the following algorithm:

- Exchange property sets S_A, S_B (and $S_{p(A)}$, $S_{p(B)}$ when applicable)
- If $S_A \cap S_B = \varnothing$ then abort composition and stop the algorithm (if there are no common properties, it is not possible to enhance the hierarchy).
- otherwise go to step □
- Examine which operations are allowed taking into account existing hierarchy. Merge(A,B) is not permitted if A or B is a leaf node while the expand() operation can only be applied to root overlays.
- For each allowed operation, evaluate the difference of estimated maintenance cost before and after the operation (see “relative optimality in overlay hierarchy” in section A3.3.1) and select the one with the largest gain.
- If gain exceeds a threshold, then perform the selected operation and stop the algorithm
- If gain is not large enough, optimality is traded off in favour of stability. Go to point □
- Forward composition to parent overlay (if it exists). If both parties are top-level overlays, then abort the algorithm. Note: forwarding composition at one or both sides depends on relationship between property sets and should be optimized based on further performance evaluations.

Decomposition operation

When a link is broken between two nodes, it may result into partitioning of an overlay. In this case, the overlay hierarchy should be also modified by splitting this overlay. The algorithm for decomposition is as follows:

- Find lowest level common ancestor overlay of the two nodes.
- If there is no such ancestor (they belong to two different top level overlay), then do nothing and stop the algorithm

- Else go to step □
- Determine whether this lowest level common ancestor overlay has been partitioned or not. If partitioning occurred, then perform a split() operation

A3.2 Promise theory based self-organization model

The promise theory based approach is an attempt to use promise theory to maintain hierarchical overlay structures over a physical topologies consisting of autonomous and cooperating agents. A detailed description of this work can be found in [35]. For evaluation purposes, a novel regulative policy system has been developed based on Promise Theory whereby Ambient Network nodes cooperate without any central administration.

A3.2.1 Promise Theory

Promise Theory [35] is a graph theoretical framework that simplifies the understanding of complex relationships in network environments where certain constraints have to be met. The basic idea is that totally autonomous nodes interact with each other through promises that they make. Cooperating autonomous agents are organized into new structures. These structures are created on the grounds of conventional management principles. Agents can assert promises to one another agent. Each promise implies a constraint on the behaviour of the promising agent. This construction enables studying the behaviour of a system of autonomous agents, and find inconsistencies and contradictions.

In promise theory, every policy object is regarded as an autonomous agent that works with other parts of the network through voluntary cooperation. Promises mean an agreement between actors of the network.

In the definition, nodes and promises made between them are kept separate from each other. To build more complex structures form promising agents, two special promise types have to be introduced. First of these is needed for forming cooperative groups. When a node promises that it behaves like another node, a group can be formed. The node's promise is called the cooperative promise. Another special promise type is the use promise, when a node promises to use the promise of another node. This promise is important for modelling client-server interactions. Individual promises can be combined to allow agents to form cooperative groups.

A3.2.2 Promise-based cooperation and hierarchical management

Three different methods were identified to map promise-based cooperation concept onto self-organization models. In the first and second methods, the hierarchy of overlays is already created, and promises are used to guarantee services between network nodes. These two techniques fit as an extension to the policy based self-organization model. The third technique however presents a new self-organization algorithm, where the overall hierarchy of the network is built on promises made by individual nodes.

Building overlays with promises

By default, each node has its own profile. Profile is a representation of what the node can promise for other nodes when joining the network. It determines the overlays formed later in the network during composition processes induced by newcomer nodes. Nodes are organized into hierarchical administrative groups based on regulative principles. For instance a node can be the member of a group called "hu.bme.tmit" (which means, that the node can join the overlay "hu.bme.tmit"), another node can be the member of "AN.wp1" and so on. A node can be the member of more than one group. The promises of nodes are based on this scheme, so if a node is in group "hu.bme.tmit" it promises "I am in hu.bme.tmit". There can be a huge number of administrative groups present, and available groups always adapt to the needs and the environment where the current Ambient Network is used. Hierarchy in administrative groups means that if a node is in group "hu.bme.tmit", it is implicitly in "hu.bme" and "hu". In case of promises, this hierarchy works the same way.

Hierarchy is built up with cooperation promises. If two nodes interact and compose with each other, they promise to each other what is in their profiles. If they can cooperate, they promise cooperation to each other, and they join the overlay, which represents the group of nodes that cooperates in the same thing. Cooperation is done by cooperative promises. An example scenario about how this process works can be seen on Figure 69.

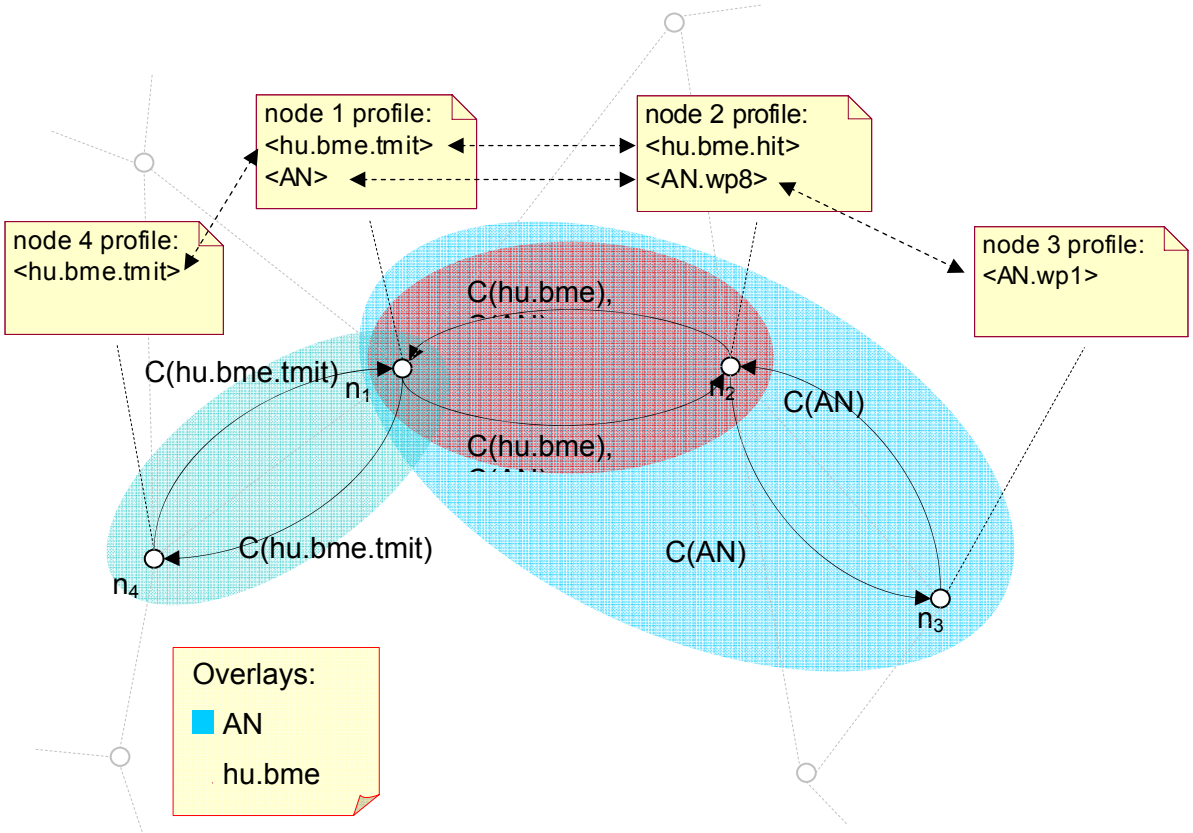


Figure 69 – Creating overlays with cooperative promises

The overlay “AN” is totally independent from the overlays “hu.bme” and “hu.bme.tmit”. But in case of “hu.bme” and “hu.bme.tmit”, the structure is not a tree. Despite of the fact that node n4 is able to join the overlay “hu.bme”, it is not the member of it, it did not join it. This is because, according to the proposed solution, overlays are created on demand. If n4 does not need the services of “hu.bme” and does not want to share its services with this overlay, it does not join it.

A3.2.3 Simulations

Two types of promise simulations were implemented. The first implementation works only in a deterministic way. This simulation engine does not uses overlay utility values. If a node is able to join more than one overlay, it joins all the overlays it is able to join. Cooperation is done in a deterministic manner. Each node negotiates and tries to cooperate with its closest neighbour. If the cooperation fails, the node tries its second closest neighbour, and so on.

In the other implementation, a utility value is calculated for each overlay the node is able to join, and the node chooses the best one to join. Various strategies were implemented for this simulation engine. Common in these strategies is that only low-level overlays are created automatically, in contrast to the deterministic strategy, where higher-levels are created as well.

A3.3 Evaluation of self-organization algorithms

In order to evaluate stability, communication overhead and optimize algorithms, metrics and utility functions have to be defined. First, we have defined these evaluation criteria and metrics for the relatively simple and abstract property set based model. These evaluation criteria follow the two step approach used for the definition of the property set based algorithm: as a first step, we define metrics to describe optimality of an overlay hierarchy on top of a physical network, then as a second step, we define criteria to describe dynamic properties of the algorithm (such as temporal evolution of hierarchy, stability, communication overhead)

A3.3.1 Optimality of overlay hierarchy for property set based models

We define optimality of hierarchy using a comparative approach by comparing estimated maintenance cost of the property set based algorithm to the estimated maintenance cost of an equivalent flat clustering algorithm. Recalling section A3.1.2, one of the main application areas of property set based self-organization algorithms is grouping nodes with similar properties into one group, and maintaining these groups as the underlying physical topology evolves in time. This can also be achieved by flat clustering algorithms, maintaining separate clustering for each property. However, the property set based algorithm may decrease group maintenance cost by aggregating multiple properties into the same group. The metric defining optimality of hierarchy is based on this comparison to the flat clustering algorithm as reference.

It is important to note that the original property set based algorithm doesn't offer full functionality of per property flat clustering. For some properties, one single cluster created by per property clustering may appear as multiple clusters in the hierarchy created by the property set based hierarchical algorithm. In order to make a fair comparison, it is necessary to introduce a complementary clustering for each property grouping into one cluster all the overlays in the hierarchy whose nodes would belong to the same per property cluster.

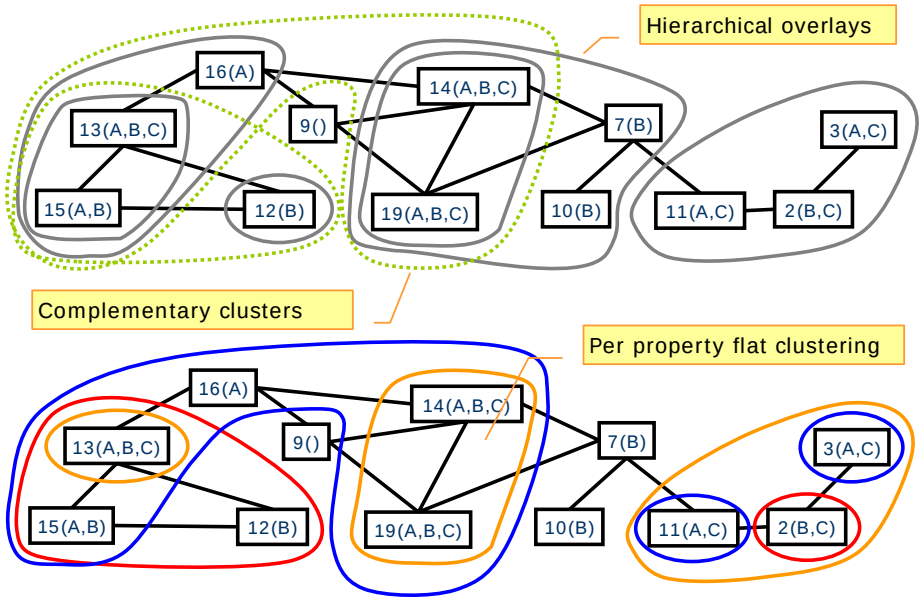


Figure 70 - Evaluate optimality of overlay hierarchy (Example)

The metric describing optimality of the hierarchy is then defined as the ratio of estimated maintenance cost of flat clustering versus property set based clustering + complementary clustering. As a rough first approximation, we assumed that group maintenance cost is linearly proportional to the number of group members both for per property clusters, overlays in the hierarchy and also for complementary clusters. Thus numerically:

- $COST_{flatClustering} = \sum_{property}$ clusterSize
- $COST_{hierarchicalOverlays} = \sum_{hierarchy}$ clusterSize + $\sum_{complementary}$ clusterSize
- savings = ($COST_{flatClustering} - COST_{hierarchicalOverlays}$) / $COST_{hierarchicalOverlays}$

Figure 70 demonstrates calculation of the optimality metric described above. Numbers in the figure represent node IDs while letters correspond to unary properties of these nodes.

- $COST_{flatClustering}$ = (5+1+1)+(3+4+1)+(1+2+3) = 21
- $COST_{hierarchicalOverlays}$= (2+2+1+2+3+3)+(2+2) = 17
- Savings = (21 – 17) / 17 = **23.5%**

Simulation results

Simulation results have shown that optimality of hierarchy created by the property set based self-organization algorithm depends largely on the following factors

- physical network topology (ex.: average connectivity degree)
- distribution and dependencies of properties

Simulations have been performed using unary properties. Unary properties can take one single value (ex. multicast membership for a given multicast group – the property is present if the node is member of the group). They have been chosen because all properties with discrete values can be described as a set of unary properties.

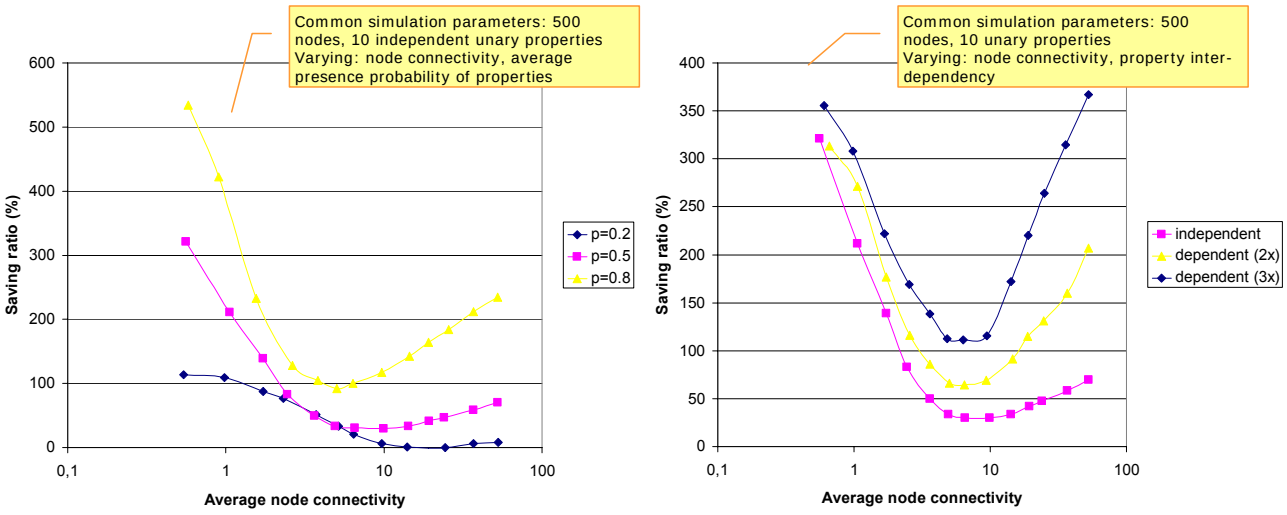


Figure 71 - Evaluation of property set based self-organization algorithm for unary properties

Simulation results in Figure 71 show that the property set based hierarchical algorithm performs significantly better than the flat clustering based reference algorithm if

- average node connectivity is high
- and/or average presence probability of properties are high
- and/or there are significant dependencies between properties

The rational behind these results is that all of the above factors increase the number of common properties per group and this implies smaller maintenance costs than using per property flat clustering. Figure 71 shows high savings for very low average connectivity values too. However, these are not real savings, since in these cases most clusters consist of one single node.

Relative optimality of overlay hierarchy

During a composition operation, the property set based clustering needs to compare optimality of hierarchy before and after a specific composition operation. This can be computed using the presented optimality metric by calculating the difference for both the overlay hierarchy and complementary cluster maintenance cost and finally adding these two differences.

The difference for estimated overlay hierarchy maintenance cost is easy to calculate for each basic graph operation (ex. it is 0 for merge and +1 for a join). Exact calculation of the difference for estimated complementary clustering cost would require to perform a per property clustering, however, good estimates can be given simply relying on property sets of composing overlays.

A3.3.2 Continued work

Future work is needed to define criteria and metrics to describe dynamic properties of property set based algorithms such as stability, evolution of hierarchy in time and maintenance overhead in function of node mobility.

New metrics are also needed to extend property set based evaluation criteria for policy based algorithms.

A4 Peer-to-peer Management with Patterns

For centralized network management approaches it is hard to adapt to the always changing behaviour of dynamic ad hoc networks, where nodes are moving, appearing and disappearing. A better way for the management of such networks is to apply a decentralized network management solution, like the pattern based network management approach and especially the echo pattern.

Echo patterns were introduced for distributing management operations. These operations do not need a-priori knowledge of the network topology, they can dynamically adapt to changes. A management operation based on the echo pattern has two phases. First in the expansion phase, the network is flooded with management commands to be run on the network elements and second in the contraction phase, the results of the local management operations are aggregated inside the network and propagated back to the initiator of the management command.

The echo pattern could be used for dynamic, on-the-fly FE or service discovery, for real time monitoring and control, or for other kind of information distribution and collection like exchanging policies between nodes. One of the main question is to discover on which topology the echo pattern should run when sending out queries and gathering the necessary information. Should it follow the physical topology of the network and, if not, how should the overlay topology look like, to run the echo pattern in an efficient manner (e.g., using the less hops and messages as possible)? In addition an analysis of the suitability and performance of the echo pattern and a comparison of the echo pattern with distributed database approaches needs to be performed and decided which solution is more effective in ANs.

It would be useful if an encrypted echo pattern will be applied in ANs, and all nodes belonging to an AN would own a security key of the AN, which key could be used to decrypt the encrypted echo distributed inside a given AN. It is necessary, that a node owns the key of each AN it is belonging to. The superpeer is generating and distributing these keys inside the ANs. The encrypted echo pattern would allow not to follow the hierarchy of the AN (super-superpeer to superpeers to ordinary nodes), but the echo could run on an arbitrary overlay topology, which for example could be constructed to generate the least message exchange between the nodes for a given echo query.

Figure 72 shows a case, where running the echo pattern on the overlay structure would lead to more messages in the physical structure, but Figure 73 shows a case, where the overlay is structured that way, where running the echo pattern on the overlay would generate less messages in the physical network, compared to the case of running the echo pattern on the physical network itself.

So in the future we plan to answer the following questions. One question is what kind of measures exist which show the efficiency of a topology to run the echo on. The second question is that, how can we analyze a given topology (a graph) to monitor these measures continuously. Finally the third question is that how can we create and adapt an overlay topology to have the selected measures on an optimum.

