uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Ibrahim Al-Oqily
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Ph.D. (Computer Science)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Autonomous Management for Service Specific Overlay Networks

TITRE DE LA THÈSE / TITLE OF THESIS

Ahmoud Karmouch
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Shikharesh Majumdar

Samuel Pierre

Amiya Nayak

Tet Yeap

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

AUTONOMOUS MANAGEMENT FOR SERVICE SPECIFIC OVERLAY NETWORKS

By

Ibrahim Z. Al-Oqily

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For Doctor of Philosophy degree in
Computer Science

School of Information Technology and Engineering
(SITE)
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-48382-4
Our file Notre référence
ISBN: 978-0-494-48382-4

Abstract

Overlay networks emerging as a main player in content delivery because they provide effective and reliable services that are not otherwise available. Extensive research has recently focused on the design of Service Specific Overlay Networks (SSON) to deliver media in a heterogeneous environment. This dissertation investigates the problem of SSON's management, and proposes an autonomous SSON management framework. The framework consists of a policy layer that in turn constitutes a set of Overlay Policy Enforcement Points (OPEP) and Overlay Policy Decision Points (OPDP). An OPEP is where policy decisions are actually enforced—policy decisions are made primarily at the OPDP. The research plan presented in this dissertation addresses the functionalities of these components.

To realize dynamic SSONs construction, a novel, fault-resilient semantic overlay for MediaPorts resource discovery is proposed. It allows services to be efficiently and accurately located, and is based on a widely studied family of chordal rings called the optimal chordal ring. In addition to the semantics of the services offered, our solution is based on the geographical locations of the nodes.

The increased complexity and heterogeneity of SSONs led to the proposal of autonomic overlays management architecture. Overlays are viewed as a dynamic organization for self-management in which self-interested nodes can join or leave according to their specific goals. It dynamically adapts the behavior of the overlay network to the preferences of the user, network, and service providers.

To capture the overlay nodes autonomic behavior, a new approach for SSONs self-organized composition is proposed. Using a self-organizing approach, autonomic entities are dynamically and seamlessly composed into SSONs to achieve system-wide goals. The algorithm that encompasses that approach is powered by learning rules induced from biological systems, and endowed with filtering rules to achieve the highest possible performance.

Experimental studies are presented to demonstrate the performance of the proposed schemes.

Dedication

To the spirit of my father, who had dedicated his whole life to his family, To my mother, for her unconditional and endless love, to Layth, Zaki, and Hazem, my sons, for their infinite love, to my wife, Shereen, for her continuous encouragement and support.

Acknowledgements

I would like to offer my profound thanks to my supervisor, Prof. Ahmed Karmouch, for his guidance, patience, and encouragement during my Ph.D. years at The University of Ottawa, during which he have taught me how to transcend above my short comings to become a good researcher, and a better person. He is not only a thesis advisor, but also a role model to me; without his continued support, encouragement, and especially his guidance and mentoring, it would not have been possible to complete this work.

I would also like to thank all my colleagues in the IMAGINE research laboratory. Their input, cooperation, and fruitful discussions always helped clarify my ideas. I am very lucky to have worked with these brilliant minds, and to have learnt so much from them.

I would like to thank my parents for their encouragement and support. I am indebted to their love and to their belief in me, and for even more than I will ever be able to express.

I would like to thank my wife, Shereen, for all the sacrifices that she had put during my research work. Her patience and encouragement were a source of continuous motivation for me.

Contents

List of Figures

Abbreviations

AAs	Active Applications
AC	Autonomic Computing
ACS	Ambient Control Space
AI	Artificial Inelegance
ALA	Analyze/Learning Agent
AM	Autonomic Manager
AO	Autonomic Overlays
APA	Automated Policy Adaptor
ARI	Ambient Resource Interface
AS	Autonomous System
ASI	Ambient Service Interface
BCP	Bounded Composition Probing
BGP	Border Gateway Protocol
CA	Communication Agent
CBR	Case-Based Reasoning
CDN	Content Distribution Networks
CE	Correspondent Entity
COA	Care-Of Address
COPS	Common Open Policy Service
DHCP	Dynamic Host Configuration Protocol
EEs	Execution Environments

FA	Foreign Agent
HA	Home Agent
ID	Identification
IETF	Internet Engineering Task Force
IMS	IP-based Multimedia Subsystem
IT	Information Technology
KB	Knowledge Base
LDAP	Lightweight Directory Access Protocol
LF	Limited-Flooding
MA	Monitoring Agent
MC	MediaClient
ME	mobile entity
MIP	Mobile IP
MP	MediaPorts
MPDS	Media Port Directory Service
MS	MediaServer
OCRA	Overlay Conflict Resolution Agent
OCS	Overlay Control Space
ONAM	Overlay Node Autonomic Manager
ONode	Overlay Node
OPDP	Overlay Policy Decision Point
OPDPMA	Overlay Policy Decision Point Management Agent
OPEP	Overlay Policy Enforcement Point

OPMA	Overlay Policy Management Agent
OSL	Overlay Support Layer
P2P	Peer-To-Peer
PD	Path-Directed
PDP	Policy Decision Point
PEA	Policy Enforcement Agent
PEP	Policy Enforcement Point
PG	Policy Generator
QoS	Quality of Service
RF	Reinforcement Learning
RIA	Resource Interface Agent
RTP	Real-Time Protocol
RTT	Round Trip Time
SAM	System Autonomic Managers
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMART	Smart Media Routing and Transport
SORD	Semantic Overlay Resource Discovery
SP	Service Provider
SPDP	System Policy Decision Point
SSON	Service-specific overlays network
TCP	Transmission Control Protocol
TTL	Time to Live

UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
VMO	Virtual Management Overlay
VoD	Video-on-Demand
WSDL	Web Service Description Language

Chapter 1

Introduction

1.1 Overview

The growth of the Internet in terms of size and speed, as well as the flood of network applications and services that have been deployed in the last few years, is indicative of a shift from the traditional communication systems designed for simple data transfer applications to highly distributed and dynamic systems. Naturally, the spread of such systems has led to an increase in Multimedia development, in itself a feature that has become indispensable in networking environments. Audio and video content on the internet are more popular than ever, and many systems are designed with the purpose to carry this media; video conferencing, video on demand, IP Telephony, and Internet TV are but a few. In addition to being of large scale, these distributed networks and applications are unpredictable and complex; they are highly dynamic in changing environments. As a result, their management (networks and applications) is continuously faced with new complexities, putting the burden on the shoulders of network managers and service providers to design and implement mechanisms that are aware of the nature of different applications demands, and that can conform to various users' requirements. This has left management system paradigms in a continuous struggle to keep up with the ever increasing demands, and advancing technologies.

Another aspect that has contributed to increased management complexity is the rapid growth of overlay networks and their users. Overlay networks consist of a set of nodes that are connected via virtual links, and are built on top of other computer networks with the purpose of implementing new applications that are not readily available in the underlying network. They can be used to increase routing robustness and security, reduce duplicate messages, and provide new services for mobile users. They

can also be incrementally deployed on end hosts without the involvement of ISPs, and they do not incur new equipments or modifications to existing software or protocols. Overlay networks are becoming more popular because of their flexibility and their ability to offer new services; extensive research that has been recently exerted in the realms of overlay networks has focused on the design of specific networks to deliver media in a heterogeneous environment. In that course, a specific overlay network for each multimedia delivery service is created, leading to hundreds of overlays coexisting, and as a result, increasing management complexity and posing additional challenges to ISPs. This—in addition to rapid growth of systems such as P2P networks, pervasive computing networks, wireless sensor networks, ad-hoc networks, and wireless communication technology—renders traditional network management operations insufficient, and incurs new requirements on the networks: To become autonomous, scalable, interoperable, and adaptable to the increasingly dynamic and the widely distributed network demands.

Management refers to the task of planning, allocating, configuring, deploying, administering, and maximizing the utilization of the underlying network resources. Functionalities of a management system also include aspects such as authorization, security management, reliability assurance, and performance guarantees. Little progress has been made in addressing the problem of designing an overall autonomous management framework for service-specific overlay networks that can be self-configurable and adaptable by automating their management tasks.

In this dissertation, we address the problem of developing an autonomous and self-adaptable management framework for service-specific overlay networks. This chapter briefly discusses different aspects of the problem of autonomous management, and presents the motivation behind the proposed work. Subsequently, the proposed management architecture is briefly described, and the contributions are outlined. Finally, the organization of the remainder of the dissertation is presented.

1.2 Service Specific Overlay Networks

Our lab (IMAGINE) was involved in the European project, The Ambient Networks [1], in which a working group has developed a sub-project called Smart Media Routing and Transport (SMART) [2]. The work presented in this thesis is developed using SMART as the starting point.

Media distribution, adaptation, and caching have been very active areas of research in the last few years. However, most of the proposed work has taken only a partial view of the overall problem of media routing and delivery. Past research work was mainly dedicated to either caching architectures, media adaptation, or multicast protocols. Furthermore, the proposed solutions brought about by the research efforts were usually optimized to solve only one specific problem, such as a network congestion state, limitations of end-devices, or mobility. In contrast, the work on media routing and adaptation in SMART takes a holistic view, and supports media adaptation, distribution, and caching in an integrated way by making routing decisions based on available context information, such as underlying network constraints like Quality of Service (QoS) and congestion, mobility information, user preferences, and device limitations.

Today's networking technologies consist of a broad heterogeneity of access networks, terminals, network interfaces, users, signaling and transport protocols, applications, and services. As a consequence, certain independent streams of multimedia data are required to be proactively cached, trans-coded, split, synchronized, translated, filtered, legally tapped, or transformed in some way or another before they can be delivered according to a variety of constraints, or properly displayed to the user. With today's technology, this transformation of multimedia content is generally assumed to be located at the end devices, either the user terminal or the media server. In both cases, this would lead to either quite complex user terminals or redundant content transmissions from the server. Transformation of multimedia data and possibly signaling traffic may therefore be motivated by the service provider, the network provider, or by user preferences. In most cases, it would be unreasonable to place the burden of data transformation on the client device, as mobile devices are limited by performance

constraints such as battery power, processing capability, memory capacity, available media codecs, and signaling protocols. It would also be unrealistic to expect that service providers can, or should, be responsible for performing all required transformation and adaptation operations. Thus, there is a need for network-side media processing capabilities and transformation services (which we term MediaPorts or MPs) somewhere on the media path between the sink (MediaClient, or MC) and the source (MediaServer, or MS). These MPs must be able to transform multimedia data from the MS into a form that is acceptable by the MC. This transformation takes into account the available context information for the purpose of optimal service delivery. Hence, there must be an option to take away the responsibility for the data transformation from the end users and the service providers. To this end, and to provide the flexibility to deliver multimedia content, SMART proposes the concept of Service-Specific Overlays Networks (SSONs), which enable the flexible configuration of virtual networks consisting of Overlay Nodes (ONodes) on top of the underlying physical network. This allows the transparent inclusion of network-side data processing capabilities (MediaPorts) in the end-to-end media delivery path from the MediaServer to the MediaClient. These MediaPorts can perform value-added processing, such as overlay routing, smart caching, and media adaptation among other functions. In SMART, an SSON will be created for each media delivery session or group of sessions, thus many SSONs can be created and deployed simultaneously.