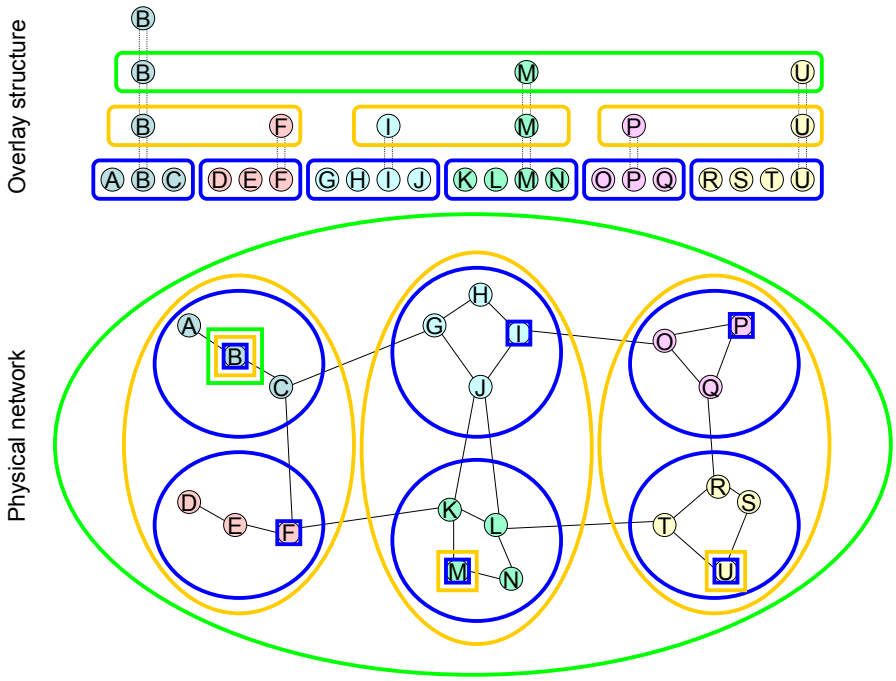


Figure 72 – Overlay and physical structure example – in this case using the echo on the physical structure would lead to fewer messages in the physical network

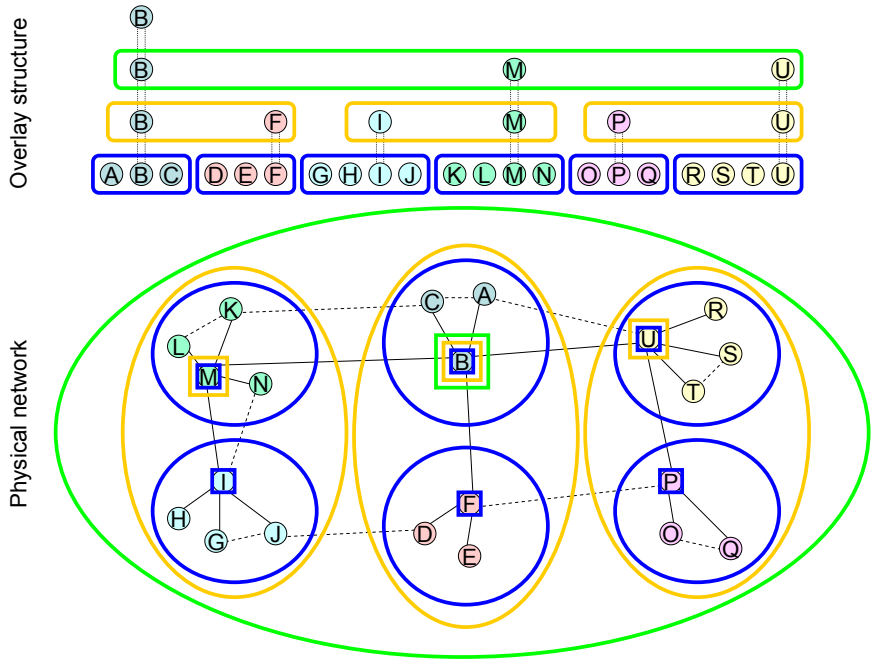


Figure 73 – Overlay and physical structure example – in this case using the echo on the overlay structure would lead to fewer messages in the physical network

A5 Other Management Functions

We have in this chapter described the different parts of network management we are focusing on and working with during phase 2 of Ambient Networks. We have described how to perform decentralized real-time monitoring of the network and network nodes with A-GAP, and how we can drastically lower the management overhead by tolerating a certain level of error between the actual state and perceived state of the network. We show our work on formal models for self-organisation where we describe three formal models, the property set model, the policy-based model and a model based on promise theory. We also present an early attempt on how to evaluate the effectiveness of these models. We continue talking about how self-organisation will work during compositions and what requirement this puts on other system components. We describe how we want to do distributed auto-configuration of which radio channels different base-stations (in our case 802.11 base-stations) different base-stations will use, and how to do continuous load-sharing of traffic between the base-stations. We continue by talking about how we can use pattern-based management to monitor, gather and distribute instructions to nodes in different overlay networks directly without going through the different overlays but still not including all nodes in all connected networks, before we describe how we will modify distributed hash tables to continue to work during (de)compositions without using all available resources to update the DHTs and keep them consistent. All of these aspects and approaches are vital and prerequisites for a complete management system that will serve the Ambient Networks.

However, there are many other aspects that must be researched and solved as well before we can claim we have an integrated management solution for Ambient Networks, and due to limited amount of resources we will not be able to investigate all these aspects. We will list some of them below anyhow to be used in the future as the project continues and the resource situation might change.

We need to further investigate different self-configuration and self-organisation aspects. In phase 1 we investigated how WLAN base-stations can use plug-n-play techniques to configure themselves, and in phase two we continue this work with how to choose radio channels and do load-balancing. However, we also need to investigate other aspects of self configuration, such as how this can be applied to other types of nodes, how this can be extended to work with networks as well as nodes, how this can be done in a secure and verifiable way, etc. We also need to investigate how self-configuration will interact and correlate with changes in the networks and fault management aspects; self-organisation is a continuous process rather than something only being done during start-up of a node.

We need to further investigate how to deal with changes in the network, events and faults. We need to investigate how we could do alarm correlation and alarm filtering in a distributed way. It will not be possible to send all alarms upwards and correlate them in an OSS to detect the root cause, this must happen inside the network. We also need to define ways to actually report alarms to the outside world in the cases when this is necessary. At the same time we need the network to adapt and hide faults in the network to minimise the impact of these faults before they can be dealt with by human operators in the case of hardware faults. We should further investigate alternative methods for dealing with alarms, for example how to detect something is about to go wrong and deal with it before it actually goes wrong.

Related to this is how will we continuously optimise our networks, the traffic throughput, service availability etc. in an ever-changing world. During phase 1 we did some work on traffic engineering, and we're investigating in phase 2 how to do load-sharing between base-stations, but we also need to investigate aspects such as how to optimisation in networks where different nodes have different capabilities (batteries, processors, bandwidth,…), how to prioritise between wishes and available resources, how to make

sure we can actually get access to physical nodes and networks even if they are heavily loaded, etc.

And last but not least, we need to identify how all these different approaches and aspects will work together to form an integrated management approach, for example how can A-GAP and patterns be used together with re-configuration of recourses, or how can DHTs enable efficient composition of networks. We need to go beyond proof-of-concept of different mechanisms and start using actual data to be sent and see how these mechanisms will work together. And security is of course vital, the lack of proper security mechanisms are a showstopper.

A5.1 DHT composition and Decomposition models

An AN consists of potentially large numbers of independent, heterogeneous mobile nodes that can logically interact with each other to share a common control space, known as the Ambient Control Space (ACS), for resource sharing. Given that an AN may consist of large number of AN nodes, there is clearly a need for a distributed and scalable management data storage and retrieval mechanism for each AN to support composition management. Composition management was described in Chapter 2. Due to the heterogeneous natures of (potentially large number of) AN nodes, composition management in AN must be supported by a scalable and distributed database to store management primitives and management information that are useful to support the composition management. In order to support composition management, some form of distributed registry is required to carry out mapping between Universal Context Identifiers (UCIs) and location of context objects. For instance, the Context Sensors discussed in section 2.2.5 are used for monitoring composition management required network parameters; these monitored parameters (i.e. context objects) must be mapped by some entities to UCIs; the technique of enforcing this mapping must be distributed and scalable.

Previous research works [18][19][21] have suggested that Distributed Hash Tables (DHTs) is a candidate. Example DHTs are Content Addressable Network (CAN) [20], Chord [23], Pastry [22], and others. However, much of the existing research mainly focuses on optimising DHT routing and scalability; and usually assumes a common DHT across the entire network that any nodes can join [18]. Since ANs may constantly compose and decompose with other ANs[19]. We argue that the successful use of DHTs in implementing various distributed management components in ANs depends on an ability to efficiently compose and decompose DHTs. By DHT (de)composition, we refer to member nodes of homogeneous or heterogeneous[20] DHTs of different ANs interacting with each other to share distributed information.

The challenge is that DHT (de)composition in ANs must be conducted in a resource-limited environment i.e. a wireless environment with limited power, processor power, storage capabilities, etc. Thus, in addition to the need for a more efficient underlying routing algorithm in DHTs (which is beyond the scope of this section), there is a need to minimise

[19] By AN composition, we refer to process of which the ACSs of two (or more) ANs interact with each other, to establish a common ACS between the two (or more) ANs for resource sharing. AN decomposition refers to the process of a common ACS being divided.

[20] By homogenous DHTs, we refer to DHTs that use keyspace of the *same* keysize (e.g. both uses 160-bit keyspace). By heterogeneous, we refer to the opposite (e.g. 160-bit keyspace Vs. 256-bit keyspace).

the disturbance caused by the (de)composition process to existing member nodes of the DHTs. By minimising the disturbance, we mean to minimise: (a) the amount of network overhead incurred by the (de)composition process, and (b) the amount of storage data that needs to be (re)distributed to other nodes in the DHT after (de)composition has completed (during which keyspace might have been re-assigned to new members). As far as we are aware, there has not been a vast amount of research work on DHT (de)composition; a DHT merging process designed for DHTs based on the Chord protocol was presented in [[18]]. Here we outline two novel (de)composition models for CAN-based DHTs for AN. We shall also discuss the mappings of our models to DHT implementations other than CAN. We start our investigation based on CAN-based DHTs because of its design simplicity, which could help readers understand this new research challenge (of DHT (de)composition). Furthermore, this would enable us to gain a better understanding of the design requirements of DHT (de)composition, and put ourselves in a better position to evaluate, experiment with, and to tailor a generic DHT (de)composition model that would also cover other DHTs.

A5.2 Dynamic Composition & Decomposition of DHTs in Ambient Networks

To simplify our discussion, we assume that:

d) each AN has DHT-based management components;

e) each AN node should have its own keyspace in the DHT once it has joined a DHT;

f) each node maintains a coordinate routing table that keeps IP addresses and virtual coordinate zones of its neighbours in the approach as indicated in [20].

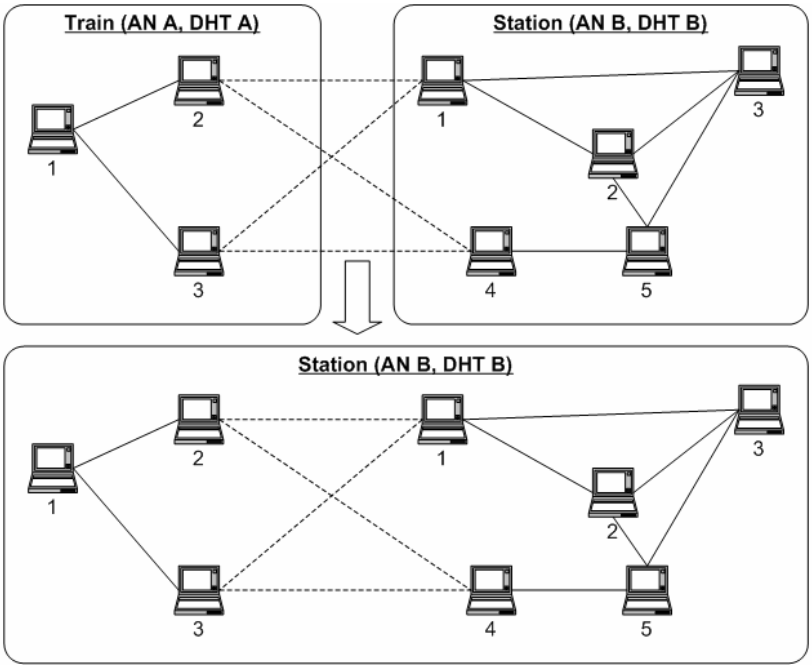


Figure 74 – Composition through absorption

We present two DHT composition models for ANs, known as *absorption* and *gatewaying*. The absorption model (Figure 74) refers to two (or more) individual DHTs (that are owned by two or more ANs respectively) completely merging together, resulting in one *uniform* DHT across the composing ANs. The gatewaying model (Figure 75) refers to *bridging* two (or more) individual DHTs together without modifying their original keyspace. Both approaches enable information sharing between DHTs, but are tailor-designed to accommodate *different* network environments. We do not intend to specify the exact criteria when absorption or gatewaying between DHT should be triggered; we believe these

issues should be defined by the corresponding AN/DHT Service Providers (SPs) (readers are referred to [24] for more details). Also, SPs should define policies for guiding nodes when nodes are presented with multiple joining offers from different (nearby) DHTs. To illustrate the differences between absorption and gatewaying, first, consider this deployment scenario: when a train arrives at a train station, an unknown (but potentially large) number of passengers will be getting off the train and join the station. Assume each passenger is a passenger node, and DHTs have already been established among member nodes of the station and the train respectively (i.e. the station DHT and the train DHT in Figure 74). Because the number of passenger nodes getting off the train is unknown (i.e. a very dynamic situation), it would be difficult to establish a *fresh*, *new* DHT (e.g. passenger DHT) among members of such a highly dynamic group in *real-time*. Instead, because the station DHT is readily available, should passenger nodes (those getting off the train) wish to become members of a DHT (say, to share information), they should be *absorbed* into the (more static) DHT (i.e. the station DHT). The absorption model is designed to minimise the network overhead on members of an existing DHT when many nodes attempt to join practically at the same time. The gatewaying model is suitable when one (or more) of the ANs/DHTs come together within reachable distance, but member nodes of the AN/DHTs remain (relatively) static. For example, when two wagons are joined together at an intermediate train station (Figure 75), member nodes of the two wagons are likely to be static (some passengers might get off the train but the majority would stay). There is a need to bridge between the DHTs, so that the DHTs can share information.

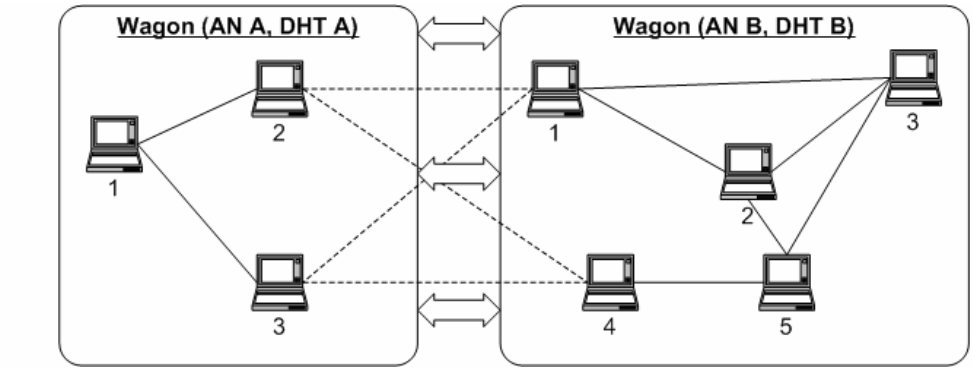


Figure 75 – Composition through gatewaying

A5.3 DHT Composition through Absorption in ANs

One way of enabling absorption (as suggested in [19]) would be to allow nodes (of a discarded DHT, or do not belong to any DHT) to join a stable DHT, by negotiating with nodes of the stable DHT individually using the standard procedure as [20]. Typically this would require each of the joining nodes to randomly select a keyspace, and obtain keyspace directly from the node that "owns" it in the stable DHT. We refer this as *simple merging*. The advantage of simple merging is its simplicity, i.e. no changes to the existing protocols are needed. But the drawback is that *all* key-value pairs hosted on member nodes of the discarded DHT or on individual nodes would have to be re-distributed to nodes of the stable DHT. Also, if keyspace is randomly selected, many nodes in the stable DHT must also update their neighbourhood information each time a new node joins (see later for more details). In the case of large-scale deployment, this could potentially create unnecessary network traffic, which is not desirable, particularly in wireless networks.

We suggest that, absorption negotiations should be conducted through the point(s) of contact between the (two) ANs only. Points of contact are the nodes that have *physical connections* with other ANs. For example, node A2 and A3 are the points of contact of AN A (Figure 74); whereas node B1 and B4 are the points of contact of AN B (Figure 74). Instead of node A2 and A3 randomly selecting points in the keyspace of any nodes of DHT

B (which is the case in [19][20]), node B1 and node B4 will give up some of their keyspace to node A2 and A3 respectively, by carefully selecting appropriate keyspace from *within* the keyspace that they own. Note that the entire absorption process is a transient process, which ceases to operate after a timeout. After the timeout, new nodes will be joining the (unified) DHT under the normal procedure; that is, by randomly selecting keyspace from any nodes in the DHT.

Note that a key feature of absorption is that keyspace is selected *within* a keyspace (instead of splitting), with the goal of reducing the level of disturbance to neighbouring nodes. Figure 76[21] shows different keyspace partitioning approaches in absorption. Figure 76a shows the original keyspace of DHT B (before absorption). If nodes of DHT A are allowed to randomly select keyspace from member nodes of DHT B, or if node B1 and B4 simply split up their keyspace for node A2 and A3 respectively, the resultant keyspace might end up as shown in Figure 76b. As a result, node B1, B2, B3, B4 and B5 would *all* have to update their neighbourhood information. This is obviously less desirable. However, by carefully selecting keyspace within the owner's keyspace (Figure 76c), only node B1 and B4 would have to update their keyspace respectively. Note that (internally) the data structure that a node uses to represent the keyspace that it owns consists of the set of edges that demarcate the boundaries of its zone with respect to that of its neighbours. The points that lie exactly along these edges, which are inclusively part of its zone (and keyspace) and which are directly adjacent to neighbouring zones, are not given out to the new joining nodes during absorption. This ensures that any keyspace allocated to absorbed nodes is purely internal to the absorbing node, and is not directly adjacent to its neighbours. Hence, less (neighbouring) nodes are disturbed.

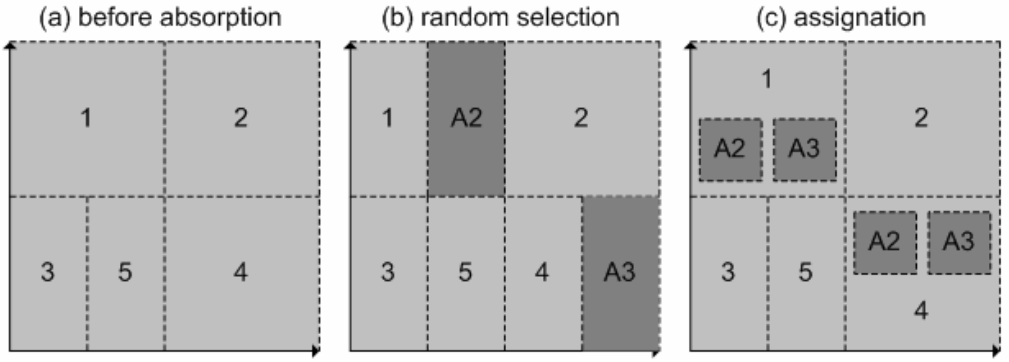


Figure 76 – Different composition approaches

Once dedicated members of DHT A (i.e. node A2 and A3) have been assigned with keyspace, they will (re)distribute the keyspace to other members of DHT A wishing to join DHT B (i.e. node A1). This arrangement is again to minimise the disturbance to existing member nodes of DHT B. Imagine if all the remaining nodes of DHT A (i.e. node A1, and potentially many more) join DHT B using the standard procedure: if the scale is large (i.e. many nodes joining at once), existing member nodes of DHT B would have to expend significant resources on the tasks of keyspace partitioning and updating neighbourhood information. Thus, the remaining nodes (e.g. node A1) should *not* obtain keyspace from

[21] For simplicity, we illustrate our examples using a 2-dimentional coordinate space.

nodes other than their points of contact (i.e. node A2 and A3). When node A1's `join_DHT` request traverses through node A2 (or A3), node A2 should terminate the request; node A2 should select *within* the portion of its assigned keyspace, and return the selected keyspace to node A1. The same approach is repeated between node A1 and other nodes of DHT A: when node A1 has intercepted a `join_DHT` request from other nodes of DHT A, node A1 will terminate the request, and will response with a selected portion of keyspace within its own keyspace. In this way, each node is responsible for (re)distributing keyspace; thus, a distributed approach for keyspace (re)distribution among the joining nodes is achieved.

The advantage of the absorption approach is that once keyspace has been assigned by node B1 and B4 to node A2 and A3, (node B1 and B4 would have updated their own neighbourhood information by then), all other existing member nodes of DHT B are not disturbed. The process is entirely transparent to other nodes that originally own DHT B; and other nodes that were members of DHT A may join DHT B through node A2 and A3 (and subsequently through node A1 once node A1 has obtained its keyspace from node A2 and A3), which is also a transparent process to existing member nodes of DHT B. Unlike the simple merging approach, our approach requires only a few nodes of the DHT (i.e. DHT B) to be disturbed, and with less network traffic (i.e. negotiation and communications between AN/DHT are conducted between the points of contact only). Furthermore, because each node is capable of (re)distributing keyspace, keyspace distribution is achieved in a distributed and scalable manner.

It may appear that the requirement of explicit assignation of keyspace by keyspace owners (e.g. node A2) to new joiners (e.g. node A1) violates the random balancing rules in DHTs (i.e. nodes should randomly select keyspace for balance loading). However, we argue that absorption is a transient procedure. After the timeout, new nodes must join through the normal procedure; the keyspace would eventually be randomly and evenly distributed on average. It is possible that all nodes in the DHTs have physical connectivity with each other (i.e. a fully meshed structure of physical connectivity). In this case, one may argue the use of absorption, because effectively all nodes are points of contact. However, if all (wireless) nodes are physically interconnected, then the nodes must be within close physical range. This implies that the number of participating nodes in this merging would be limited. Thus, the amount of network traffic created by, say, a simple merging, would have much less effect. It should be noted that the use of dedicated points of contact for handling keyspace does not affect scalability of the absorption model. The points of contact are responsible for *initially* collecting a portion of keyspace from nodes of the other DHT (i.e. DHT B), and redistributing keyspace to their *immediate* neighbours only. Once the immediate neighbours have obtained their keyspace from the points of contact, the immediate neighbours shall intercept (and terminate) any traversing `join_DHT` requests from other member nodes (of DHT A), and (re)distribute keyspace to those nodes. Therefore, keyspace (re)distribution to member nodes of DHT A is carried out in a distributed fashion. By selecting keyspace within a keyspace, the resultant keyspace (of the owner) may end up being in "irregular shapes" (if the keyspace is represented in x-dimensional graphical forms); which may *seem* to affect the efficiency of locating a point within the keyspace for future data lookup or keyspace splitting; and irregular-shape keyspace may *seem* to have more neighbours (i.e. more boundaries). Note that according to the original design of CAN, the only requirement on keyspace splitting is that the owner splits its keyspace into halves along one axis first, then another, and so on. Determining whether a point is located within an irregular-shaped keyspace is simply a mathematical problem that can be easily resolved on-the-node (e.g. by geometry). Furthermore, nodes that own "irregular-shape" keyspace do not have more new neighbours because keyspace are assigned from *within* a keyspace; thus, only one node (i.e. the owner) is affected.

Once new nodes have joined a DHT, they may distribute their local data to other nodes in the DHT if desired (i.e. the `put(key, value)` operation). To minimise traffic caused by many

nodes putting data onto many other nodes at one time, provisioning [21] has been made in our approach to upload pointers only, instead of the actual piece of data. For example, if node A3 needs to put a piece of data on node B2 (Figure 76c), a pointer is put instead of the actual piece of data. The pointer refers to the actual location of where the data is residing on (e.g. the data source). Thus, the amount of data storage traffic caused by (many) new joining nodes is reduced. One may argue that this arrangement increases the round-trip delay for retrieving a piece of data. However, this approach is ideal for situation where the data to be stored requires frequent updating, such as real-time bandwidth monitoring data. Instead of the data source continually updating the data values on a remote node, a requester – once obtained a pointer to the data source through the DHT – can contact the data source *directly* (readers are referred to [21] for more details).

The absorption model is different from the overloading coordinate zone approach proposed in [20] (Figure 77). The latter allows multiple nodes (known as peers) to share (transparently) a keyspace with the original owner who owns that keyspace. It is designed to improve fault tolerance and to reduce path length and per-hop latency. It may appear that there are similarities between the two approaches. However as stated in [20], besides keeping a list of neighbours, the original owner of a keyspace (i.e. node B2 in Figure 77b) must also keep state of *each* of its peers (i.e. node A1, A2 and A3 in Figure 77b); and all peers must also keep state about the other peers of the zone, *and* the neighbours of the original keyspace owner. For example, in Figure 77b, node A2 would have to maintain state of *five* other members (i.e. node A1, A3, B1, B2 and B4). This arrangement adds complexity (i.e. additional overhead). Whereas in the absorption model, once keyspace have been assigned/spitted to new joining nodes; the new joining nodes are simply treated as neighbours. As shown in Figure 76c, node A2 would have to maintain state of one neighbour only i.e. node B1. This suggests the absorption model is more scalable than the overloading coordinate zone approach.

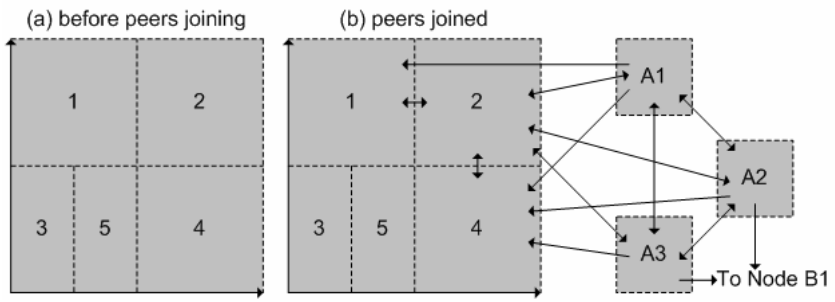


Figure 77 – Overloading coordinate zones

A5.4 DHT Composition through Gatewaying in ANs

We have mentioned in earlier section that through gatewaying, the composed AN/DHTs would be able to share information, but at the same time retaining their original keyspace. Existing DHT approaches usually assume only one common DHT (i.e. one common keysize); however, due to the dynamic and heterogeneous nature of ANs, there is a need to support composition between DHTs of different keysize. Thus, there are two environments in which gatewaying may be deployed: gatewaying between DHTs that use keyspace of the same keysize; and gatewaying between DHTs that use keyspace of

different keysize. When gatewaying between DHTs of the same keysize, nodes of the composing DHTs are notified of the existence of the other DHTs that are now becoming accessible, as well as their own gateways to the other DHTs[22]. Gateways are the points of contact, which have physical connections with nodes of another DHTs; but they serve a different purpose from the points of contact in the absorption model. For example, nodes of DHT B (Figure 75) must be notified that node B1 and B4 are the gatewaying nodes to another DHT (i.e. DHT A). The gateways do no more than notification (i.e. they will not request for keyspace). To gain access to another (gatewayed) DHT, member nodes would need to maintain the state of at least one of their gateways[23]. Let's say node A1 (Figure 75) would like to retrieve a piece of information (after DHT gatewaying). We provide two options for this node to do a search over the gatewayed DHTs: (a) a sequential minimal-evaluation search, and (b) a parallel search. Sequential minimal-evaluation search is ideal for locating unique pieces of data (e.g. a particular video file); whereas parallel search enables a node to locate network-wide information e.g. a node that needs Internet access may wish to search all gatewayed ANs for the best available Internet throughput link that.

In the sequential minimal-evaluation search, node A1 computes the location of the information in its home DHT (DHT A) in the same way as stated in the standardised protocol [20], and tries to get the information from the node (of DHT A) that is supposed to hold the information. Suppose the information of interest is not stored in DHT A (i.e. data not found); with gatewaying, node A1 can now try to retrieve the same piece of information from other gatewayed DHTs (i.e. DHT B). The node of DHT A, which fails to provide node A1 with the requested information, say, node A3, will inform one of its gateways (i.e. node A2), and requests the gateway to search for the same piece of information in other (gatewayed) DHTs. The request is conducted through the gatewaying nodes i.e. node A2 and node B1, as if node A2 is an individual node that wishes to locate a piece of information in DHT B. Node B1 (i.e. the corresponding gatewaying node of DHT B) will try to locate the piece of information on behalf of node A2, and will fetch over the results to node A2 (and subsequently node A1) if the information can be located in DHT B. The search terminates as soon as the information is located (hence the term *minimal evaluation*), and is not forwarded to other DHTs that may be similarly gatewayed in this scenario. In contrast, in a parallel search, as the name implies, the query is forwarded simultaneously (through the gateways) to all DHTs through their respective gateways. If the composing DHTs' keyspace are of different keysize, we use a similar approach as above, except that node A2 must use the correct hash algorithm to compute the correct location in DHT B. For instance, if DHT A's keyspace is 160-bit whereas DHT B's keyspace is 256-bit, node A2 must use SHA-256 to compute the correct keyspace of the requested information. Note that when a new AN node wishes to join a DHT that has already been gatewayed, the new node joins the DHT that it is in contact with.

The advantage of gatewaying is its simplicity and reduction in network overhead. It enables information retrieval across DHTs without modifying existing keyspace structure. Thus there is no need to update neighbourhood information on each node in the gatewayed

[22] As an initial approach, notifications are sent to member nodes of a DHT through multicast. Multicast is chosen for its simplicity.

[23] Ideally for robustness, member nodes should maintain as much state of its gateways as possible. However, there is a trade-off between overhead and robustness. As an initial design, we require each node to maintain at least one of its gateways.

DHTs (which would be required in simple merging or, to some extent, absorption). Also, the chances of successfully retrieving a particular piece of information increase as the number of gatewayed DHTs increases; which enhances the robustness of the overall data retrieval process (i.e. more likely to locate the piece of data of interest). The scale of state maintenance at one gateway is not dependent on the size of the neighbouring networks/DHTs, but depends only on the number of immediate neighbouring gateways, which makes our approach scalable. More importantly, because there is no change to ownership of the keyspace of the gatewayed DHTs, there is no need for data (pointer) (re)distribution, which reduces network overhead. The downside is that the gatewaying model is only applicable when member nodes of the to-be-gatewayed DHTs are relatively static. It may appear that the use of gateways for inter-DHT communications would result in some centralised processing (on the gateways). However, it should be noted that not all `get(key, value)` requests are processed by the gateways; for example, cross-DHT data search takes place only when data retrieval within a DHT fails in sequential minimal-evaluation search, and stops as soon as the data of interest is found; whereas parallel search is used for searching specific-types of information only (e.g. network-wide information). This mode therefore trades off slightly higher traffic costs and processing overhead (on the gatewaying nodes), to enhance overall robustness of the data retrieval process (when comparing to searching for data within one DHT only i.e. no gatewaying), and achieves potentially faster and more comprehensive searches over the entire composed space, with the possibility of returning multiple results from all gatewayed DHTs.

A5.5 DHT Decomposition in ANs

By AN decomposition, an AN is virtually divided into two (or more) ANs, the decomposed ANs do not recognise the existence of each other in the view of control and management[24]. There are several decomposition scenarios. Decomposition between DHTs that were composed through gatewaying is the simplest form. This may happen when, using our previous train scenario for this example, two wagons (which were gatewayed) are detached. Nodes in the gatewayed DHT are informed that the other DHT no longer exists, and the DHTs are said to be decomposed. Decomposition of nodes of a DHT that share one unified keyspace (i.e. may have previously composed through absorption) is slightly more complicated. A node may leave a DHT without establishing its own DHT or joining with another DHT. For instance, a train passenger switches off his laptop. This situation can be considered as a node departure, and can be handled through the standard procedure as specified in [20]: the departing node either handovers its keyspace to a neighbour (i.e. a clean approach), or the unoccupied keyspace will be taken over by its neighbours when its neighbours think it is dead [20]. In other circumstances, a set of nodes may decompose from a DHT, and would like to have a DHT of their own but using the original assigned keyspace. This can be achieved by the departing nodes simply by expanding their own keyspace as if some other nodes have departed from the DHT (Figure 78b and Figure 78c). For instance, if node B1 and B5 are the departing nodes (Figure 78b), they will expand to occupy the remaining space left over by node B2, B3, and B4. The result would be two separate DHTs, but nodes would be able to retain the original assigned keyspace structure. The last scenario would be when some departing nodes decided to

[24] This does not necessary mean the decomposed ANs are *physically* disconnected: we are referring to a separation of control space during decomposition of ANs.

create a new DHT among themselves. In this case, they must discard the original DHT, and creates a new DHT from scratch.

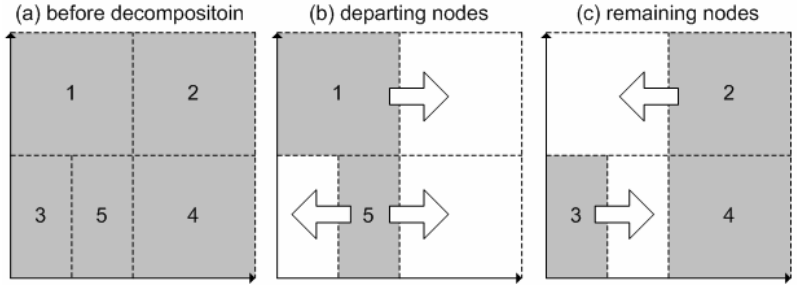


Figure 78 – Decomposition of DHTs in ANs

The concepts between our two models are applicable to other DHT implementations. Many other types of DHTs like Chord and Pastry operate with a circular keyspace. In these DHTs, each node obtains a key randomly distributed around a circular keyspace. Hence, for these types of DHTs, the keyspace of an individual node could be defined as the "space" from midway between its predecessor and itself, to midway between itself and its successor [22][23]. For the absorption model, keyspace for the joining nodes are selected between and within the keyspaces of the point(s) of contact of the DHT. As the keyspace is not randomly selected throughout the entire keyspace, there is minimal disruption to the neighbour tables of the DHT with only those of the point(s) of contacts and possibly their predecessors and successors being affected. In the gatewaying model, composition is exactly as described above with only the tables of the gateways in the two DHTs being affected. The only penalty would be the update message, sent probably through multicast, to inform the other nodes in the DHTs. The overall concepts of (de)composition of DHTs through absorption and gatewaying, can hence be applied to other types of DHTs.

A5.6 Conclusion and Continued Work

We presented two novel models for DHT (de)composition in ANs under different networking environment to support composition management in ANs. DHTs are established within an AN to provide a mapping between UCIs and locations of context objects that are useful for composition management. Thus, composition management can only be achieved if the challenges of DHTs composition are solved. We have discussed how our designs would minimise the disturbance to existing member nodes during DHT (de)composition, taken into account scalability and efficiency. As future work, we intend to investigate further into the inter-node communications during DHT (de)composition in ANs, and deploying our approaches on other DHT protocols.

A6 Maintenance of P2P routing

This section analyses maintenance of routing in structured P2P systems under churn based on the paper “Stochastic maintenance of P2P routing” submitted to the special issue of Elsevier Computer Communications on Foundations of P2P Computing.

P2P routing architectures can be described by a connectivity graph whose vertices are embedded into a metric space. Most existing P2P routing mechanisms define two different connection types in this graph: short-range connections and long-range connections. Our contribution is a novel long-range connection maintenance mechanism, which reduces maintenance overhead to the theoretical minimum without degrading routing performance.

We consider long-range connections as a stochastic process in a one dimensional Euclidian metric space derived from a stationary Poisson process by an exponential transformation. We define long-range connection density independent of network size as the λ-intensity of the generating Poisson process and we show that λ unequivocally determines distribution of node degree and lookup cost in the system. We show that a system with such long-range connection distribution scales logarithmically with the system’s size like most existing structured P2P systems. We propose a loose stochastic maintenance mechanism ensuring that long-range connection density is within a predefined range. We also describe evolution of the system under a given node arrival and departure rate by a Markov chain model. Finally, we show both analytically and via simulations that the communication overhead of long-range connection maintenance scales logarithmically with the system’s size, which is a theoretical lower bound for maintenance traffic to ensure connectivity of the network.

A6.1 Introduction

During the last five years, a significant number of structured P2P systems [109][110][111][112][113][114] have been proposed to store, retrieve and lookup data in a distributed way; most of them based on the Distributed Hash Table (DHT) concept. A DHT defines a common identifier and key space and it is composed of a routing and storage subsystem. Each data item is mapped onto the key space by a hash function. Data (or reference/pointer to that data) is stored at the node whose ID is the closest to the key generated by this hash function. As such DHT is a storage subsystem. To store or retrieve a data element, the closest node to the key has to be found in the network – this is the functionality of the routing subsystem. In this section, we examine the routing subsystem of DHTs, more specifically the efficiency of the maintenance of routing in dynamic network environments.

At first glance, various DHT routing mechanism seems very different. However, the basic principles are common behind all DHT routing designs. Aberer et al. give a very comprehensive reference model to describe common properties of structured P2P routing mechanism in [122] and in the followings, we use their reference model and terminology to present the scope of this section.

P2P routing is based on an overlay network that can be described by a directed or undirected graph. Each routing mechanism uses a one or multi-dimensional metric space defining logical distances between the identifiers of nodes and the overlay network graph is embedded into that metric space. A node is found by forwarding lookup requests to the node being the closest to the target. Efficient lookup is provided by creating two different connection types: local (or short-range) connections and long-range connections. Each node has short-range connections to some specific subset of its closest neighbours and has long-range connections to some distant nodes so that the probability of having a long-range connection between to nodes is inversely proportional to the d^{th} power of their distance (where d is the dimension of the metric space). In [60], Kleinberg has show that

the above are necessary requirements to provide efficient distributed search based solely on local information. Short-range connections guarantee success of greedy forwarding. Since each node is connected to its closest neighbours in the metric space, it is always possible to forward lookup requests at least a small step closer to the target. In contrast, the role of long-range connections is to expedite the lookup process and to provide *O(*log *N)* bounds on the average number of lookup hops. This is achieved by ensuring that the average distance from the target decreases by a constant factor after each lookup step.

The key difference between various DHT routing mechanisms is the degree of determinism in selecting long-range connections. This does not affect routing performance in a static or quasi-static network, but is crucial for maintenance in dynamic environments. *In this section, we argue that determinism in the choice of long-range connections can be completely eliminated without degrading performance of routing and it should be completely eliminated to provide efficient maintenance in dynamic environments.*

Therefore we have chosen as a starting point the generalized Kleinberg “small worlds” model [60], where the only constraint on choosing long-range connections is the inverse power law distance distribution. *We consider long-range connections as a stochastic process in a one dimensional Euclidian metric space derived from a stationary Poisson process by an exponential transformation.* We define long-range connection density as the λ intensity of the generating Poisson process and we show that λ unequivocally determines distribution of node degree and lookup cost in the system. *We propose a loose stochastic maintenance mechanism ensuring that long-range connection density λ is within a predefined range* and we describe evolution of the system under a given node arrival and departure rate by a Markov chain model.

To describe maintenance overhead, we use the approach presented in [61] by Liben-Nowell et al. We characterize maintenance in terms of the half life of the systems, which essentially measures the time for replacement of half of the nodes by new arrivals. *We show both analytically and via simulations that communication overhead of long-range connection maintenance per node and per system half life is O(*log *n),* where n is the size of the network. The overhead of the maintenance algorithm proposed in [61] was $O(\log^2 n)$. Furthermore we point out that maintenance overhead cannot be further decreased since this is the theoretical lower bound for maintenance traffic to ensure connectivity of the network [61].