1.3 Management Problems and Challenges

As described earlier, SSONs have many attractive features, but they come at the cost of increased overhead (due to the additional packet header and redundant work at the overlay and the IP layer) and complexity. Moreover, as traffic on the overlays increases (which occurs continuously), the network becomes overloaded, and its resources consumed [3]. In addition, overlays are usually designed independently, thus increasing the chances of negatively affecting each other (which will result in creating bottlenecks), and degrading their performance, as well as the underlying network performance.

Therefore, it is essential to incorporate an overlay network management mechanism that reduces the complexity of managing overlays and preserves their correct operations.

The SSON management problem is generally perceived from two, almost contradicting angles: Users' and service providers' perspective. The users' perspective of SSON management problem is fundamentally limited to the ability to access a set of services that are customized to their needs; a user basically wants to be able to get the best service quality while suffering the least possible cost. From the service providers' perspective, the problem of SSON management deals with the satisfaction of two objectives. The first is to provide users with the desired services in a timely manner, and the second is to maximize their total revenue by utilizing resources as efficiently as possible. This adds to the list of complexities in the SSON management problem, which already includes a host of three major problems: 1) The dynamic changes in network conditions and topology, which renders management information quickly obsolete. For example, network nodes may fail, links may get congested, and routing information may change over time. Moreover, changing the overlay routing path is affected by the required QoS, bandwidth, latency, and the existence of other overlays. 2) Overlay members that are also dynamic; new users may join or leave the overlay, which introduces mobility issues of users roaming across different domains and changing their point of attachment to the network as they move. They may even be serviced by different providers during one running session, but nevertheless, expect that their sessions will always be delivered regardless of their location. 3) The limited knowledge that overlay nodes have about the network (this knowledge varies between overlay members). With a big number of overlays, the task of management becomes harder to achieve using traditional methods, and therefore, new management scheme should be provisioned to overcome these challenges. As described earlier, a management scheme is an end-to-end problem that is concerned with providing users with their required services by best utilizing the available network resources. Therefore, the new management scheme should consider the different phases that overlays go through during their lifetime. Specifically, a management scheme that deals with overlay creation, optimization, adaptation, and termination is needed. Creation requires the setup

of an overlay routing table in each overlay node along the end-to-end path—a path that must be optimized to the QoS metrics. Adaptation produces a new behavior that reflects a change in the overlay environment, and may be necessary to assist mobility, to deal with the failure of an overlay node, or to control congestion. Termination involves claiming the reserved resources and updating overlay routing tables.

1.4 Motivation

SMART creates an SSON for each media delivery service or group of services, however, it does not specify the means by which SSONs are constructed and managed. Creating an SSON for each media delivery session implies that a numerous number of SSONs will co-exist and thus if left unmanaged, they will not only degrade the performance of each other but also that of the underlying network. In addition, it is essential to have suitable mechanisms to discover the required media processing functions, and to seamlessly integrate them in the multimedia delivery session. Moreover, once SSONs are created, there should be a mechanism to adapt them dynamically to the ever changing conditions of the network, users, and service providers.

Policy-based management represents one possible solution for SSONs management problem. The use of policies offers an appropriately flexible and customizable management solution that allows network entities to be configured on the fly [4], [5]. Usually, network administrators define a set of rules to control the behavior of network entities. These rules can be translated into component-specific policies that are stored in a policy repository, and are retrieved and enforced as needed. Policies therefore represent a suitable and efficient means of managing overlays. However, existing management systems usually direct the management task to the physical network entities, such as routers, switches, and gateways. Therefore, the management task to overlays and their logical elements is not considered.

Adaptive management systems represent another obvious solution to the problem of SSON management. A closer examination of existing adaptive management techniques shows that they can be classified into two distinct approaches for adaptation: 1) Adaptation with respect to the network and the operating system components, and 2) Adaptation at the application level. Adaptive applications can accept and tolerate resource scarcity by dynamically changing demands based on the availability of existing resources—apparently, applications which have strict real-time requirements do not fit in this category. On the other hand, network level adaptation solutions provide flexible means for the management of the underlying variable resources. Nevertheless, existing adaptation frameworks still have certain limitations; they usually lack an essential degree of flexibility, they are heavily dependent on decisions taken by human operators, and more complexity is added to their management functionalities.

Although active networks-based management seems to provide some promising solutions, introducing more programmability into network devices also implies adding more complexity to their management functionalities. Also, excessive utilization of active packets results in network performance deterioration due to them exhausting network resources. One solution to this problem is to restrict the functionality of the programs carried by the active packets, alas resulting in architectures with decreased capabilities. Furthermore, the dispatched active packets or programmable codes introduce new safety and security concerns.

A major limitation of most of the existing approaches arises from their static configurations, which are built a-priori by administrators into network devices. These approaches usually lack the flexibility required by SMART communication environments, and may not be sufficient to handle different changes in the underlying environments. Furthermore, with the current high competitive market of service providers, besides service quality, service cost becomes an important factor. However, the reliance on human operators is a major contributor to the current cost of services. Also, in current management systems, network reconfiguration in response to users' requests for service customization is only performed manually by a network operator.