Maintenance of a DHT storage subsystems is more difficult and resource expensive than the maintenance of a routing subsystem. Arrival and departure of nodes imply repartitioning of the key space of neighbouring nodes which in turn requires moving the affected data items from one node to another. Additionally, redundant storage is required to provide fault tolerance in dynamic environments. The resulting data replication generates significant additional maintenance overhead at high churn rate.

Here we do not consider maintenance of DHT storage subsystem. Nevertheless, we show that many applications do not even need storage functionality. In many cases, a piece of data needs to be available only when the node hosting this data is part of the network. For example it does not make sense storing a pointer to a service or resource hosted by a particular node after the given node has left the network. The same applies to the lookup of the locator of a node or to lookup the locator of the node at which a user is currently available. In these cases, it makes sense creating a separate virtual node in the DHT for each of these data items so that the ID of this virtual node is generated as the hash of the data item. Hence they can be looked up using only the routing subsystem of the DHT. If the number of data items per nodes is small (this is typically the case for the above examples), maintenance overhead using such a lookup network with the proposed stochastic maintenance mechanism will be much smaller that using a complete DHT. *Hence distributed service and resource registries and lookup of users in P2P instant messaging systems are potential application areas for the proposed stochastic maintenance model in*

dynamic network environments. Furthermore, this lookup mechanism could also be used as a distributed address lookup system for the Host Identity Protocol [115].

The rest of the appendix is structured as follows: Section A5.2 presents related works giving an overview of various resilience issues and maintenance mechanisms in P2P lookup networks. Section A5.3 describes the network model that the proposed maintenance mechanism is based on and presents the transformed Poisson process model for long-range connections. Section A5.4 presents the proposed maintenance mechanisms while Section A5.5 evaluates communication overhead of long-range connection maintenance both analytically and via simulation results. Finally, Section A5.6 concludes this part.

A6.2 Related work

Almost all P2P routing systems include some maintenance mechanisms ensuring consistency of the network as nodes join, leave or fail. However the degree of resilience and robustness they provide in different dynamic network scenarios varies in a very broad range. The following overview focuses on two important properties of these systems: static resilience and self-stabilization. Static resilience refers to fault tolerance of the network without maintenance mechanisms or before completion of maintenance. Self-stabilization in contrast characterizes maintenance algorithms. A system is self-stabilizing if maintenance algorithms ensure that it always converges to a consistent state as the network evolves.

A6.2.1 Static resilience

An interesting evaluation of DHT routing mechanisms can be found in [116] examining static resilience from a routing geometry point of view. The authors argue that flexibility in neighbour selection and route selection is a key issue that geometries should allow in order to provide resilience. Comparing the tree, ring, hypercube, butterfly and xor topologies, the ring topology is selected as the most flexible one from both points of view.

Chord [111] was one of the very first DHT routing algorithms inspiring a dozen of other P2P routing protocols based on the ring geometry. Although neighbour selection in the original Chord algorithm is deterministic, several variations have been published allowing flexible (probabilistic) neighbour selection. Most of these probabilistic variations are based on the "small worlds" network model proposed by Kleinberg in [60].

In [110], Aspnes et al. present a P2P routing mechanism based on the one dimensional version of the Kleinberg model. They provide analytically upper and lower bounds on delivery time in static networks and examine the effect of node and link failures on expected delivery time. Their results are based on the assumption that connections to the nearest neighbours are always present and long-range connections follow an inverse power law distance distribution. However, their model does not include any maintenance algorithm to restore validity of these assumptions after failures in the network. This explains the mismatch between good analytical findings and poor simulation results presented in Section 4.3.2 by the authors. To reduce the number of failed searches caused by missing short-range connections, a backtracking algorithm is incorporated into the greedy routing mechanism.

Symphony [109] is an other P2P routing protocol based on the Kleinberg model proposed by Manku et al. The authors identified that short-range links are critical for successful lookups and propose additional backup short-range links to increase resilience against failures. However, similarly to [110] this only provides static resilience since they do not describe any maintenance algorithm to restore short-range links after failures.

An interesting aspect of resilience against network partitioning is presented in [118]. Loguinov et al. propose ODRI (Optimal Diameter Routing Infrastructure) based on de

Bruijn graphs and focusing on edge bisection width, demonstrate that de Bruijn graphs are several times more difficult to disconnect than other structures. However, preventing partitioning is not sufficient to provide fault tolerant routing and the paper does not provide details on how to select alternative routes.

Finally, the authors in [117] examine analytically the resilience of various DHT routing geometries. They define routability in function of network size and failure rate as the proportion of node pairs that can reach each other. Assuming a fully populated identifier space, they calculate routability using a Markov chain model and say that a P2P routing is scalable if and only if its routability converges to a nonzero value as the system size goes to infinity for a nonzero failure probability.

A6.2.2 Self-stabilization

Almost all P2P routing systems include some kind of maintenance mechanism ensuring consistency of the network as nodes join, leave or fail (Chord [111], CAN [114], etc.). However, most of these algorithms assume that the system starts from an ideal state and returns to that ideal state after each failure [61]. Accumulation of failures in the network may eventually result in states that the protocol is unable to recover from.

Liben-Nowell et al. take an important step towards adaptation to dynamic network environments by considering a P2P network as a continuously evolving system instead of the above quasi-static approach. In [61], they introduce a new metric to describe performance of maintenance algorithms in P2P systems: the rate at which each node must expand resources in the maintenance protocol in terms of the half life of the systems (which essentially measures the time for replacement of half of the nodes by new arrival). Using this metric, they give a lower bound on maintenance protocol bandwidth needed to keep the network connected as nodes join and leave. They also present a modification of maintenance algorithm in Chord approaching this lower bound within a logarithmical factor.

In [120], Jelasity et al. propose T-man, a gossip based protocol to create a large class of different overlay topologies. Starting from a random network, T-man gradually converges to the desired topology by organizing peers according to a ranking function. However, T-man requires a synchronization point when starting the protocol and the initial topology has to be close to a random graph.

The Ring Network protocol (RN) [121] goes one step further for the special case of ring topology. RN is self-stabilizing because it converges to a Chord-like ring topology starting from any weakly connected bootstrap system and the protocol maintains this structure in face of peers joining, leaving or failing. The protocol is completely asynchronous; it does not require any synchronization between peers. However, RN does not exploit the inherent neighbour selection flexibility of the ring geometry [116]: Not only short-range, but long-range connections are also deterministic resulting in several times higher maintenance traffic.

Maintenance timing is also an important issue. In some protocols, nodes periodically ping their peers to detect node failures and/or periodically run a maintenance process to restore consistency of the network. For instance in Chord [111], nodes periodically invoke *stabilize()* and *fix_fingers()* functions to maintain consistency of successor and finger pointers in the network. Other protocols like Kademlia [112] take a reactive approach and detect failures only when a peer is unreachable during normal communication. In these cases a maintenance mechanism is triggered by these failures instead of a periodic execution. The reactive approach has the advantage of saving a lot of unnecessary maintenance traffic but in case of very low normal lookup traffic it may lead to disconnection of the network.

A6.3 Network model

In this section, we describe the network model chosen for the proposed stochastic maintenance using the terminology and reference model of [122]. Node identifiers are mapped onto a one dimensional Euclidian metric space. Similarly to Chord [111] and its derivates, the one dimensional metric space wraps around, thus nodes of the network can be represented along a ring (see Figure 79). We opted for bidirectional connections and bidirectional routing in the network. As a consequence, we define logical distance of two nodes as their smallest distance along that ring. To have logical distances among nodes in the range [0, 1], the circumference of the ring is 2 units, starting from -1 and wrapping around at the same point at 1.

The choice of bidirectional connections and routing has been motivated by the nature of physical connections, security issues, maintenance efficiency and simplicity. Communication between nodes in packet switched networks is mostly bidirectional since messages has to be acknowledged. Hence at a transport level, connections will be mostly bidirectional even if they are not at an application level. Furthermore, in distributed systems where connections are unidirectional, it is much more difficult to impose incentives for cooperation than in systems using bidirectional connections. However, the strongest arguments for bidirectional connections will be presented in Section 0, where we conclude that using stochastic maintenance, most of the connections deleted as a result of nodes failing or leaving can be automatically replaced by incoming connections of arriving nodes – assuming that connections are bidirectional.

Figure 79 shows connections of a node on the logical ring representing the one dimensional metric space. Each node has a fixed number ($2N_S$) of short-range connections (Figure 79 shows an example of N_S=3) to the N_S closest neighbours in both directions. Furthermore each node has a variable number (N_L + $N_{L'}$) of long-range connections labelled in decreasing distance order at both to the left and right sides of the node. *While short range-connections are deterministic, long-range connections only have to satisfy probabilistic requirements: the probability of having a long-range connection between two nodes inversely proportional to their distance.*

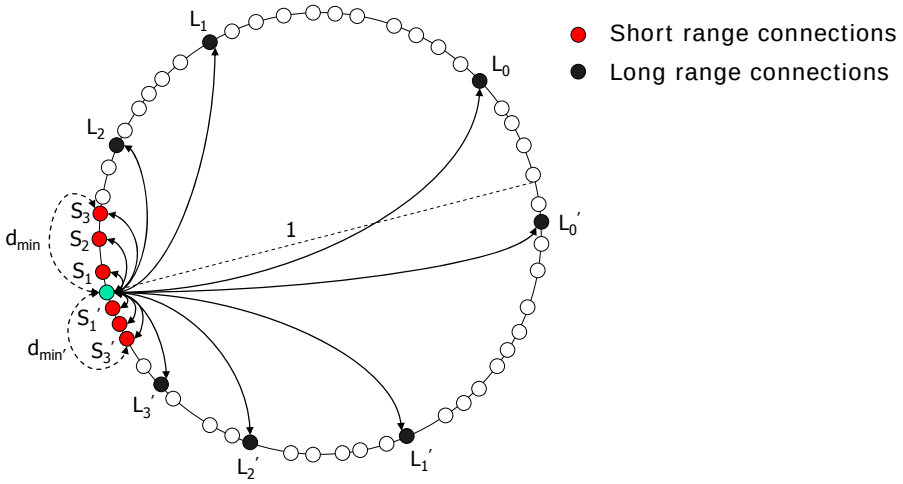


Figure 79 - Lookup network model

Some P2P routing protocols (e.g. in Symphony [109] or De Bruijn graph based networks) use only a fixed number of long-range connections. However this approach has a number of disadvantages. Routing performance in fixed degree networks do not scale logarithmically with network size. This is not an important problem in static networks, since the achieved polylogarithmic scalability is also acceptable in most practical cases. However, logarithmic scalability also plays an important role in self-adaptation of the network. For example, in Section A6.3.1, we show that in logarithmically scalable networks,

it is possible to define by transformation a constant connection density independent of the network size. This allows easy self-adaptation of the number of connections to the network size. In fixed degree networks, network size has to be estimated by a separate protocol before joining the network. Other interesting aspects on inflexibility of fixed degree P2P routing networks can be found in [123].

In the following subsections, we first introduce a stochastic model describing long-range connections of a node then we describe operation of the network using this model.

A6.3.1 Stochastic model for long-range connections

To describe analytically the maintenance process of long-range connection, we introduce a mathematical model based on the transformation of stationary Poisson processes. We consider the sequence of long range connections in both directions as a Yi stochastic process. Arrival times of the left process correspond to the sequence $L_0 - L_{N_L}$ while in the right process they correspond to the sequence $L'_0 - L'_{N'_L}$. Both processes start from the point being the furthest from the node on the ring.

Let X_i be a stationary Poisson process of rate λ given by the sequence of arrival times. We derive from X_i the stochastic process Y_i by transforming each arrival time x_i into an arrival time y_i so that $y_i = e^{-x_i}$ (see Figure 80). This transformation is a bijection; there exists a one-to-one correspondence between the two processes. Exploiting this property, it is possible to derive all properties of long-range link distribution by transforming well-known properties of a stationary Poisson process.

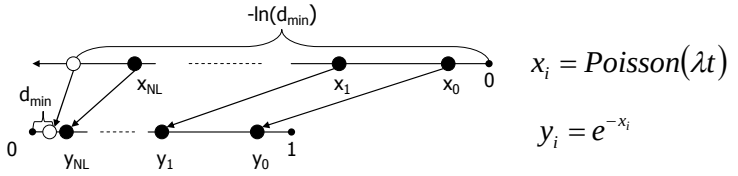


Figure 80 – Generation of stochastic process corresponding to the sequence of long-range connections

Both the original stationary Poisson process X_i and the derived stochastic process Y_i are defined in a distance domain (this latter corresponds to the one dimensional Euclidian metric space that the network embedding the network). On the analogy of arrival rate in the time domain (as the most widespread application of Poisson processes) this Poisson process-based model allows definition of connection density as the rate λ of the original Poisson process. As we will show later, this connection density unequivocally determines lookup performance and plays an important role in maintenance algorithms.

The stochastic process Y_i ends with the last short-range connection (shown as an empty circle in Figure 80) at a distance d_{min} from the own ID of the node. This point maps to a distance $-\ln(d_{min})$ in the X_i Poisson process. Since short range connections are deterministic, d_{min} is also deterministic for a specific node of a given network. As a consequence, the number of long-range connections (at one side of the ring) follows a Poisson distribution:

$$P(N_L = k) = \frac{(-\ln(d_{\min})\lambda)^k}{k!} e^{\lambda \ln(d_{\min})} = \frac{(-\ln(d_{\min})\lambda)^k}{k!} {d_{\min}}^{\lambda}$$

However, d_{min} itself is also a random variable depending on the distribution of node identifiers in the metric space, thus the above formula is only valid for a specific node in a given network.

In the followings, we show that the derived Y_i stochastic process exhibits an inverse power law distance distribution required for logarithmically scalable routing (Theorem I.); hence it can be used to describe the sequence of long-range connections. Then we derive the distribution of inter-arrival time of the stochastic process Y_i (Theorem II.) to enable probabilistic creation of initial long-range connection.

Theorem I.: The probability of having a connection in the range *[y-dy, y]* is $\frac{\lambda}{y}dy$

Proof: Since x is a Poisson process of rate *λ*, the probability of having an arrival in the infinitesimally small range *[x, x+dx] is λdx*. As a consequence of the bijection property of the transformation, the probability of having an arrival in the corresponding range *[y-dy, y]* of the transformed process Y_i is the same. In the transformed process, $y = e^{-x}$ and $dy = y'(x)dx = -e^{-x}dx$ thus we get $dx = -\frac{dy}{y}$. Hence the probability of having a connection in the range *[y-dy, y]* is proportional to the arrival rate of the original process and inversely proportional to the distance.

Theorem II.: The logarithm of the ratio of consecutive long-range connections follows an exponential distribution of parameter λ

Proof: Using the transformation $y_i = e^{-x_i}$, we get $\ln\left(\frac{y_i}{y_{i+1}}\right) = \ln\left(\frac{e^{-x_i}}{e^{-x_{i+1}}}\right) = (x_{i+1} - x_i).$

Hence the logarithm of distance ratios in the stochastic process Y_i corresponds to the inter-arrival times of the original Poisson process. Since inter-arrival times of a stationary Poisson process of rate *λ* follow an exponential distribution of parameter *λ*, $\ln\left(\frac{y_{i+1}}{y_i}\right)$ will be also exponentially distributed with parameter *λ.*

A6.3.2 Lookup

Similarly to most P2P routing algorithms lookup process is based on a very simple greedy algorithm. Lookup request is forwarded to the peer being the closest to the target in the metric space until getting to one of the peers of the target (see pseudo code description of the algorithm in Figure 81). Since connections are bidirectional, forwarding is possible in both directions, but the next hop node must always be closer to the target. This condition prevents infinite forwarding loops when the target of the lookup is unreachable.

```
lookup(Node target) {
 if (target ∈ peers) return peersTable.getAddress(target);
 Node proxy = min_{x∈peers} d(x, target);
 if (d(proxy, target) > d(this, target)) return failure;
 else return proxy.lookup(target);
}
```

Figure 81 – Simple greedy lookup algorithm

If the request cannot be routed closer to the target, then the lookup fails. Assuming that each node is connected to its closest neighbours on the ring in both directions, this can never happen, since forwarding the request to one of these two neighbours, it is always

possible to get closer to the target. Hence the role of short-range connections is to guarantee success of lookups.

Although short-range connections guarantee lookup success, forwarding exclusively through short-range links would result in O(N) time and message complexity for lookups. The role of long-range links is to decrease the distance from the target every step by a constant factor in expected value and thus guarantee O(log N) expected value of hop counts for lookups. In Theorem III, we prove that the rate at which consecutive lookups approach the target is independent of the distances and that the expected value of distances (*d0, d1, …, dN)* diminish exponentially with a constant average ratio (*d1/d0=d2/d2=di/di-1=…*). Transforming this logarithmically back to the Poisson process model of long-range connection, search process advances towards the target in this transformed metric space at a fixed speed (in expected value).

Theorem III.: Let λ be the density of long-range connections and d_i and d_{i+1} be the distance from the target before and after the i^{th} lookup step. Then $E\left(\frac{d_{i+1}}{d_i}\right) \le \frac{1}{1+\lambda}$

Proof: Let p_B be the long-range peer right before the target and p_A the long-range peer right after the target on the ring. We first show that $E\left(\frac{d_{i+1}}{d_i}\right) = \frac{1}{1+\lambda}$ holds if only clockwise or counter clockwise lookups are used (so that it is not possible "overshooting" the target even if this would decrease the absolute distance). Based on this, the inequality follows trivially if both directions can be used; the given value is hence a rough (probabilistic) upper bound where the actual values could be considerable smaller.

Let *t* be the target of the lookup and d_i the distance from this target on the ring before the i^{th} lookup step. We are looking for the distribution of the random variable $Z(d_i)$ corresponding the distance from the target after the i^{th} lookup step. We calculate distributions of $Z(d_i)$ using the Poisson model introduced in Section A6.3. Starting from the stochastic process Y_i corresponding to long-range connections, the target and the two connections being the closest to the target are transformed back to the original Poisson process (see Figure 82).

Let $\hat{Z}$ be the random variable corresponding to distance from the target to the closest arrival in X_i. Since X_i is a Poisson process, the pdf of $\hat{Z}$ is $f(\hat{z}) = \lambda e^{-\lambda \hat{z}}$ in both directions, independently of the position of the transformed target *t*. The random variable $Z(d_i)$ is a function of $\hat{Z}$ thus its pdf g(z) can be expressed by transforming $f(\hat{z})$.

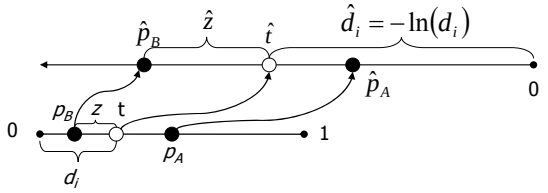


Figure 82 - Derive pdf of distance distribution after the i^{th} lookup step

It is known that given a random variable $\hat{Z}$ with a probability density function $f(\hat{z})$ and a random variable Z = ψ($\hat{Z}$) where ψ is a strictly monotone increasing function, the pdf g(z) of Z can be expressed as:

$$g(z) = f\left(\Psi^{-1}(z)\right)\left|\frac{d\Psi^{-1}(z)}{dz}\right|$$

According to the definition of transformation between X_i and Y_i:

$$d_i - z = \frac{1}{e^{-\ln(d_i)+\hat{z}}} \rightarrow z = \Psi(\hat{z}) = d_i\left(1 - e^{-\hat{z}}\right)$$

$$\hat{z} = \Psi^{-1}(z) = -\ln\left(\frac{d_i - z}{d_i}\right) \text{ and } \frac{d\Psi^{-1}(z)}{dz} = \frac{1}{d_i - z}$$

Hence $g(z) = f\left(\Psi^{-1}(z)\right)\left|\frac{d\Psi^{-1}(z)}{dz}\right| = \lambda e^{-\lambda\left(-\ln\left(\frac{d_i - z}{d_i}\right)\right)} \frac{1}{d_i - z} = \lambda\left(\frac{d_i - z}{d_i}\right)^{\lambda} \frac{1}{d_i - z}$

Calculating the expected value of $Z(d_i)$:

$$E\left(Z\left(d_i\right)\right) = \int_0^{d_i} g(z) z dz = \int_0^{d_i} \lambda\left(\frac{d_i - z}{d_i}\right)^{\lambda-1} \frac{z}{d_i} dz = \frac{d_i}{1+\lambda}$$

$$E\left(\frac{d_{i+1}}{d_i}\right) = \frac{E\left(Z\left(d_i\right)\right)}{d_i} = \frac{1}{1+\lambda}$$ as stated in the theorem.

A6.4 Maintenance

Maintenance mechanisms exploit the fundamental difference between the role and importance of short-range and long-range connections. While short-range links are critical for the success of lookup operations, there are only loose probabilistic requirements towards long-range links. Therefore, a strict self-stabilising maintenance algorithm is proposed for short-range connections and a loose probabilistic connection maintenance algorithm for long-range connections.

A6.4.1 Maintenance of long-range connections

By *probabilistic maintenance* we mean the weak enforcement of the required power law distance distribution on long-range connections. We show that each of our maintenance steps preserves power law distribution, where only the distribution parameter – corresponding to the density of long-range connections – changes as the system evolves. Our objective during the maintenance is to restrict the density of long range connections to a predefined range.

Additionally, we propose an algorithm, which creates the initial long-range connections in according to the desired power law distribution with only O(1) communication overhead per connection.

Generating initial long-range connections

One way of creating the initial link distance distribution is to pick up random points in the metric space according to that distribution and create connections to nodes being the closest to these points (this is the approach taken in Symphony [109]). Since the creation of each connection involves a normal search operation, this would require O(log N) messages per connections.

We propose another approach: connections are created in a decreasing distance order starting from the furthest ID and approaching the ID of the node from both directions. Choosing distance ratios so that their logarithm follows an exponential distribution results in the same distance distribution according to Theorem II.

Furthermore, the connection is not necessarily established to the node closest to the drawn ID but to some small range of it. Hence, the first (fastest) lookup hit (ID) matching the

determined range will be used. Figure 83 shows how this range is defined. Let d_i be the distance from the i[th] long-range peer and d_{i+1} the distance drawn to create the next long-range connection. Then we define a range $[d_i/c, d_i c]$ where c = 1 + ε (constant system parameter). If $d_i c > d_i$ then we truncate the range at d_i to ensure that the next long-range connection is always closer in the metric space.

It can be shown that finding an arbitrary node in this range take only *O(1)* steps. When searching for the ID at d_{i+1} distance, it is guaranteed to reach a node within that range after reducing the original distance by a factor of $\frac{d_{i+1} - d_{i+1}/c}{d_{i+1}} = \frac{c-1}{c} = \frac{\varepsilon}{1+\varepsilon}$. Since this is a constant, and according to Theorem III, distance to the target diminishes at each lookup step by a constant factor in expected value, finding a node in the given range takes only O(1) steps.

Simulation results in Section A6.5.2 show that the small bias caused by picking up an arbitrary node from the above defined range instead of the closest node to the drawn ID does not affect considerably the distance distribution of long-range connections. Furthermore this flexibility in the choice of connections also opens up the possibility of taking into account network locality by selecting the physically closest node from the first few nodes found in that range.

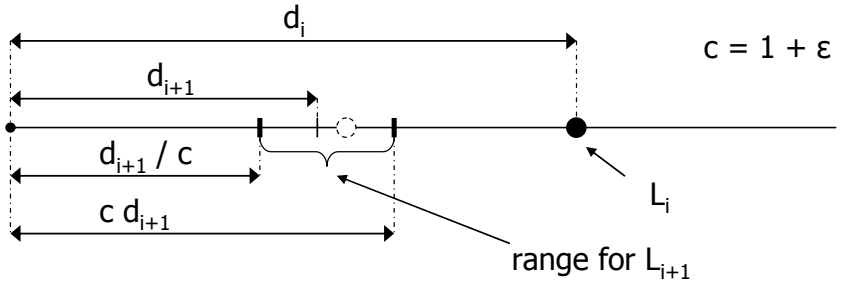


Figure 83 - Defining range for initial long-range connections

Sequential creation of long-range connection also has the advantage of self-adapting the number of long-range connections to the size of the network; there is no need for an estimation protocol as in Symphony [109]. Establishment of long-range connections is completed when no more peers are found in the given next closer range. Visually, using the notations of Figure 80, this happens when the generator Poisson process goes beyond the point $-\ln(d_{min})$. As a final step, the join process is completed by finding and establishing short-range connection.

Maintaining long-range connection distribution

Assuming that all nodes use the above algorithm to create their long-range connections when joining the network, it can be shown that the type of the distribution of long range connections will not be altered at other nodes of the network. Inverse transforming long-range connections as shown in Section A6.3.1, the sequence of long-range connections created during the join process can be described by a Poisson process of rate λ. Assuming that nodes are evenly distributed in the ID space, new incoming connections can be considered as the superposition of another Poisson process (with a rate proportional to the arrival rate of new nodes) to the Poisson process describing initial long-range connections of the node. Similarly, deletion of connections as a result of nodes failing or leaving the network can be considered as random selection from a Poisson process. Both superposition of two Poisson process and random selection from a Poisson process results also in a Poisson process (of different rate) thus network dynamism does not affect the inverse power law nature of long-range connections.

Although distribution type is not affected, the distribution parameter describing density of long-range connection does change as the network evolves. Maintenance mechanisms should keep connection density within a given range for two reasons: On the one hand, the average number of lookup hops is a strictly monotone decreasing function of connection density, thus a lower bound on connection density is necessary to provide an upper bound on the average lookup hop count. On the other hand, node degree is proportional to connection density thus a lower bound on connection density is needed to have an upper bound on node degree.

Connection density corresponds to the λ rate of the generator Poisson process. Since the rate of a Poisson process can be estimated as the number of arrivals per interval units, connection density can be estimated from the number of connections and the length of the process: $\lambda = \frac{N_L}{\ln(d_{\min})}$ .We define an upper and lower bound for connection density: $\lambda_{min}<\lambda_{opt}<\lambda_{max}$ and ensure that the connection density be always within this range.

If the estimated connection density falls below λ_{min}, then new connections are created until $\lambda > \lambda_{opt}$. Since the network size can be estimated from d_{min}, new connections can be directly created by generating random identifiers in the range $[d_{min}, 1]$ according to an appropriate distribution. The probability of having a connection between two nodes is inversely proportional to their distance (see Theorem I.), hence the pdf used to generate these identifiers should have the form of $f(x) = \frac{c}{x}$. To satisfy the distribution criteria: $\int_{d_{\min}}^{1} \frac{c}{x} dx = 1$ giving $c = -\frac{1}{\ln(d_{\min})}$. Similarly to the creation of initial long-range connection, maintenance process creates new connections around a given small range of the drawn identifier to ensure O(1) maintenance overhead per connection creation.

If the estimated connection density exceeds λ_{max} then some connections are deleted until $\lambda < \lambda_{opt}$. It is important that selection of connections to delete should be independent from their distance. Random deletion of arrivals from a Poisson process results in another Poisson process of smaller rate as long as the selection is unbiased.

A6.4.2 Maintenance of short-range connections

Having connections to the closest peers on the ring in both directions is critical to guarantee successful lookups. Maintaining a small S_N number of short-range connections instead of only one increases static resilience but cannot replace maintenance mechanisms. Therefore short-range connection maintenance is proactive; each node periodically sends echo messages to a randomly selected peer of its short-range peers. Short range relations are bidirectional, hence echoing ensures failure detection in bounded time. This mechanism is very similar to leaf set maintenance in Bamboo[25] [124], and the self-stabilizing maintenance algorithm presented in [121]. In brief, exchanging leaf sets, nodes can detect if they have missing connections and searching for closest nodes, they can create missing short-range connections.

[25] Bamboo is a derivate of Pastry, and leaf set correspond to the list of short-range connections

Since the number of short-range connections is fixed, per node short-range connection maintenance overhead does not increase as system's size grows. However, it is challenging to adapt dynamically the period for echo messages to churn rate.

A6.5 Evaluation

In this section, we first analytically evaluate the long-range connection density distribution in a network under churn. Using a Markov chain model, we derive the communication overhead of long-range connection maintenance for a steady state. Next we support these findings by simulation results and finally, we compare qualitatively the proposed maintenance model to existing DHT routing solutions.

A6.5.1 Analysis of maintenance traffic in a network under churn

As described in Section A6.4.1, long-range connection maintenance is based on keeping long-range connection density within a lower and upper threshold. To analyse maintenance traffic, we first need to introduce a few variables describing long-range connections and change rates.

Let $\lambda_{min} = \lambda_{opt}-\Delta\lambda$ and $\lambda_{max} = \lambda_{opt}+\Delta\lambda$, thus long-range connection density can vary within the range $[\lambda_{opt}-\Delta\lambda, \lambda_{opt}+\Delta\lambda]$. Furthermore, as a first approximation, let us assume that the d_{min} distance from the furthest short-range connection is the same for all nodes, hence the number of long range connections only depends on long-range connection density: $N = 2 N_L = -2 \lambda \ln(d_{min}) = c \lambda$, where c is a constant. Let now define $N_{opt} = c\lambda_{opt}$, $\Delta N = c \Delta\lambda$, $N_{max} = N_{opt} + \Delta N$ and finally $N_{min} = N_{opt} – \Delta N$. If N drops below N_{min}, then new connections are created until reaching N_{opt}. If N exceeds N_{max}, then connections are deleted until reaching again N_{opt}.

We assume that the arrival of new nodes as well as the departure or failure of nodes can be described by a Poisson process. As a consequence, connection establishment and deletion will be also described by a Poisson process. Let r_{in} be the arrival rate and r_{out} be the departure rate of nodes in a system under churn. We assume a worst case scenario where departing nodes either fail or depart ungracefully. Let R_c be the creation rate of new long-range connections and R_d the deletion rate of existing long-range connections in the system (hereafter, we refer to long-range connections simply as connections in this section). We define all of the above rates normalized to the size of the network; for example R_c denotes the number of new connections created per nodes and per time units.

Both R_c and R_d can be decomposed into two components. New connections are created on the one hand by new nodes joining the network and on the other hand by nodes whose connection density drops below the lower threshold λ_{min}. Let us denote by R_{cm} the rate of connection establishment resulting from exceeding the lower threshold λ_{min}. Hence

$$R_C = N_{opt} r_{in} + R_{cm} \quad (1)$$

Similarly, connection deletion occurs when a node detects failure of a connection or when a node deletes connections after exceeding the upper threshold λ_{max} of its connection density. The rate of connection deletion resulting from failure of nodes cannot be expressed directly from the failure rate. This is a consequence of the applied lazy failure detection mechanism: a failed link is detected only by timeouts during normal communication. Assuming that the probability of selecting a "dead" link for message forwarding equals to the ratio of "dead" links in the network, failure detection rate $R_f = c_{dead} R_{all}$, where c_{dead} is the ratio of "dead" links in the network and R_{all} is the overall communication rate over long-range connections. Let us denote by R_{dm} the other component corresponding to the rate of connection deletion resulting from exceeding the upper threshold λ_{max}. Then R_d can be expressed as

$$R_D = c_{dead} R_{all} + R_{dm} \quad (2)$$

Both R_{cm} and R_{dm} can be expressed by R_c, R_d and the probability of having a connection number corresponding to the upper and lower level connection density thresholds:

$$R_{dm} = (\Delta N + 1) P(N = N_{\max}) R_C \quad (3)$$

$$R_{cm} = (\Delta N + 1) P(N = N_{\min}) R_D \quad (4)$$

Replacing (1) and (2) into (3) and (4) we obtain:

$$R_{dm} = (\Delta N + 1) P(N = N_{\max}) (N_{opt} r_{in} + R_{cm}) \quad (5)$$

$$R_{cm} = (\Delta N + 1) P(N = N_{\min}) (c_{dead} R_{all} + R_{dm}) \quad (6)$$

c_{dead} and the two probabilities in the above equation system depend on the history of the network. Furthermore, besides R_{cm} and R_{dm}, there is a third independent variable (R_{all}) depending on many factors in a complex way. Hence it is difficult to provide a general solution. However, in the followings, we show that the above equation system can be solved for steady states when the arrival rate and failure rate of nodes is the same ($r_{in} = r_{out} = r$). First, we calculate the $P(N = N_{max})$ and $P(N = N_{min})$ probabilities then we derive the ratio of dead links in the network for this steady state.

If we consider the number of long-range connections as a random variable, we can describe the evolution of this random variable by a stationary Markov chain. The transition matrix ***P*** of the resulting Markov chain is show in Figure 84, (the element $x_{i,j}$ of the matrix corresponds to the transmission probability to the state *j* during an infinitesimally small *dt* time, assuming that the current state is *i*).

	N_{min}	N_{min} +1	...	N_{opt}	…	N_{max} -1	N_{max}
N_{min}	1- (R_c+R_d)dt	R_c dt	…	R_d dt	…	0	0
N_{min} +1	R_d dt	1- (R_c+R_d)dt	…	0	…	0	0
...	…	…	…	…	…	…	…
N_{opt}	0	0	…	1- (R_c+R_d)dt	…	0	0
…	…	…	…	…	…	…	…
N_{max} -1	0	0	…	0	…	1- (R_c+R_d)dt	R_c dt
N_{max}	0	0	…	R_c dt	…	R_d dt	1- (R_c+R_d)dt

Figure 84 - Transition matrix of the Markov chain describing evolution of the number of long-range connections

In steady state, $R_c = R_d = R$. Let π be the vector describing stationary distribution of the random variable corresponding to the number of long-range connections. Then π should satisfy the following equation: $\pi = \pi \boldsymbol{P}$. As a consequence π can be expressed as a left eigenvector of the transition matrix associated with the eigenvalue 1. The above transition matrix has only one left eigenvector associated to the eigenvalue 1 and independent of the size, it is:

$$\left\{ \frac{1}{(\Delta N+1)^2}, \frac{2}{(\Delta N+1)^2}, \ldots, \frac{\Delta N+1}{(\Delta N+1)^2}, \ldots, \frac{2}{(\Delta N+1)^2}, \frac{1}{(\Delta N+1)^2} \right\} \quad (7)$$

Thus $P(N= N_{min}) = P(N= N_{max}) = 1/(\Delta N+1)^2$. (8)

A steady state also implies that c_{dead} is constant, hence the rate at which dead links are created and deleted are equal. A dead link is either deleted when a node detects it via

timeouts or simply disappears from the system when a node fails or leaves the network. Hence $N_{opt} r_{out} = c_{dead} R_{all} + N_{opt} r_{out} c_{dead}$. Replacing these results into (5) and (6), we obtain the following equations:

$$R_{dm} = \frac{(\Delta N + 1)}{(\Delta N + 1)^2} \left(N_{opt} r + R_{cm} \right) \quad (9)$$

$$R_{cm} = \frac{(\Delta N + 1)}{(\Delta N + 1)^2} \left(N_{opt} r (1 - c_{dead}) + R_{dm} \right) \quad (10)$$

Solving the equation system, we obtain

$$R_{dm} = \frac{N_{opt}}{\Delta N} \frac{\Delta N + 2 - c_{dead}}{\Delta N + 2} r \le \frac{N_{opt}}{\Delta N} r = \frac{\lambda_{opt}}{\Delta \lambda} r \quad (11)$$

$$R_{cm} = \frac{N_{opt}}{\Delta N} \frac{\Delta N + 2 - c_{dead}(\Delta N + 1)}{\Delta N + 2} r \le \frac{N_{opt}}{\Delta N} r = \frac{\lambda_{opt}}{\Delta \lambda} r \quad (12)$$

Before evaluating these results, let us recapitulate the notion and background of different maintenance rates used in the above analysis. Our analysis focuses on maintenance in steady state when $r_{in} = r_{out} = r$. All of these rates are normalized by network size.

We have defined R_C as the number of new long-range connections created per nodes and per time units. R_C has two components: the first one is the rate of connection establishment due to connections created by new nodes when they join the network ($N_{opt}\, r$) while the second one corresponds to the rate at which long-range connection maintenance creates additional links ($R_{cm} \le \frac{\lambda_{opt}}{\Delta \lambda} r$). N_{opt} increases logarithmically with network size while R_{cm} does not increase with system size since λ_{opt} and $\Delta\lambda$ are constants. The creation of a new connection involves a search to a range with O(1) communication overhead (see section 0) and a connection establishment with also O(1) overhead.

We have defined R_D as the number of long-range connections deleted per nodes and per time units. R_D has also two components: the first one is the rate at which dead links are discovered during normal operation $R_f = \frac{N_{opt} r}{R_{all} + N_{opt} r} R_{all}$ while the second one corresponds to the rate at which maintenance process deletes existing connections ($R_{dm} \le \frac{\lambda_{opt}}{\Delta \lambda} r$). When detecting a dead link, the corresponding message has to be forwarded via another long-range peer (O(1) overhead) while deleting a connection only involves sending one disconnect message (O(1) overhead again).

Creation and deletion rate of long-range connections as well as the associated communication overhead is summarized in Table 8 (*n* is the number of nodes in the network). Let us define time unit for churn rate as the system's half life defined in [61] (system half life measures the time during which half of the nodes in the network are replaced by new arrivals). Using this definition, the *per node maintenance traffic of long-range connections per half life is upper bounded by* O(log *n*) *in our system*. This result is very important because the authors in [61] show that this is in fact the theoretical lower bound of maintenance traffic to ensure that the network remains connected.

	Connection creation/deletion rate	Communication overhead

New nodes joining	O(log n) r	O(log n) r
Create by maintenance	O(1) r	O(1) r
Detect failed connections	O(log n) r	O(log n) r
Delete by maintenance	O(1) r	O(1) r

Table 8 - Maintenance overhead

Another interesting observation related to Table 8 is that most of the missing long-range connections are replaced by new connections established by joining nodes and only a few fraction need to be created explicitly by maintenance. In other words, most of the maintenance is achieved automatically as a side effect of the unavoidable maintenance traffic related the creation of new connections of joining nodes. This is a direct consequence of two important factors: bidirectional connections and stochastic maintenance.

A6.5.2 Simulation results

In this subsection, we support by simulations the analytical results for static and dynamic properties of the network.

First, we examine static properties of the network, namely the distance distribution of long-range connections, node degrees and lookup hops in the function of network size. Note that all of these static properties have been measured in a system under churn (steady state). We have used the following common parameters in all of the simulations as summarized below:

N_S = 3, hence each node maintains 3 short-range connections in both directions;

$\lambda = \frac{1}{\ln 2}$, hence long-range connection density corresponds to that of Chord (see Section A6.5.3);

$\Delta\lambda$ = 0.2 λ, thus λ_{min} = 0.8 λ and λ_{max} = 1.2 λ;

Arrival and departure of nodes is simulated by Poisson processes. The rate of these processes (churn rate) is 10% of the lookup traffic rate. Thus one node leaves and a new node join on average for every 10 lookups.

Figure 85 shows the distance distribution of long-range connections in a network of 100 000 nodes. The dotted line was drawn based on simulation results while the continuous line corresponds to the theoretical values obtained by integrating the inverse power law probability density function $P(D < d) = \frac{\ln(x / d_{\min})}{-\ln d_{\min}}$. d_{min} is the expected value of the distance from the last short-range peer, calculated from the network size as $E(d_{\min}) = 2N_S / n$ (N_S is the number of short-range connections in one direction while n is the number of nodes in the network).

Simulation results are very close to the theoretical values. Slight deviations from the theoretical values in the smaller distance range result are most probably due to the un-uniform distribution of identifiers in the metric space. Since node identifiers were drawn at random, d_{min} values are not the same for all nodes but follow a gamma distribution.