This results in significant delays ranging from minutes to even days. Existing frameworks must be extended so that customers are able to tailor individual services to their particular requirements. Moreover, it is usually disadvantageous to limit the SSONs topologies at the time of connection establishment. Specified resource requirements do not often remain valid for the lifetime of the entire session.

The aforementioned limitations of current management frameworks represent strong motivations for the development of a novel, autonomous SSON management framework with inherent dynamic capabilities. This framework will manage, customize, and extend SSONs resources in response to the continuously changing requirements. By making the management systems more autonomous, the need for direct and continuous involvement of human operators is reduced.

1.5 Dissertation Overview

This thesis approaches the issue of SSON management from two different, though related, levels: The first, policy-based adaptation and resources discovery, is concerned with locating the required MediaPorts (MPs) in the underlying network, as well as adaptively managing existing SSONs. This facilitates the processes of creation, configuration, adaptation, and termination of SSONs based on user, network, and service provider context information. In the second level, autonomic overlays, SSON autonomic management is developed to deal with increased management complexity.

The resource discovery phase becomes particularly challenging in the case of dynamic network—the network resources and the users are also dynamic. We address this issue through the utilization of the optimal chordal ring features to build a fault resilient, scalable, and cost efficient resource discovery scheme. The adaptation is approached as a dynamic process where overlay network components are configured at run-time, rather than statically by network administrators. To facilitate this task, polices are utilized as tools to continually guide the behavior of the underlying overlay networks. This is carried out through a multi-layer autonomous framework. In the first

layer, the required overlay nodes and resources are identified and reported to the second layer, in which overlay-specific decisions and policies are dynamically generated and dispatched to the appropriate overlay policy enforcement points in the third layer. Overlay network components are then dynamically reconfigured to best utilize available resources while maintaining a smooth multimedia delivery.

As mentioned before, information technology components produced over the past decades are so complex that they increase the management challenge of effectively operating a stable environment. Overlay networks management is further increased by the huge numbers of users, terminals, and services. Although human intervention enhances the performance and capacity of the components, it drives up the overall costs, even as technology component costs continue to decline. Due to this increased management complexity, autonomic overlays were developed in the second part of this dissertation. SSONs and their constituent overlay nodes are made autonomic, and so, self-manageable. Construction, configuration, and resource discovery were achieved using self-composition, which is realized using a self-organization algorithm. The algorithm is powered by learning rules induced from biological systems, and supported by filtering rules to achieve the highest possible performance.

1.6 Summary of Contributions

The goal of this dissertation is to investigate new principles and design new models for SSONs autonomous management. The major contributions of this dissertation can be summarized as follows:

1. **An Autonomous SSONs Management Framework.**

   SSONs consist of a set of overlay nodes and links. To enable the adaptive management, we proposed extending overlay nodes to include an Overlay Policy Enforcement Point (OPEP) that communicates policy objects and requests decisions from a remote Overlay Policy Decision Point (OPDP). Both OPEP and OPDP consist of a set of agents that are used to realize their behavior.

Management actions are expressed though policies generated primarily at the OPDP, and enforced at the OPEP. Our proposal therefore is a complete design and functional specification of an autonomous SSON management framework. The framework makes use of the available context information, such as user, network, and service provider context information, to automate the creation, adaptation, and termination of SSONs [6], [7], and [8].

2. **A Semantic MediaPorts (MPs) Resource Discovery Scheme.**

   MediaPorts are essential to the construction and adaptation of SSONs because they allow the flexibility of modifying the content transparently. Discovering these MediaPorts, therefore, is an integral part of the autonomous management infrastructure, which should be scalable, efficient and accurate. To this end, a novel scheme for a semantic MediaPorts resource discovery is proposed. It is based on a widely studied family of chordal rings called the optimal chordal ring. The geographical network area is divided into a set of sub-areas. A ring connecting semantically similar MediaPorts is constructed for each sub-area. For each sub-area, one of the MediaPorts is identified as the access point for that sub-area. Access points are then connected to each other using an optimal chordal ring of degree 4. Queries are then routed on the optimal chordal ring and descended into local rings only if they can be answered in that particular ring. This preserves the geographical proximities and allows for efficient locations of MediaPorts while minimizing the query cost and response time [9], [10].

3. **An Autonomic Overlays Architecture for SSONs.**

   As illustrated, the rapid growth (in terms of size and complexity) of information technology increases the management challenges. The use of overlay networks exhibited a similar growth, and they have been widely used to implement new services which pose more challenges to their management. Due to this increased management complexity, autonomic overlays were proposed to render overlays self-manageable. SSONs and their constituent overlay nodes are made autonomic, and thus become able to self-manage, ensuring that the creation,

optimization, adaptation, and termination of overlays are controlled by policies, and thus the behaviors of the overlays are tailored to their specific needs [11], [12].

4. **A Self-organizing Composition Algorithm for Autonomic Entities.**

   A major challenge in realizing autonomic overlays is how to compose a set of autonomic overlay nodes to construct SSONs, and to achieve the system-wide goals. To address this challenge, we proposed a novel self-organized composition for autonomic entities. Overlay nodes are composed into SSONs using a self-organizing algorithm to achieve system-wide goals. Knowledge about interactions, negative and positive feedback, and orientation-based modulation learning rules induced from biological systems, are all used to enhance the composition algorithm and to guarantee a valid and an efficient composition. The algorithm is also powered by filtering rules to achieve the highest possible performance [13].

1.7 Proposed Research Objectives

The objectives of the proposed research work can be summarized as follows:

Autonomous Management: The management system has to be self-adaptable and self-reconfigurable in response to changes in the surrounding environment.

Simplify human management Tasks: By automating management systems, administrators are shielded from unnecessary details of management, and freed up to other design and development tasks.

Scalability: The performance of the management system has to be maintained regardless of the number of managed SSONs.

Maximize Resource Utilization: Similar to all management systems, the key goal of the proposed framework is to maximize the utilization of the underlying network resources.

Mobility: The management framework must minimize service disruption during mobility management operations, such as mobile users.

1.8 Organization of the Dissertation

The remainder of the dissertation is organized into the following chapters.

Chapter 2 presents and discusses essential background information on mobility management, and different standard technologies for multimedia delivery.

Chapter 3 presents related work and discusses various approaches adopted by the research community, and identifies different issues addressed by various research groups to provide autonomous management.

Chapter 4 outlines and discusses the proposed autonomous SSONs management framework; responsibilities of the different components along with their interactions are specified. Simulation results are also presented to demonstrate the performance of the proposed scheme.

Chapter 5 presents a novel scheme for a semantic MediaPorts resource discovery based on the use of the optimal chordal ring. Simulation results are also presented to demonstrate the performance of the proposed scheme.

Chapter 6 presents a novel architecture for autonomic overlays. Autonomic entities are driven by policies. This ensures that the SSONs are created, optimized, adapted, and terminated by policies, thus achieving their specific needs.

Chapter 7 presents a novel, self-organizing composition algorithm. Autonomic overlay nodes were built into SSONs by utilizing a self-organization algorithm. Learning and filtering rules were utilized to increase the performance of the algorithm. Simulation results are also presented to demonstrate the performance of the proposed scheme.

Finally, Chapter 8 summarizes the presented contributions, and discusses directions of future research work.

Chapter 2

Background

In its current (and original) architecture, the internet was designed for wired links and fixed end systems, without explicit support for mobile nodes or wireless connections. The wide usage of mobile devices and the increasing popularity of wireless communication links essentially give rise to varying link conditions, multi-homed devices, and handovers between physical access nodes, thus affecting the network infrastructure and introducing new challenges that must be addressed. To optimize the quality of communication, end-to-end connections will have to be adapted to actual link conditions and user preferences. Also, dynamic handovers will have to be realized in a seamless and secure way. At the user end, he/she needs to be supported by self-configuring and self-managing devices and networks in order to achieve optimal performance in mobile and wireless environments.

This chapter is organized as follows: The mobility management problem, existing and ongoing research in mobility management are first presented in Section 2.1. Models and standardization efforts that have been proposed for multimedia delivery are reviewed in Section 2.2. Section 2.3 discusses multimedia delivery framework in SMART. Finally, Section 2.4 concludes the chapter with a discussion that summarizes existing contributions and identifies some open issues.

2.1 Mobility Management

Wireless technologies have become characterized by rapid advances, seeing enabled access at several levels such as personal area networks (PANs), wireless LANs and WANs, and cellular and satellite networks. This has led to the emergence of new

network types and services—albeit complicating the challenges of heterogeneity and interoperability mechanisms—which would enable the mobile end user to seamlessly traverse different networks while maintaining Internet connectivity.

This section presents an overview of the current mobility management solutions, and investigates the different IP stack layers including application layer, network layer, and hybrid mobility solution.

2.1.1 Definition

Mobility Management is a communication scheme that enables the underlying network to deliver multimedia contents and calls to the roaming entities, regardless of their current points of attachments. In the mobile environment, an entity could be a laptop, a desktop computer, a wireless device, or any other computing device.

Mobility management consists of *location management* and *handoff management* [14]. Location management allows the network to locate the Mobile Entity's (ME) current location by providing the means that allow the ME to announce its current location, and periodically updating the ME's location profile, which will be queered by any entity wishing to contact the ME. Handoff management maintains the ME's connection during its movement around the network, which might involve a new connection generation in the new subnet and the packet flow management for ongoing calls or sessions.

There are two types of movements for MEs: *1) Inter-domain*, and *2) Intra-domain roaming* [15]. The latter refers to the movement of the ME between different domains of the same system, which implies that mobility management is based on similar network interfaces—handoff management is called Horizontal Handoff. *Inter-domain* roaming, on the other hand, refers to the movement of the ME between different backbones, protocols, technologies, or service providers—handoff management is called Vertical Handoff, and can be further classified into soft or hard. In soft vertical handoff, the new location and the old one handle the interchange between them while

performing the handoff; the handoff is achieved by proactively notifying the new location before the actual handoff takes place, thus minimizing packet loss but introducing delay. In hard handoff, the ME moves to the new location, and from there tries to re-establish the connection; consequently, the connection may be off for a small period of time during the move, however, the delay and signaling are less than those of the soft handoff [16].

In general, a mobility management scheme/protocol usually supports one or more of the following mobility types:

Terminal mobility: The ME is reachable regardless of its current location, i.e. the ME is allowed to move between different sub-nets and, at the same time, being always reachable for incoming calls. It is important to maintain the session during the sub-net change.

Personnel mobility: The user is able to access his/her services regardless of location and terminal being used. In that course, the user is allowed to have different terminals and will be reached at any of them or at all of them. Also, the user may have more than one address and any of them may be used to reach the user's active terminal.