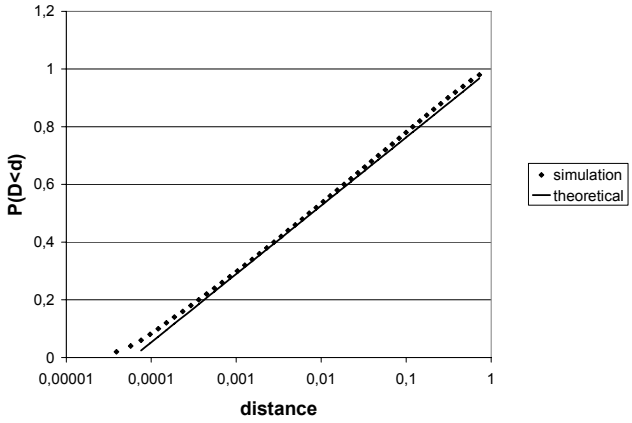


Figure 85 - Distribution function of long-range connection distance (theoretical and simulation results)

Figure 86 shows the number of lookup hops and node degree in function of network size. In both figures, the average values and the 5th and 95th percentiles are represented. In accordance with our analytical results, both variables clearly increase logarithmically with network size.

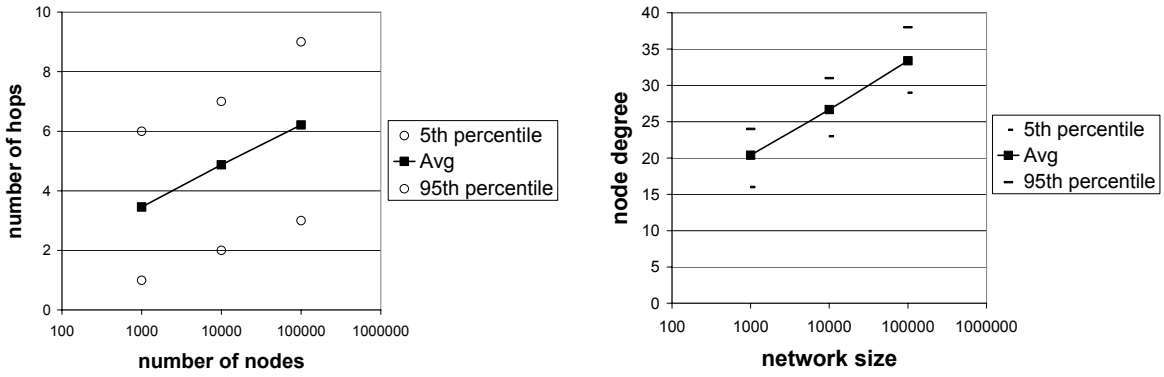


Figure 86 - Number of lookup hops and node degree in function of network size

Our major contribution is the O(log n) bound on long-range connection maintenance overhead. Figure 87 shows the number of maintenance messages per node and per system half-life in function of network size. Maintenance traffic is measured in the number of request/reply messages.

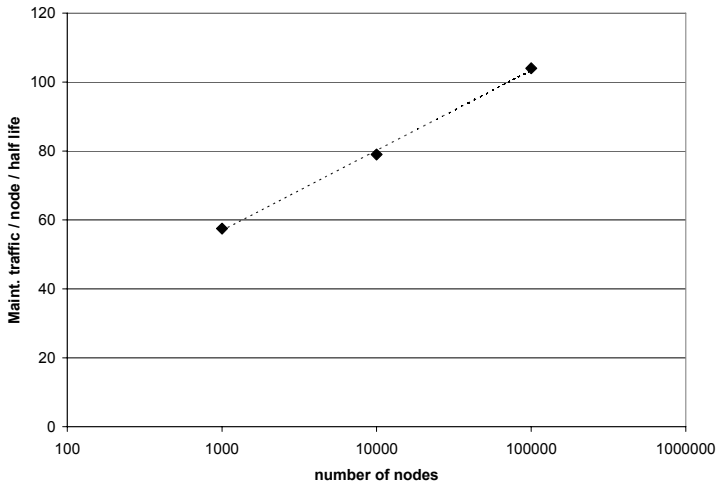


Figure 87 - Number of maintenance messages per node and per network half life

Finally, Figure 88 summarizes the impact of the $\Delta\lambda / \lambda$ - the relative size of the range for long-range connection density – on various static and dynamic network properties. Both charts are based on simulation results in a network of 10 000 nodes.

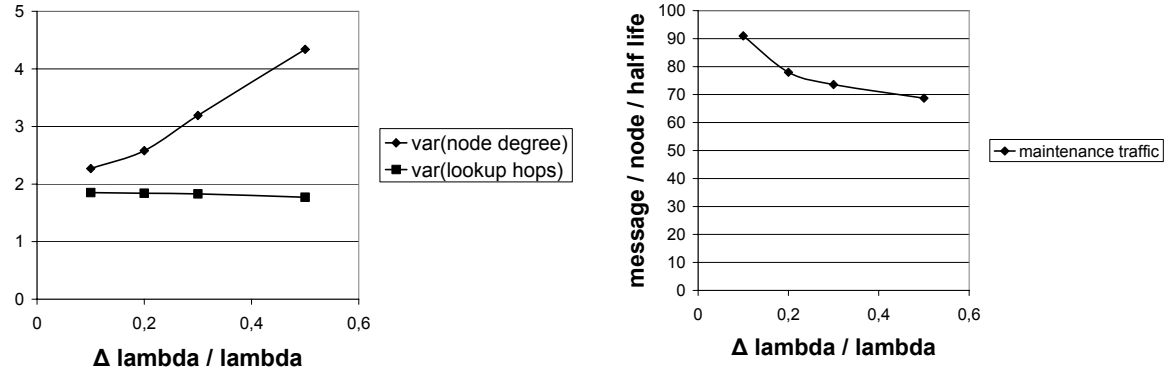


Figure 88 - Effect of $\Delta\lambda$ on maintenance overhead and variance of node degree and lookup hop count

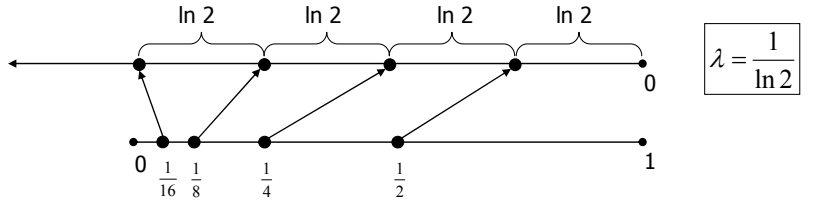
The left graph represents variance of node degree and variance of the number of lookup hops in function of $\Delta\lambda / \lambda$. Since node degree is proportional to long-range connection density, it is not surprising that variance of node degree depends near linearly of $\Delta\lambda$, determining variance of connection density. In contrast, variance of node degree does not depend on $\Delta\lambda$.

The right graph represents the number of maintenance messages per node and per system half life in function of $\Delta\lambda / \lambda$. On the on hand, the graph reflects that maintenance overhead associated with creation and deletion of long-range connections is inversely proportional to $\Delta\lambda / \lambda$ (see equations 10 and 11). On the other hand, the graph shows that this maintenance overhead accounts for only a small fraction total maintenance communication - in accordance with various maintenance overheads summarized in Table 8.

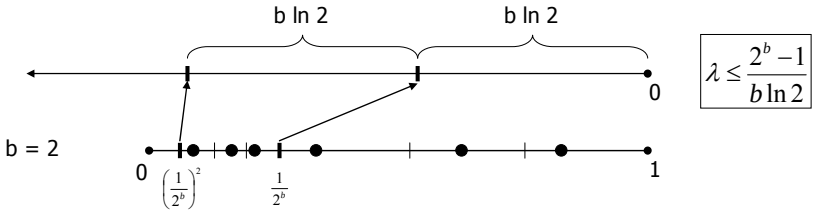
Thus we can conclude that except for very small or very large $\Delta\lambda / \lambda$ values, this parameter does not considerably influence network performance.

A6.5.3 Comparison with existing DHT routing solutions

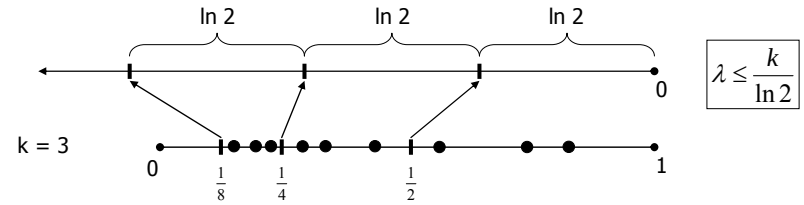
Using the transformed stochastic model introduced in Section A6.3.1, we examine the three most popular DHTs using one dimensional metric spaces for routing: Chord [111], Pastry [113] and Kademlia [112]. Figure 89 shows the original (bottom line) and the inverse transformed (upper line) distance distributions for each variant. (For Pastry, the parameter *b* is the bit length of numbers in the routing table, while the parameter *k* for Kademlia is the maximum size of buckets).



a) Chord



b) Pastry



c) Kademlia[26]

Figure 89 – Derive long-range connection density of various DHTs by inverse transformation

The transformed line shows perfectly the fundamental similarities in the distance distribution of long-range connections as well as the important differences in the degree of determinism. As mentioned in the introduction, long-range connections follow in inverse power law distance distribution in all DHT routing solutions. As a consequence, it is possible to define a long-range connection density in the transformed model independent of the distance for each of these DHTs (though in Pastry, there are slight local irregularities). Determinism in the choice of long-range connections diminishes in the order Chord → Pastry → Kademlia.

[26] Unlike other DHTs, Kademlia uses a non-Euclidean metric space where distances are calculated using a XOR metric. Hence graphical representation of distances in Figure 9/c is meaningful only between the node itself and its long-range peers, but not among different long-range peers.

Two interesting observations can be made: First, using the inverse transformation, it is possible to define a long-range connection density parameter λ for each DHT independent of the distance and network size (though for Pastry, there are slight irregularities). The second is the differences in the degree of determinism. This later is easy to be picked out in after the transformation. In Chord, each connection is deterministic. Pastry is somewhat more flexible, each routing entry may contain any node from a given range. Kademlia goes one small step further and allows the choice of any nodes (up to a maximum number of k) from a given range.

Our proposal's very low maintenance overhead could only be achieved because long-range connections in the underlying network model do not impose any of the above constraints. As shown in Section A6.3.1, we derive long-range connections from a Poisson process, being the "most random" solution ensuring the required distance distribution.

A6.6 Conclusion

We have presented a maintenance model for routing in structured P2P systems where short-range connection maintenance is proactive, deterministic and strict while long-range connection maintenance is reactive, stochastic and loose. We have shown that this dual maintenance strategy reduces maintenance overhead to the achievable theoretical lower bound without degrading routing performance.

We have considered the sequence of long-range connections of a node as a stochastic process in a one dimensional Euclidian metric space and derived this stochastic process by an exponential transformation from a stationary Poisson process. We have analytically shown that the transformed probabilistic creation of long-range connections results in a network where routing scales logarithmically with the system's size. Using the Poisson process transformation model, we have defined a long-range connection density parameter which is independent of network size. We have also shown analytically that this single parameter unequivocally determines routing performance and node degree distribution in a given network. Starting from this result, we have proposed a loose stochastic maintenance algorithm based solely on limiting at each node the deviation from a predefined connection density value.

We have described relationship between maintenance rates and churn by an equation system. Solving this equation system, we have analytically derived the maintenance overhead in steady state for a network under churn. We have described maintenance overhead in terms of system half-life, introduced in [61] corresponding to the time during which half the nodes in the network are replaced by new arrivals. We have shown that long-range connection maintenance traffic per node is upper bounded by $O(\log n)$, where n is the system's size; which is at the same time the theoretical lower bound for a system to remain connected. Finally, we have validated our analytical results by simulations for network sizes of up to 100 000 nodes.

Given the flexibility in the choice of long-range connections, our model can be potentially extended to take into account network locality. We also plan to investigate how this stochastic model and the bidirectionality of connections could be used to provide incentives for cooperation and protection against malicious nodes.

A7 Evaluation of various approaches

A7.1 Evaluation of VMB protocol

We will start off with an overview on our implementations for the standard VMB protocol; followed by an evaluation on the efficiency, scalability, and robustness of the standard VMB protocol.

A7.1.1 Implementations for the Standard VMB protocol

Due to space limitation, we will only provide an overview on the standard VMB protocol implementation here. All implementations are Java-based. We have developed a CAN-DHT simulator that creates n-dimensional CAN-DHTs of different network sizes, and generates a graphical display of the DHT keyspace ownership. This simulator is also capable of determining the overlay hop count (from one overlay node to another), and the number of immediate overlay neighbours of a particular overlay node. We have also developed a VMB node management simulator, which takes different values of the neighbourhood scale *N*, then simulates the activation of VMB node(s) on different number of overlay nodes in the DHT accordingly, and returns the total number of a particular type of VMB node that are currently running in the overlay network.

A7.1.2 Scalability Evaluation

We evaluate scalability by determining the number of messages that are needed to be processed by *one* client, in order to negotiate and locate which other node(s) in the overlay network whom should host a particular type of VMB node (i.e. responsible for a particular task), under different approaches. The following approaches are used in our evaluation: (a) *the SuperPeer approach* ([87]): a client broadcasts to all nodes in the network. Each node responses individually with its capabilities. Then, the client decides which one to be responsible for a task, and notifies other nodes in the network with the decision; (b) *the standard VMB protocol*: the client computes the corresponding keyspace of a type of VMB node, and makes a direct contact with the node(s) that are expected to host that VMB node.

For the standard VMB protocol, for simplicity we assume *M*=1, but with a very heterogeneous and dynamically changing environment i.e. *N*=5 and a 80% failure rate i.e. 80% of the nodes that are expected to host the VMB node of interest are in fact not capable of hosting the VMB node (due to extreme high mobility, lack of processing power etc.). Figure 90 shows the total number of (negotiation) messages processed by the client under different approaches to complete the “which node could aggregate what” and “which node to aggregate what” negotiations in overlay networks of different sizes.

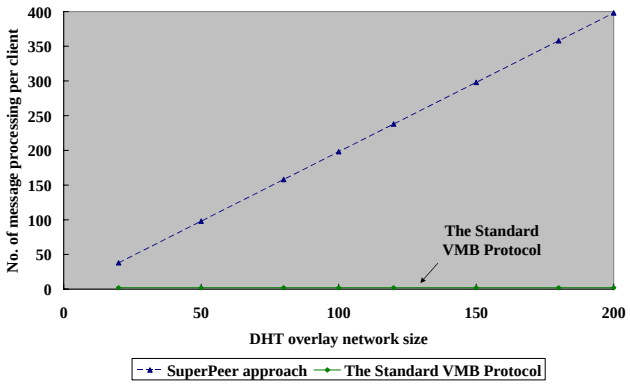


Figure 90 – No. of message processing per node Vs. DHT overlay network size

From the results shown in Figure 90, the SuperPeer approach does not scale. This is because the number of messages to be processed by a client is dependent on the size of

the *entire* managed network. Note that the standard VMB protocol is much more scalable, no matter what the network size is. This is because if the standard VMB protocol is used, no negotiation on "who is capable of aggregating what" or "who should be aggregating what" is needed. The originating node determines the type of the context that it wishes to access, and maps the name to a keyspace, and contacts the node that owns the keyspace *directly* in order to access the VMB node.

For completeness, when testing the scalability of the standard VMB protocol, we simulated a highly dynamic environment by enforcing a 80% failure rate in this evaluation, which means four out of five nodes in the network (that are supposed to host a particular type of VMB node) will fail to host the VMB node. However, the results shown in Figure 90 show a high failure rate has no effect on the scalability of the originating node when comparing to other approaches. This is because, as explained in earlier sections, the recovery process of VMB node is *automatic*, that no re-negotiation is required to locate another capable node when one fails. In case the client first contacted a incapable node, the incapable node forwards the request sequentially to its immediate overlay neighbours (which are expected to host the missing VMB node), then retrieves the aggregated results from one of its overlay neighbours[27], updates itself with the recently received aggregated context, prior to responding to the originating node. The client is not disturbed. The size of the search conducted by the incapable node is dependent on the number of its immediate overlay neighbours, which is always limited (see later).

One may argue that the recovery process (i.e. the retrieval process between an incapable node and its immediate overlay neighbours) incurs additional messages being exchanged. However, the total number of messages involved (including the sequential search conducted by the incapable node) is limited to the number of the incapable node's immediate overlay neighbours only. We have determined that for a 100-node network, the average number of immediate overlay neighbour is ~5.5 nodes, which means the sequential search would incur additionally a maximum of 5.5*2 = 11 messages (one pair of request/respond message per neighbouring node). The total maximum number of messages exchanged would be 13 messages (i.e. one request message from the client, a maximum of eleven messages for recovery communications, and one message for responding to the client after recovery). The total overhead is still much less than other approaches. Therefore, the standard VMB protocol is efficient even when failure rate is high (see also next section for efficiency evaluation).

A7.1.3 Efficiency Evaluation

The efficiency of the simplified VMB protocol (i.e. N=0) is evaluated by determining the average number of overlay hops needed by a randomly selected client (which is also an overlay member) to reach a particular type of VMB node in the overlay, in overlay networks of different sizes[28]. We assume all nodes in the network are 100% capable of hosting any type of VMB node i.e. node failure rate=0% (see later for evaluation on robustness of VMB node). The simplified VMB protocol is chosen for efficiency evaluation because there is

[27] Unless we assume extreme high failure rate (~100%), otherwise *statistically* there should always be at least one (or more) immediate neighbour to an incapable node that is capable of aggregating results.

[28] We have explained earlier that the actual physical hop count between two overlay nodes would depend on routing locality, which is out of scope of this chapter.

always only *one* VMB node per type in the network, so the average hop count is most likely to be higher than the standard VMB node protocol.

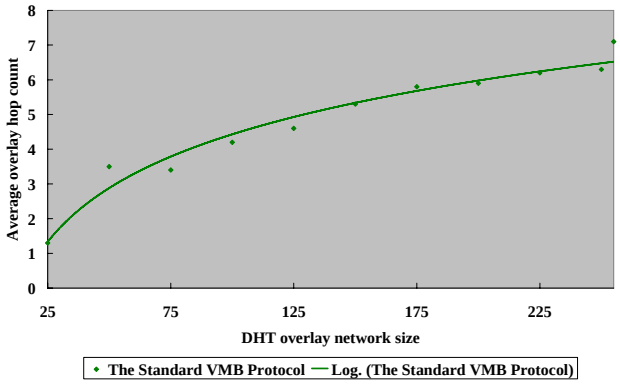


Figure 91 – Average overlay hop count Vs DHT overlay network size

Figure 91 shows the how the overlay hop count varies in overlay networks of different sizes. Our approach is efficient because there are only a few overlay hops in between a VMB node and a client (~6 overlay hops for a 225-node network). We have discussed that our protocol is applicable even if routing locality is optimised. The slope of the curve gradually decreases as the size of network increases. Thus, Figure 91 the effect of a network with an increasing size has limited impact on the efficiency of the simplified/standard VMB protocol.

Robustness Evaluation

We have discussed that the standard VMB protocol is robust to handle heterogeneity and dynamicity in ANs: that is, under the standard VMB protocol, an incapable node is "backup" by its immediate overlay neighbours. The number of "backup" nodes can be adjusted by the value of *N* (and also by *M*). To illustrate the robustness of the standard VMB protocol, we shall prove in this evaluation that the total number of active VMB nodes (of the same type) that are available at one time in the network can be made constant (i.e. a constant number of VMB nodes of the same type in the network), through adjusting the value of *N* (or *M*) under different failure rates.