Session mobility: The user is able to continue a session (or part of a session) even while changing terminals. For example, a user may want to continue a session that had initially been started at his/her PDA on an office desktop computer when entering his/her office [17].

Service mobility: The user is allowed to access his/her services while roaming or changing devices. For example, a user may want his/her buddy list, address book, and call logs to be accessible from any terminal; the user must have the ability to alter these services from any terminal.

2.1.2 Network Layer Mobility Schemes

Network layer solutions provide mobility-related features at the IP layer. They do not rely on or make any assumption about the underlying wireless access technologies [18], [19]. Signaling messages for mobility purposes are carried by IP traffic.

Mobile IP (MIP) [18], [19] is a standard protocol proposed by the Mobile IP Working Group of the Internet Engineering Task Force (IETF). It utilizes special mechanisms to offer continuous media support when MEs change their locations. Each ME has two addresses, the Home address and a Care-Of Address (COA). The former is a static address, and is used by any entity wishing to contact the ME. The latter is dynamic, i.e., it represents the current location of the ME, and is assigned to the ME whenever it connects to another network. The ME has a Home Agent (HA) in the home network, and whenever it connects to another network, it will register with a Foreign Agent (FA) to obtain a COA. The COA may be the IP address of the FA (in which case, it will be called a co-located COA), or it may be obtained from a separate entity, e.g. a Dynamic Host Configuration Protocol (DHCP) server. Any entity wishing to contact the mobile entity is called a Correspondent Entity (CE) (which might be a mobile entity); A CE does not need to have any mobile IP knowledge at all.

The handoff procedure is carried out whenever a ME moves from one domain to another. The ME obtains a new COA when it enters the new domain and registers it with its HA. The HA sets up a tunnel to the COA, using it to deliver packets to the ME.

MIP does address the terminal mobility problem, but it does not, nor do its related schemes by themselves, support device-independent persona mobility, or session and service mobility. In an effort to remedy that, two common versions of Mobile IP have emerged, version 4 (IPv4) and version 6 (IPv6). IPv6 solves the shortage of address in IPv4, with the issue of mobility having been considered from the start.

MIP suffers from a set of drawbacks: 1) Due to tunneling, routing in mobile IP is inefficient; it is also asymmetric as the ME directly contacts the correspondent entity. A set of route optimization (MIP-RO) [20] techniques have been proposed as a solution, but they require the CE to be modified in order to understand binding updates–binding

updates inform the CE of the COA of the ME and hence the CE can tunnel packets to the COA without going through the HA. However, the correspondent entity must use triangular routing until it receives the binding update from the HA; reverse tunneling has been proposed to solve the problem of asymmetry [21]. 2) Firewalls cause security problems as they block traffic arriving from different sub-nets. Thus, the mobile entity will not be able to send the registration information to the home agent while it is roaming in a different network. [22]. 3) Tunneling the packets from the home agent to the mobile node causes an extra overhead; also the registration process causes an extra overhead. 4) Handoff latency problems which are caused by the long latency in the communication path between FA and HA as each time the ME changes its location, it has to re-register the new care-of address with the HA. This problem can be solved using a micro-mobility scheme, such as, Hierarchical Mobility Agent schemes, and Host Based Routing schemes (HBR) [23]. The Hierarchical Mobility Agent schemes (e.g. hierarchical Mobile IP (HMIP) [22], MIP with Regional Registration [24] (MIP-RR), and intra-domain mobility management protocol (IDMP) [25] and TeleMIP [26]) exploit the hierarchy of the network to reduce the signaling between the mobile entity and the home agent and thus achieve faster hand-off, but they suffer from a scalability problem. On the other hand, the HBR schemes (e.g. CIP [27] [28], HAWAII [29], and MMP [30]) are more flexible and can be integrated with different macro-mobility management schemes, like SIP and MIP. They also offer the lowest latency networking re-routing solution for micro-mobility management as they take an optimal path to the closest node that should handle both the location and route updates. A comparison between these different protocols is presented in [14] and [15].

2.1.3 Application Layer Mobility Schemes

Application layer mobility can be used to solve the problems inherent in mobile IP. The Session Initiation Protocol (SIP) [31] is an IETF signaling protocol that allows users to establish, modify, and terminate a session consisting of audio, video, or any internet communication mechanism. SIP is a text-based protocol that is similar in both syntax

and semantics to the Hyper Text Transport Protocol (HTTP); the difference is that SIP can use any transport protocol in combination with its different logical entities (proxy, redirect, etc...) to ensure request reliability. SIP is an application layer protocol independent from packet layer, and supports both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). SIP can be integrated with other protocols to support more functionality, such as Session Description Protocol (SDP) for delivering multimedia sessions, and Real-Time Protocol (RTP) [32] for transmitting real-time data. In addition to integration with other IP components, SIP has been recognized for its simplicity, programmability, modularity, and extensibility [33] [34].

SIP consists of the following entities: 1) *SIP User Agent (UA)*: This is the end point that acts on behalf of the user; it is either a User Agent Client (UAC) that initiates requests, or a User Agent Server (UAS) that responds to requests. UAs communicate with each other directly or via another entity, like a proxy server or a redirect server. 2) *SIP Proxy Server*: This entity's main functionality is to forward incoming request to another server. A SIP proxy server can be either state-full or stateless; a state-full proxy maintains information about the request and all the responses that indicate the progress, in addition to the final response that indicates whether the request has been successful or not (collectively called a transaction). A stateless proxy does not maintain any information about the request; it just forwards the request to another server. If a state-full proxy does not know the final destination of the request, it can *fork* the request by sending a copy to each possible destination, either in parallel or sequentially. 3) *SIP Redirect Server*: This simply returns to the requestor the address of the destination server, so that the UA (requestor) can contact the destination server directly. 4) *SIP Registrar*: This maintains the location information of the SIP users. SIP proxy and redirect servers regularly contact the registrar to know the existence of a SIP UA's address that will help in establishing a session between two parties. SIP proxy, redirect, and registrar are logical entities that may co-exist together in the same server.

When the mobile entity changes its location, a registration process occurs to inform the home registrar about the new point of attachment. To continue ongoing sessions, the

mobile entity sends a RE-INVITE request to the correspondent entity, informing it with the new point of attachment. During the registration and the RE-INVITE, all data packets sent from the correspondent entity will be lost, thus it is highly desirable to reduce the packet lose as much as possible.

2.1.4 Hybrid Approaches

There are two different kinds of data that the ME and the CE might exchange. The first is the non-real-time traffic that has been usually carried over TCP; and the real-time traffic that must be carried over RTP/UDP. The two data types differ from each other in their delay and loss characteristics [35].

Different management schemes have emerged to support mobility for real-time and non-real-time traffic. SIP for example, basically supports multimedia real-time traffic, but does not support non-real-time traffic [36], as it breaks the TCP connection. Mobile IP, on the other hand, is more suitable for non-real-time traffic. Based on these facts, there have been attempts to combine both MIP and SIP (Hybrid) to support mobility for all kinds of traffic; however, MIP and SIP are not suitable for intra-domain mobility (mobility in the same domain). These approaches are called multi-layered as they combine both the network layer and the application layer to support mobility.

In [37], a pure SIP approach is proposed, where SIP signaling is used to support macro-mobility, and Hierarchical Mobile IP (HMIP) or Cellular IP (CIP) is used to support micro-mobility as both provide faster handoff mechanisms [38]. Encapsulation is introduced to prevent the TCP session from breaking. The encapsulation takes place in both the ME and the correspondent entity. If the session is real-time, then no encapsulation is required. This technique clearly has the disadvantage of requiring the ME and the correspondent entity to have encapsulation capabilities. As an alternative solution, the authors propose a second approach that is similar to the first one in that it uses either HMIP or CIP to support faster handoffs, while the inter-domain mobility is supported by both MIP and SIP. SIP is responsible for real-time traffic, MIP for non-real-time. The second technique uses the tunneling capabilities of MIP to deliver data

packets from the correspondent entity to the mobile entity, and therefore entails the main problems that MIP suffers from. Although faster handoff techniques were used, there is a signaling problem, as the mobile entity has to register its new location with both its home agent and the home SIP registrar.

As in [37], [35] proposes a new mobility management scheme for wireless IP networks that handles real and non-real time traffic. SIP is used to handle Macro-mobility for real-time traffic, and MIP-LR (mobile IP with location registrar) is used for non-real-time traffic. In both cases, MMP (Micro-mobility management) is used to handle micro-mobility. The difference between [37] and [35] is that the former uses MIP to handle macro-mobility for non-real-time traffic, while the latter uses MIP-LR for the same task. They also differ in how they integrate SIP with MIP-LR; in [35] a policy table is used. Based on the policy table an entity (between the IP level processing and the link layer processing) examines each packet and sends it to the suitable handler. To handle terminal mobility, a SIP Re-INVITE message will be sent to the CE whenever the ME changes its location, and for non-real time traffic, an update message will be sent to the CE and to the HLR (Home Location Registrar). To handle micro-mobility, micro mobility schemes [39] in addition to SIP are used. While in [37], data packets from or to the mobile entity are separated at the domain edge routers.

2.2 Standard Technologies for Multimedia Delivery

Today, there already exist a number of different solutions for providing multimedia services. Of these solutions, there are few technologies that can generally be regarded as standards for multimedia delivery; in this section we shed light on these technologies.

2.2.1 Session Initiating Protocol (SIP)

SIP is an application-layer control protocol for creating, modifying, and terminating sessions with one or more participants. As pointed out in the previous section, SIP is widely used to provide session control for real-time communications. For example, all

multimedia communications in the IP-based Multimedia Subsystem (IMS) of today's 3G networks are based on SIP. While SIP is a very flexible and powerful technology, it was designed with mainly end-to-end usage in mind; adaptation of media content to meet user demands is generally assumed to occur at the end devices, and introducing dedicated adaptation components can only be achieved through non-standard approaches. Further, SIP was mainly designed to support point-to-point communication; supporting multi-point communication increases the complexity of the protocol considerably, and requires additional non-standard capabilities at the SIP servers. Finally, support for peer-to-peer or content distribution networks is completely non-existent in SIP. Besides those application level shortcomings, SIP has only very simple support for mobility, thus causing long handover periods. SIP can only react to application level triggers for controlling the communication session, thereby, effects of network load or failures are completely ignored by SIP.