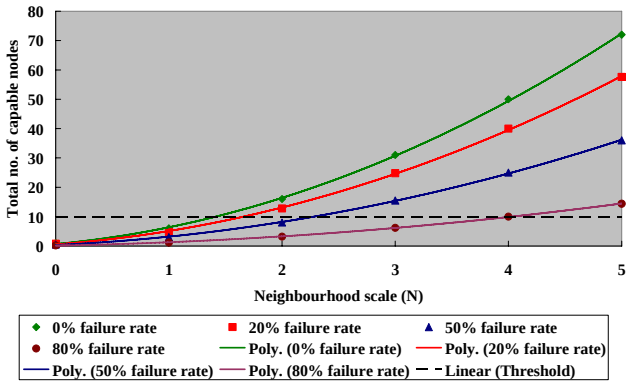


Figure 92 – Comparison between robustness

For simplicity, we set *M*=1. The general rule is that the value of *N* is determined by the failure rate: the higher the failure rate, the higher the value of *N*. The evaluation is carried out over a network of 100 overlay nodes. We assume that an administrator would like to have 10% of the nodes in the network to be the hosts of a particular type of VMB node (based on load balancing, overhead, demands for that type of aggregated context, etc.), so we set the threshold to 10 nodes. The total number of capable nodes (for one type of VMB node at one time) is determined by multiplying the average number of nodes (that host the

same type of VMB node under different values of *N*) by [6.7.1- the failure rate]. The failure rate ranges from 0% to 80%. For example, when *N*=1, we used the DHT simulator to determine that the average number of immediate overlay neighbours to be 5, which gives on average a total of 6 capable nodes in the network (when 0% failure rate applies). With a failure rate of 20%, the average total number of capable nodes would be 6*0.2 = 4.8.

The results shown in Figure 92 shows that the use of the neighbourhood scale *N* can resolve robustness of the standard VMB protocol when the network is heterogeneous and mobile (i.e. higher failure rate). For example, when the network is extremely heterogeneous and mobile e.g. with 80% failure rate, we can adjust *N*=4 to achieve a desired level of robustness i.e. to achieve a desired number of capable nodes in the network (i.e. to have 10 capable nodes in the network at one time, as defined by the administrator). Therefore, the provisioning of *N* (or *M*) in the standard VMB protocol provides the necessary facility to adjust the level of robustness.

A7.1.4 Effect of the Number of VMB nodes on Network Traffic

Once VMB nodes have been setup in an overlay network, they can start aggregating context in the network. The amount of subsequent *context aggregation traffic* generated is directly dependent on the number of active VMB nodes in the network at one time, and more importantly, on the chosen context aggregation algorithm of the VMB node. For example, if a more efficient network context aggregation model was chosen (such as the echo-pattern instead of the star pattern), the administrator might decide to host more capable VMB nodes in the network (i.e. a higher value of *N* or *M*) for enhanced robustness. But in a situation where a much less efficient network context aggregation model was chosen i.e. the star-pattern, the value of *N* (or *M*) should be set to a smaller value (despite of failure rate) to avoid overloading.

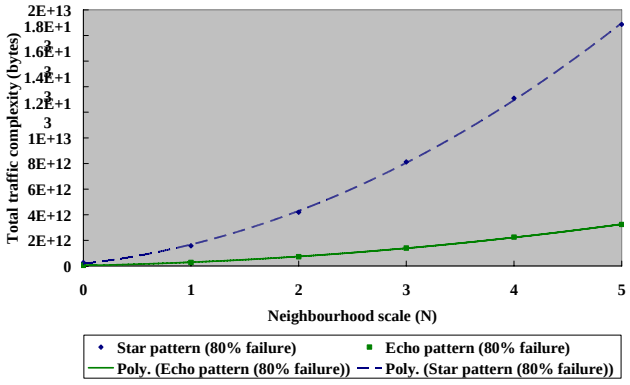


Figure 93 – Network traffic complexity Vs. neighbourhood scale

We suggest that an optimal choice on the appropriate network context aggregation model has a larger impact on network traffic, than the value of *N* (or *M*). Figure 93 shows that at a certain failure rate (fixed at 80% in this case), the total number of traffic complexity (in bytes) generated by different aggregation models when different values of *N* were chosen. The traffic complexity benchmarks for the echo pattern and the star pattern in [93] were used to generate Figure 93. Note that according to Figure 93, the slope of the echo pattern curve is much less than the slopes of the star pattern curve. This shows that network traffic

would increase much less rapidly if a more efficient and scalable aggregation model was chosen (i.e. use the echo pattern instead of the star pattern). This is true for all values of *N*. Also, according to Figure 93, a less ideal choice on aggregation model i.e. star pattern (at *N*=2) would generate 4.192×10^{12} bytes, compare to just 7.2×10^{11} bytes if the echo pattern (as *N*=2) was chosen i.e. this would be a 482.22% increase in network traffic. Whereas (suppose the echo pattern is chosen) an increase of *N* from *N*=1 to *N*=2 increases traffic by just 166.67%*[6.7.1-failure rate][29]. This shows that an optimal choice of aggregation model has a much larger impact on network traffic, than the value of *N*.

A7.1.5 Conclusion

In this section, we have identified the need for a self-organising mechanism that would efficiently and automatically determine a set of nodes in the network underlay and overlay to carry out dedicated network context aggregation tasks. Existing similar approaches such as the SuperPeer approaches rely on real-time negotiations to assign tasks to nodes. However, the use of heavy-weighted real-time negotiations should be limited in a bandwidth-limited environment such as ANs. We presented a novel solution, known as the standard VMB protocol, which enables automatic and self-organising network context aggregation task assignations, balancing and distribution of responsibilities to peers, and task recovery in heterogeneous and dynamically changing ANs, without the need for any expensive real-time negotiation. Our solution is self-organised, efficient, scalable, and robust; and has a low maintenance overhead per node, with dynamic and even load balancing. Also, the standard VMB protocol is fully decentralised, ensuring that every peer has an equal opportunity to become responsible for aggregation tasks; and several VMB nodes of the same type that are simultaneous actively running in the network form a VMB to provide clients with ready access to aggregated network context. The aggregation process is completely distributed as there are no SuperPeers to directly manage peers. Also, we have discussed how robustness can be controlled in our solution, through the use of the neighbourhood scale (*N*) and network scale (*M*), to accommodate different levels of dynamicity and heterogeneity in the network.

As future work, we will investigate synchronisation of final aggregated network context between simultaneously active VMB nodes. Currently, we assume that the administrator is capable of determining the failure rate, and uses the failure rate (and level of routing locality) to control the level of robustness by modifying the values of *N* and *M*. We will deploy our approaches in the ANs testbed (due end of 2006) to investigate further the methodologies for determining in real-time the failure rate (i.e. methods to measure in real-time the level of heterogeneity and dynamicity in the network), and fetch the automatically determined failure rate into our solution, to develop a self-configuring VMB protocol.

[29] Note that in this experiment we want to show the effect of an increasing *N* on network traffic; as such, we kept the failure rate to constant (i.e. 80%). However, as we have described in an earlier section, the value of *N* should be adjusted to accommodate *different* values of failure rate; so in reality, *N* should be increased (from *N*=1 to *N*=2 in this case) *only when failure rate goes up*. Thus, the (average) increase of traffic complexity when *N* is increased in this example is 166%*[1-failure rate]. Whatever the result, the increase in traffic complicity will always be much less than when the other option was chosen (482.22%).

A7.2 Performance Evaluation of Context Distribution

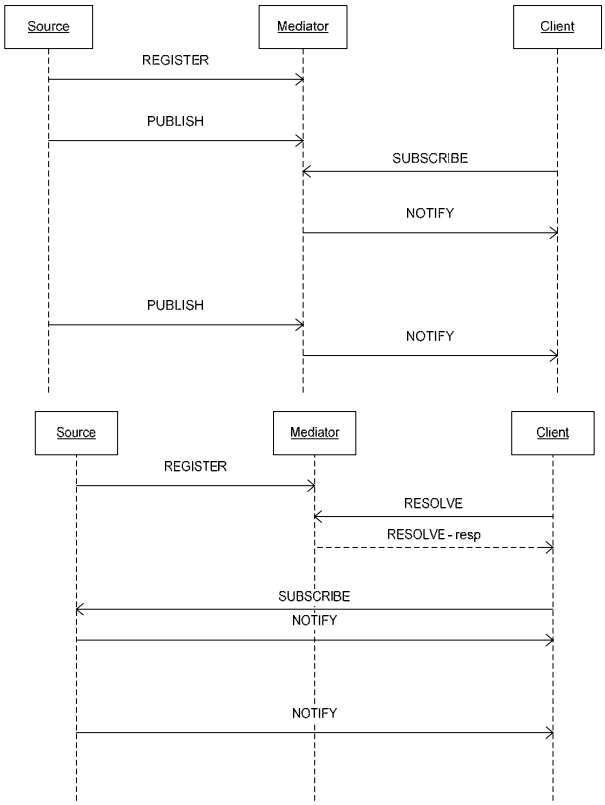
In order to study the performance of the AN management model, we start by the AN Management Node model and make some assumptions:

- The most critical part with respect to performance is processing of policy enforcement and context dissemination.
- The internal processing of context information, including aggregation, communication with the CIB, network sensors, etc. is less critical with respect to performance.

Based on these assumptions, we explore the process of context dissemination and policy enforcement by identifying possible variants of the architecture and the important characteristics of the involved nodes. The important characteristics will be the delay introduced by policy enforcement, transmission and queuing.

The architecture is represented by a graph, where the nodes are the context sources, the clients and a Mediator. The Mediator gathers functionality representing the internal processing in the ContextWare. It is modelled in this way since performance of the details in the architecture of the ContextWare is outside the scope of this study. The links in the graph represent the direction of transmission of context information.

The Mediator may play a more or less active role in the context dissemination, depending on the architecture in question. Specifically, we distinguish between a centralised and a decentralised architecture as follows. In a centralised architecture, context information is transmitted through the Mediator, which performs some processing on the data and notifies a given set of context clients accordingly. This approach is similar to presence distribution in SIP [76]. In a decentralised architecture, the context source will notify the context clients directly without the involvement of the Mediator. The difference between the two architectures is illustrated in Figure 94.



Centralised context dissemination Decentralised context dissemination

Figure 94 – Centralised and decentralised architectures for context dissemination – the Mediator represents ContextWare

A7.2.1 Modelling

In the following we shall evaluate the performance of centralised and decentralised (P2P) architectures for distribution of context changes. We shall focus on the latency in dissemination of context changes, i.e. the time from a context change occurs till it reaches the relevant context clients.

It is assumed that the following factors influence the time it takes context information to reach the context clients:

- Policy enforcement (PE): Filtering, scale conversion, derivation of higher level context, etc.
- Transmission: The time it takes to transmit context information over an outgoing link.

Some Results from Queuing Theory

Results presented in this section may be found in[75]. First, consider a general system with *L* jobs. Jobs arrive at the system with a rate λ, stay in the system for a time *T*, and then leave. Then Little's formula states the following relation:

$$E[L] = \lambda * E[T]$$

where E is the expectation, i.e. E[*L*] is the expected number of jobs and E[*T*] is the expected time a job stays in the system. Consider a one-server queuing system with infinite queue as depicted in Figure 95. Jobs are arriving with rate λ jobs per time unit, and are being served at a rate of μ jobs per time unit. The expected service time is $1/\mu$, and the expected inter-arrival time is $1/\lambda$.

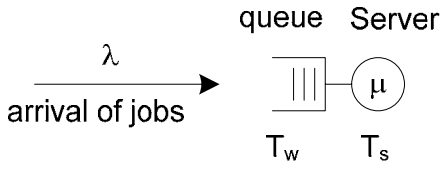


Figure 95 – One-server queuing system

Now assume that the arrival process is a general process (inter-arrival times may be dependent and follow a general distribution with intensity λ), and that the service process is a Markov process (service times are independent and identically distributed according to the exponential distribution with intensity μ). With Kendall's notation we describe this as an G/M/1-system (General arrival process, Markov server process, one server, infinite queue). Then the expected waiting time for all arrivals may be expressed as

$$E[T_w] = \frac{\alpha \cdot E[T_s]}{1-\alpha}$$

Eq 1

where 1-α is the probability that the server is not working.

For an M/G/1-system (Markov arrival process, general service process) the expected waiting time for all arrivals may be expressed as

$$E[T_w] = \frac{A \cdot E[T_s]}{2(1-A)} \cdot \varepsilon$$

Eq 2

where

$A = \lambda/\mu$

$\varepsilon = \mathrm{E}[T_s^2] / \mathrm{E}[T_s]^2$ (shape factor for the distribution of T_s)

This is Pollaczek-Khintchines's formula for M/G/1. If the serving process is deterministic (constant service time), the shape factor ε becomes equal to one. Hence for M/D/1:

$$E[T_w] = \frac{A \cdot 1/\mu}{2(1-A)} = \frac{\lambda/\mu}{2(\mu-\lambda)}$$

Eq 3

If the serving process is a Markov process (service times are independent and identically distributed according to the exponential distribution), the shape factor ε becomes equal to two. Hence for M/M/1:

$$E[T_w] = \frac{A \cdot 1/\mu}{2(1-A)} \cdot 2 = \frac{\lambda/\mu}{\mu-\lambda}$$

Eq 4

This result may also be derived from Erlang's second formula, which is valid for M/M/n queuing systems.

Decentralised Context Distribution

Now consider a system as depicted in Figure 96. Source *i* is monitoring the state of some system representing e.g. flow context, QoS context, etc. A total of n_i clients are subscribing to context notifications.

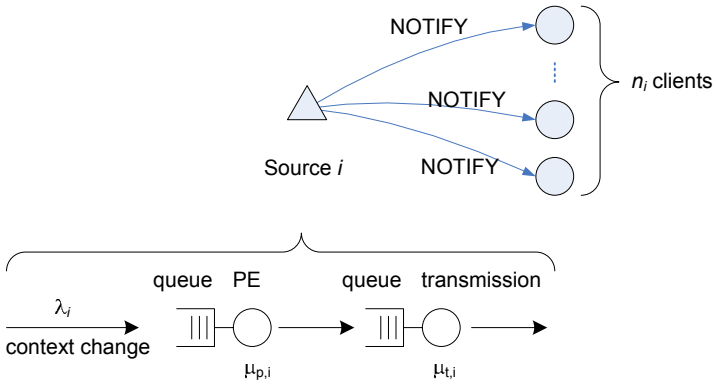


Figure 96 – Peer-to-peer context distribution from source i

Assume state changes representing context source *i* are generated according to a Markov process. This means inter-arrival times for state changes are independent and identically distributed according to an exponential distribution with intensity λ_i. Now we are interested in $T_{D,i}$, the time elapsed from a state change is discovered at source *i* until it has been sent to all relevant context consumers:

$$T_{D,i} = T_{p,i} + T_{t,i}$$

where $T_{p,i}$ is time spent for policy enforcement (queue + execution of PE), and $T_{t,i}$ is time spent for transmission of context information to the clients. In the following we assume that time spent for execution of PE follows a exponential distribution with intensity $\mu_{p,i}$. Then from the previous section we have that

$$E[T_{p,i}] = \frac{\lambda_i / \mu_{p,i}}{\mu_{p,i} - \lambda_i} + \frac{1}{\mu_{p,i}} = \frac{1}{\mu_{p,i} - \lambda_i}$$

Policy enforcement may result in a filtering event: In average, n_i' of the n_i clients are going to receive the context update, where $0 \le n_i' \le n_i$. Hence after PE, context updates will be transmitted to n_i' clients. This is identical to placing n_i' jobs in the transmission queue. For the transmission system we base our calculations on the following assumptions:

- Each job require a transmission time which is exponentially distributed with intensity $\mu_{t,i}$. The rationale for assuming varying job lengths for the same context event comes from the preceding PE: For some of the clients the context information may have gone through scale conversion, aggregation, or derivation to achieve higher level context, hence the volume of information to be transmitted will depend on the receiving client.
- The overall arrival rate to the transmission system will be $\lambda_i * n_i'$ since each change in the system state generates n_i' jobs to be transmitted. However, the arrival process will not be a Markov process since jobs arrive in batches (inter-arrival times are not independent). The queuing system is a G/M/1-system, and expected queue time depends on $1-\alpha$, the probability that the server is working (see Eq 1).

The probability that the transmission server is working, α, may be approximated in the following way: Consider a long time interval τ. During this interval there are in average $\lambda_i * n_i' * \tau$ arriving jobs. The time spent transmitting these jobs is $\lambda_i * n_i' * \tau / \mu_{t,i}$ and hence the probability that the server is working may be written as $\alpha = \lambda_i * n_i' / \mu_{t,i}$. By using Eq 1 we get

$$E[T_{t,i}] = \frac{\lambda_i \cdot n_i' / \mu_{t,i}}{1 - \lambda_i \cdot n_i' / \mu_{t,i}} \frac{1}{\mu_{t,i}} + \frac{n_i'}{\mu_{t,i}}$$

Eq 5

The first term is the expected waiting time in queue seen by the first of the n_i' jobs; the second term is the expected time for transmitting all of the jobs (recall that we are interested in the time it takes before all n_i' clients have received the notification). Eventually we get

$$E[T_{t,i}] = \frac{n_i'}{\mu_{t,i}} \frac{\mu_{t,i} - \lambda_i \cdot (n_i' - 1)}{\mu_{t,i} - \lambda_i \cdot n_i'}$$

and

$$E[T_{D,i}] = E[T_{p,i}] + E[T_{t,i}] = \frac{1}{\mu_{p,i} - \lambda_i} + \frac{n_i'}{\mu_{t,i}} \frac{\mu_{t,i} - \lambda_i \cdot (n_i' - 1)}{\mu_{t,i} - \lambda_i \cdot n_i'}$$

Eq 6

Centralised Context Distribution

Now we assume a centralised architecture as depicted in Figure 97. State changes are published by sources *1-m*, while the Mediator will handle policy enforcement and transmission to the clients. Let $T_{C,i}$ be the time elapsed from a context change is detected at source *i* till it has been sent to all the relevant clients.

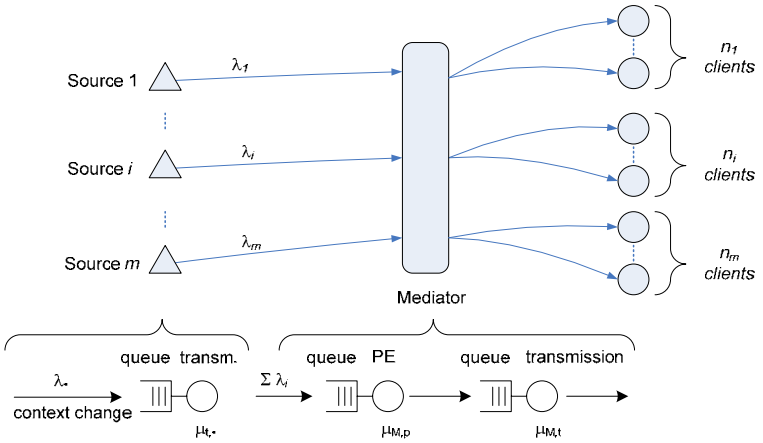


Figure 97 – Centralised context distribution

The following is valid

- The aggregated traffic towards the Mediator represents a Markov process with arrival rate $\Lambda = \Sigma_{i=1,m}\lambda_i$.
- An arbitrary context update will in average be transmitted from the Mediator to *n'* = $\Sigma_{i=1,m} n_i'/m$ clients.
- A context update from source *i* will have the following expected delay through the entire system:

$$E[T_{C,i}] = \frac{1}{\mu_{t,i} - \lambda_i} + \frac{1}{\mu_{M,p} - \Lambda} + \frac{\Lambda \cdot n' / \mu_{M,t}}{1 - \Lambda \cdot n' / \mu_{M,t}} \frac{1}{\mu_{M,t}} + \frac{n_i'}{\mu_{M,t}}$$

The first term on the right hand side corresponds to transmission of context update from source i, the second is policy enforcement in the Mediator (note that this term is independent of which context source the update comes from). The third term is the waiting time in queue for the n_i' jobs to be transmitted, and the last term is the transmission of these jobs. For a context update from an arbitrary source we have

$$E[T_C] = \sum_{i=1}^{m} \frac{\lambda_i}{\Lambda} E[T_{C,i}]$$

Eq 7

To simplify we assume that all context sources have identical characteristics:

λ_i = λ for all *i*

$\mu_{t,i}$ = μ_t for all *i*

$\mu_{p,i}$ = μ_p for all *i*

n_i = *n* for all *i*

n_i' = *n'* for all *i*

As a consequence we get

$$\Lambda = m \cdot \lambda$$

$$E[T_{D,i}] = E[T_D] = \frac{1}{\mu_p - \lambda} + \frac{n'}{\mu_t} \frac{\mu_t - \lambda \cdot (n'-1)}{\mu_t - \lambda \cdot n'} \qquad \forall i$$

Eq 8

$$\begin{aligned} E[T_{C,i}] = E[T_C] &= \frac{1}{\mu_t - \lambda} + \frac{1}{\mu_{M,p} - \Lambda} + \frac{\Lambda \cdot n' / \mu_{M,t}}{1 - \Lambda \cdot n' / \mu_{M,t}} \frac{1}{\mu_{M,t}} + \frac{n'}{\mu_{M,t}} \\ &= \frac{1}{\mu_t - \lambda} + \frac{1}{\mu_{M,p} - \Lambda} + \frac{n'}{\mu_{M,t}} \frac{\mu_{M,t} - \Lambda \cdot (n'-1)}{\mu_{M,t} - \Lambda \cdot n'} \\ &= \frac{1}{\mu_t - \lambda} + \frac{1/m}{\mu_{M,p}/m - \lambda} + \frac{n'}{\mu_{M,t}} \frac{\mu_{M,t}/m - \lambda \cdot (n'-1)}{\mu_{M,t}/m - \lambda \cdot n'} \qquad \forall i \end{aligned}$$

Eq 9

Clearly delays E[T_C] and E[T_D] will increase as the denominators in these equations approach zero. For stability we require the following:

$$\begin{aligned} &\mu_p - \lambda > 0 \\ &\mu_t - \lambda \cdot n' > 0 \\ &\mu_{M,p}/m - \lambda > 0 \\ &\mu_{M,t}/m - \lambda \cdot n' > 0 \end{aligned}$$

Eq 10

When comparing the denominators for Eq 8 and Eq 9, we also see that μ_p is substituted by $\mu_{M,p}/m$, and μ_t is substituted by $\mu_{M,t}/m$. This means that E[T_C] and E[T_D] will have the same singularities if $\mu_p = \mu_{M,p}/m$ and $\mu_t = \mu_{M,t}/m$, and one might argue that these relations should be set to give a reasonable comparison. However, in the next section we shall see that this is not necessarily a natural assumption.

A7.2.2 Case Studies

Numerical values are chosen based on the following considerations:

- Since the Mediator is a centralised component dedicated to context distribution, we will in general assume it is more powerful than the context sources when it comes to policy enforcement and transmission.
- We will vary the number of context sources *m*, the average number of clients *n'* receiving context updates from each source, and the arrival intensity λ of context changes to see the impact on distribution delay.
- For stability we require that parameter values satisfy Eq 10.

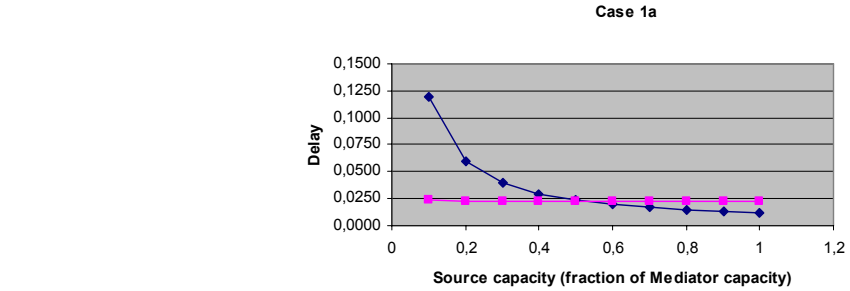
- As discussed in the end of the previous section, E[T_C] and E[T_D] will have the same singularities if $\mu_p = \mu_{M,p}/m$ and $\mu_t = \mu_{M,t}/m$. However, as the number of sources m increases, one cannot assume that Mediator capacity will increase accordingly: $\mu_{M,t}$ and $\mu_{M,p}$ will in general be less than $m\mu_t$ and $m\mu_p$, respectively. Hence as a result E[T_C] will be dominated by its singularities more often than E[T_D] as system load increases.

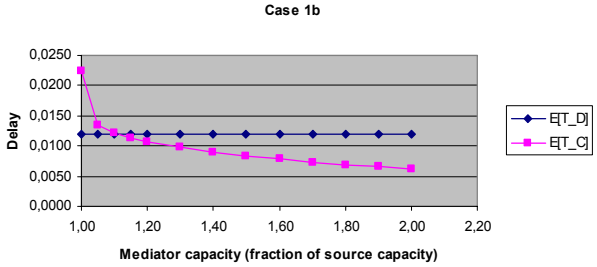
Table 9 – Overview of parameter values

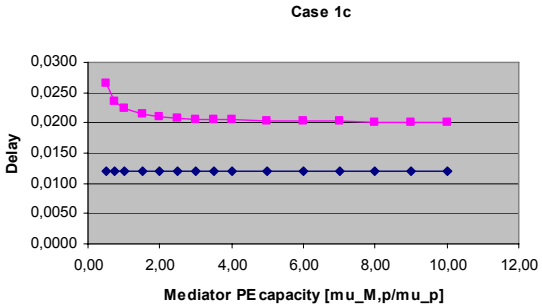
Case	Description	m[30]	n'	□	□	μ_p	μ_t	$\mu_{M,p}$	$\mu_{M,t}$
1a	Sources' capacity vary[31]	100	99	1	100	50-$5*10^2$	10^3-10^4	$5*10^2$	10^4
1b	Mediator capacity vary	100	99	1	100	$5*10^2$	10^4	$5*10^2$-10^3	10^4-$2*10^4$
1c	Mediator capacity for Policy Evaluation vary	100	99	1	100	$5*10^2$	10^4	250-$5*10^3$	10^4
1d	Mediator capacity for Policy Evaluation vary (2)	100	99	1	100	$5*10^2$	$2*10^4$	250-10^3	$2*10^4$
2a	Large number of sources and clients. *n'* varies.	100	1-$5*10^3$	1	100	250	$5*10^3$	$5*10^2$	10^4
2b	Large number of sources and clients. *n'* varies.	1000	1-$5*10^4$	1	1000	10^3	$5*10^4$	$2*10^3$	10^5
2c	Large number of sources and clients. *m* varies.	1-100	9	1	1-100	50	$5*10^3$	10^2	10^4
2d	Large number of sources and clients. *m* varies.	1-100	99	1	1-100	50	$5*10^3$	10^2	10^4
3	Varying arrival intensity λ for context change	100	100	10^{-2}-50	1-$5*10^3$	50	$5*10^4$	10^3	10^5

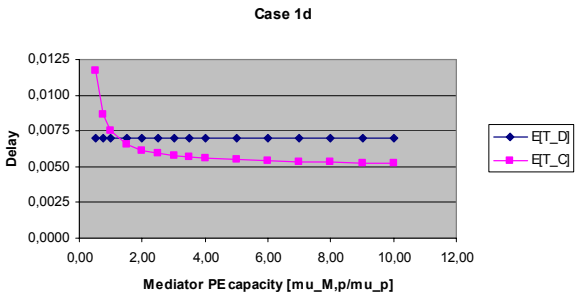
[30] *m* is the number of context sources, *n'* is the average number of clients to which a context update is sent.

[31] Processing capacity for policy enforcement, transmission capacity.









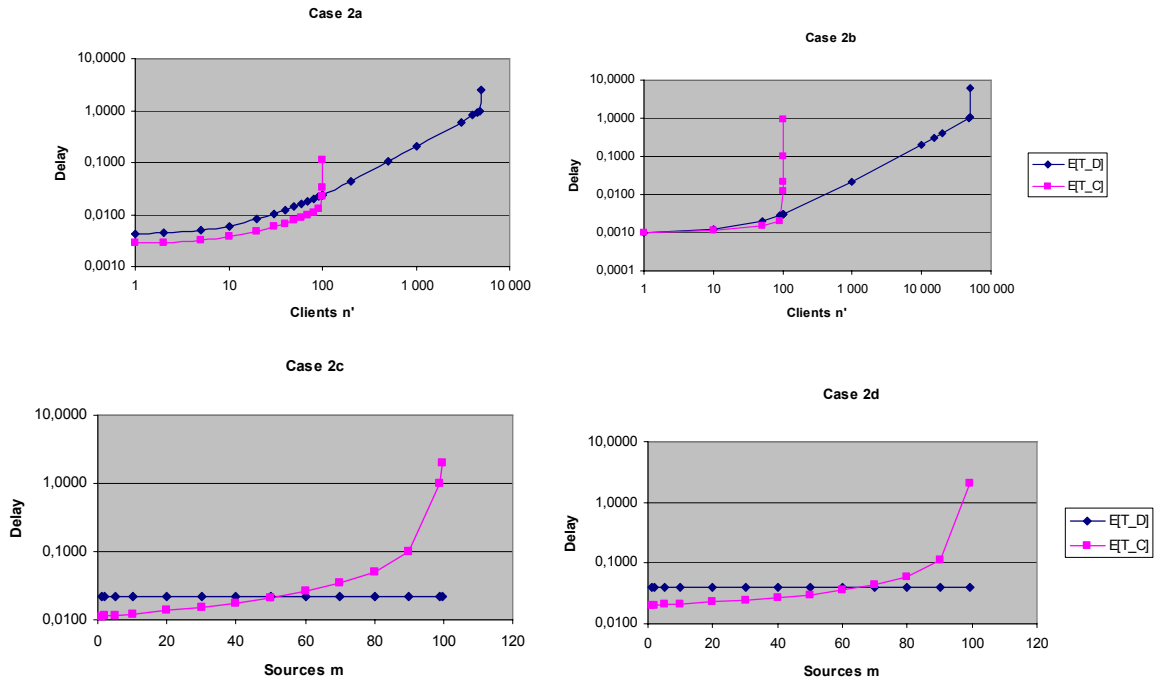
Case	Description	m	N'	□	□	μ_p	μ_t	$\mu_{M,p}$	$\mu_{M,t}$
1a	Sources' capacity vary	100	99	1	100	$50\text{-}5*10^2$	$10^3\text{-}10^4$	$5*10^2$	10^4
1b	Mediator capacity vary	100	99	1	100	$5*10^2$	10^4	$5*10^2\text{-}10^3$	$10^4\text{-}2*10^4$
1c	Mediator capacity for Policy Evaluation vary	100	99	1	100	$5*10^2$	10^4	$250\text{-}5*10^3$	10^4
1d	Mediator capacity for Policy Evaluation vary (2)	100	99	1	100	$5*10^2$	$2*10^4$	$250\text{-}10^3$	$2*10^4$

Figure 98 – Numerical results for Case 1

In Figure 98 cases 1a and 1b show that when source capacity (PE processing and transmission) is slightly lower (20-50) than the Mediator capacity, i.e. the centralised architecture gives better performance. Hence, when going from a decentralised to a centralised architecture, very little extra capacity is needed in the Mediator in order to justify this reorganisation. This is due to a basic result from queuing theory: Concentration of request streams towards one central processing node gives better overall resource usage than many distributed processing nodes with individual request streams.

Based on results from cases 1a and 1b; for many of the following cases we will use $\mu_{M,t} = 2*\mu_{M,t}$ and $\mu_{Mp} = 2*\mu_{M,p}$ since this corresponds to more or less equal performance for the centralised and decentralised architectures.

In cases 1c and 1d, we investigate the effect of $\mu_{M,p}$ (PE processing capacity at the Mediator). Here both architecture alternatives have equal transmission capacity. From case 1c we see that increase in $\mu_{M,p}$ will not provide sufficient improvement. However, in case 1d where we have doubled the transmission capacity for both architectures. Now the centralised architecture appears to be a better choice. This shows that transmission capacity seems like a more significant parameter than processing capacity.



Case	Description	m[32]	n'	□	□	μ_p	μ_t	$\mu_{M,p}$	$\mu_{M,t}$
2a	Large number of sources and clients. *n'* varies.	100	1-$5*10^3$	1	100	250	$5*10^3$	$5*10^2$	10^4
2b	Large number of sources and clients. *n'* varies.	1000	1-$5*10^4$	1	1000	10^3	$5*10^4$	$2*10^3$	10^5
2c	Large number of sources and clients. *m* varies.	1-100	9	1	1-100	50	$5*10^3$	10^2	10^4
2d	Large number of sources and clients. *m* varies.	1-100	99	1	1-100	50	$5*10^3$	10^2	10^4

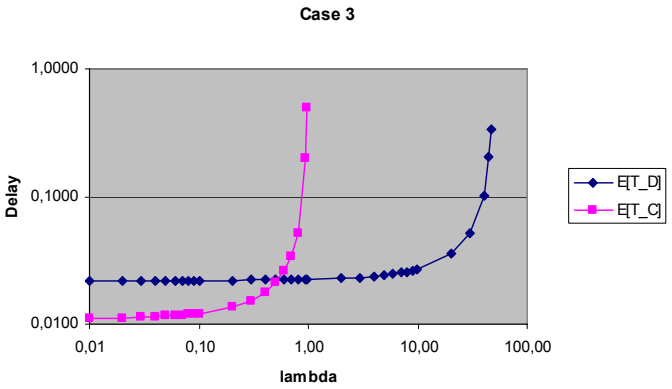
Figure 99 – Numerical results for Case 2 (note log scale on 2nd axis)

Case 2 in Figure 99 shows the effect of system scaling. In cases 2a and 2b we increase *n'*, the average number of clients receiving context updates. This leads to increased system load, and eventually both E[T_D] and E[T_C] will reach saturation. In the area before E[T_C] reaches saturation (*n'*<100), we see that it has better performance than E[T_D]. However, we see that the centralised architecture reaches saturation several decades before the

[32] *m* is the number of context sources, *n'* is the average number of clients to which a context update is sent.

decentralised architecture, and hence it is clear that the decentralised architecture scales far better than the centralised.

In cases 2c and 2d we increase the number of context sources *m*. Now recall that in the calculation of E[T_D] we consider distribution of context form one arbitrary source, and do not see the other *m*-1 sources. Hence the calculation of E[T_C] will be independent of *m*. For the centralised architecture, however, delays are depending strongly on increase in *m* since this means an increase in the overall request input. We see that for cases 2c (*n'*=9) and 2d (*n'*=99), saturation is reached at more or less the same point (for *m*=100), hence the point of saturation is more or less independent of *n'*. This may also be inferred from investigation of Eq 9.



Case	Description	m[33]	n'	□	□	μ_p	μ_t	$\mu_{M,p}$	$\mu_{M,t}$
3	Varying arrival intensity λ for context change	100	100	10^{-2}-50	1-5*10^3	50	5*10^4	10^3	10^5

Figure 100 – Numerical results for case 3 (Note: Log-log plot)

Figure 100 confirms results from case 2: In the area before E[T_C] reaches saturation (λ<0.9), we see that it has better performance than E[T_D]. However, we see that the centralised architecture reaches saturation several decades before the decentralised architecture, i.e. the decentralised architecture scales better.

A7.2.3 Conclusion

The aim of this analysis has been to provide general results and corresponding conclusions. A minor approximation was made in Eq *5* to provide formulas for the analysis. This may lead to slightly incorrect numerical results, but deviations will be within insignificant bounds. Moreover, we are aware that the choice of parameter values has some impact on the results, yet we believe that our investigations cover sufficiently large parameter ranges to give valid support for the conclusions.

General observations from cases 1-3 are:

[33] *m* is the number of context sources, *n'* is the average number of clients to which a context update is sent.

- For low load, the centralised architecture gives better performance in terms of lower delay. A centralised node will require slightly more capacity than the corresponding sources, and may hence be used to reduce overall system costs.
- The decentralised architecture scales far better, and will e.g. never reach saturation as the number of context sources increases.
- For the centralised architecture, the number of context sources *m* has greater impact on performance than the number of clients *n'*.

A8 References

[108] J. M. Kleinberg, “The Small-World Phenomenon: an Algorithmic Perspective”, In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000

[109] G. Manku, M. Bawa, and P. Raghavan, “Symphony: Distributed Hashing in a Small World”, In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003), 2003.

[110] J. Aspnes, Z. Diamadi, G. Shah, “Fault tolerant Routing in Peer-to-peer Systems”, In Proceedings of PODC ’02 Monterey, California USA, 2002

[111] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications”, In Proceedings of the 2001 ACM Sigcomm Conference, pp. 149–160, ACM Press, 2001.

[112] P. Maymounkov and D. Mazieres, “Kademlia: A peer-to-peer information system based on the xor metric”, In Proceedings of IPTPS02, Cambridge, USA, Mar. 2002.

[113] A. Rowstron, P. Druschel: “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems”, In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms, 2001

[114] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, “A Scalable Content-Addressable Network”, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM) 2001, pp. 161-172.

[115] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host identity protocol architecture - work in progress (draft-moskowitz-hip-arch-06), June 2004.

[116] K. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, “The Impact of DHT Routing Geometry on Resilience and Proximity”, In Proceedings of ACM Sigcomm 2003, ACM Press, 2003.

[117] J. S. Kong, J. S. A. Bridgewater, V. P. Roychowdhury: "A General Framework for Scalability and Performance Analysis of DHT Routing Systems", In proc. of the International Conference on Dependable Systems and Networks (DSN'06)

[118] D. Loguinov, A.Kumar, V. Rai, and S. Ganesh, “Graph-Theoretic Analysis of Structured Peer-to-Peer Systems; Routing Distances and Fault Resilience”, In Proceedings of the ACM SIGCOMM '03 Conference, Karlsruhe, Germany, August 2003.

[119] D. Liben-Nowell, H. Balakrishnan, and D. Karger, “Analysis of the evolution of peer-to-peer networks,” in Proc. of ACM PODC, Jul. 2002

[120] M. Jelasity and O. Babaoglu, “T-Man: Fast gossip-based construction of large-scale overlay topologies”, Technical Report Report UBLCS-2004-7, University of Bologna, Department of Computer Science, Bologna, Italy, May 2004.

[121] A. Shaker and D. S. Reeves, Self-Stabilizing Structured Ring Topology P2P Systems, In Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing, pp. 39–46, Konstanz, Germany, Aug. 2005.

[122] K. Aberer, L.O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, M. Hauswirth: “The Essence of P2P: A Reference Architecture for Overlay Networks”, In proc. of the 5th IEEE International Conference on Peer-to-Peer Computing, 2005

[123] A. Datta, S. Girdzijauskas, and K. Aberer, “On de bruijn routing in distributed hash tables: There and back again”, in proc. of Peer-to-Peer Computing, 2004, pp. 159–166.

[124] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz: "Handling churn in a DHT", Tech. Rep. UCB/CSD-3-1299, UC Berkeley, Computer Science Division, Dec. 2003.

[125] D6-3 “Second paper on context aware networks” – Ambient Networks Project Phase I December 2005; http://www.ambient-networks.org/phase1web/main/deliverables.html

[126] D8-3 “Ambient Network Management - Proof of concepts and evaluations” – Ambient Networks Project Phase I- December 2005; http://www.ambient-networks.org/phase1web/main/deliverables.html

[127] D10-R1 “Interim WP-D Report on Integrated design for context, network and policy management” – July 2006 - https://bscw.ambient-networks.org/bscw/bscw.cgi/375766

[128] MD4.3/MF-7/ME-4 “WP-D/WP-E/WP-F Milestone Report on Cross Layers Interactions, Optimisations and Enhancements” – October 2006; https://bscw.ambient-networks.org/bscw/bscw.cgi/380026

[129]