2.2.2 IP-based Multimedia Subsystem (IMS)

The 3GPP IP Multimedia Subsystem (IMS) is the first platform standardized towards network-independent access and session control [40], [41]. IMS uses SIP for initiating, modifying, and terminating IP-based multimedia sessions. The goal of IMS is to provide service providers with a platform that facilitates the provision and management of a wide range of services. The success of service providers using IMS and consequently, the success of IMS as a whole, depends on how important those IMS services are to users. IMS is developed for person-to-person multimedia connections in Universal Mobile Telecommunications System (UMTS)-networks, but the use of IMS is not limited to UMTS environments. More generally, IMS can provide IP-based multimedia services over any packet-switched network. While IMS is based on SIP to a large extent, it does provide various improvements to enable support for broadcast communication and better support for mobility. By closely integrating the concepts of application servers, IMS already provides the basic requirements for enabling the integration of intelligent services into the communication sessions. However, the current

version of IMS's specifications still does not support mid-session macro-handover. In other words, whenever a node changes its global IP address (typically the case when a node connects to another access network), the ongoing session has to be terminated, and the long standard SIP-based IMS session setup procedures have to be performed once more at the new access network. Those time-consuming procedures may imply long perceivable disruption times at the application layer, which is not acceptable for delay-sensitive, real-time services [42]. Moreover, similar to other SIP-based solutions, support for dedicated adaptation components is still lacking. Further, while the concept of conferencing and multi-party session is closely integrated into IMS, there is no adequate support for the routing of flows of the same session over different paths.

2.2.3 Multicasting Protocols

Content Distribution Networks (CDNs) act as trusted overlay networks that offer high-performance delivery of common web objects, static data, and rich multimedia content by distributing content load among servers that are close to the clients [43][44]. CDNs can improve access to content that is typically un-cacheable by caching proxies, including secured content, streaming content, and dynamic content [45]. Different multicasting protocols together with caching technologies are commonly used in CDNs for the purpose of distributing multi-format rich media services. CDNs normally consist of integrated distribution, streaming, security and traffic management solutions to enable a variety of high-bandwidth broadband applications such as Video-on-Demand (VoD), web casting, interactive television, e-learning, and others. Similar to SIP and IMS, the concept of CDN was designed with a single application in mind, namely efficient transport of media data. Current CDNs lack the intelligence needed, not only for transporting media, but also for adapting it to the network load situation or for supporting user mobility and preferences.

2.3 Smart Media Routing and Transport (SMART)

As pointed out, current approaches for media delivery are not sufficient for the purpose of providing network-side media processing capabilities on the media path. Therefore, a SMART framework is being developed to achieve these goals in the context of Ambient Networks. The overall goal of the Ambient Networks Integrated Project [46] is to develop a vision for future wireless and mobile networks. The aim of this project is to create an innovative, industrially exploitable new inter-networking framework that is based on the dynamic composition of networks. A key aspect of the project is to establish a common control layer for various network types, which provides end users with seamless multi-access connectivity to enable selection of the best available network. For an operator, the Ambient Network concept allows flexible and dynamic network configuration and management.

In the environment targeted by Ambient Networks, there will be a broad heterogeneity of access networks, terminals, network interfaces, users, signaling, and transport protocols, applications, and services. As a consequence, certain independent streams of multimedia data may be required to be pro-actively cached, trans-coded, split, synchronized, translated, filtered, legally tapped, or transformed in some way or another before they can be delivered according to a variety of constraints, or properly displayed to the user. To this end, Smart Media Routing and Transport (SMART) architecture [2] has been proposed to enable the seamless integration of next-generation multimedia services into Ambient Networks.

2.3.1 Media Processing Functions

Services, as defined in the SMART-context, can be simple requests of information (web browsing), multimedia streaming (audio and video), and/or conferencing, or they can be more complex service scenarios including mobility features, media adaptation features, caching features, and so on. Media that is delivered as part of SMART-like services may need to be processed along the media path and thus inside the network (e.g.,

dynamic trans-coding of video and audio streams to adapt to changing link properties, or proactive smart caching following user movement). Since services like media adaptation and trans-coding can only be located at the end systems today, they are often of very limited value. In the case of server-side adaptation, the media has to be transmitted several times (once for each type of encoding). Client side adaptation, on the other hand, has the drawback of wasting network resources (as the 'down scaling' of the media format is only done at the client end), and increasing the complexity (and hence the cost) of user terminals.

Other services, such as caching or optimal routing of media traffic in order to optimize the possible achievable QoS, can only be achieved using network side intelligence. Similar reasoning can be used for broadcasting and multi-party communication. Only with the help of network side components is it possible to optimize the bandwidth usage. Therefore with SMART, additional intelligence can be located at the provider and inside the network. Examples of such intelligence include the following features: Media routing and media adaptation to deal with terminal and user mobility; media splitting to enable session/flow mobility; synchronization for re-combining split flows; smart caching for accommodating low bandwidth access networks.

In SMART, multimedia transformation is carried out by network-side media processing capabilities and transformation services [47], termed MediaPorts (or MPs), which are located somewhere on the media path, between the sink, called MediaClient (or MC) and the source, called MediaServer (or MS). MPs must be able to transform the multimedia data originating from the MS into a form that is acceptable for the MC.

2.3.2 Overlay Routing

The concept of overlay networks is promoted by SMART in order to enable inclusion of the above mentioned media processing functions in the end-to-end media delivery path in a way that is transparent to the underlying network (i.e. without the need to replace the existing infrastructure) as well as to the end-user applications. Consequently, the

migration path from legacy networks towards SMART-enabled Ambient Networks is expected to be inexpensive and straightforward. One of the important advantages of the overlay concept is that it enables the establishment of different types of overlay networks as needed. This allows, for example, for tailoring the virtual addressing scheme and the overlay routing to best suit the requirements of a particular service. Another example of the tremendous capabilities of overlay routing include more advanced multimedia transport techniques that enable transparent integration of value-added media processing capabilities into the end-to-end media delivery path. Because of such advantages, the overlay concept has been selected as the basic building block for the SMART framework.

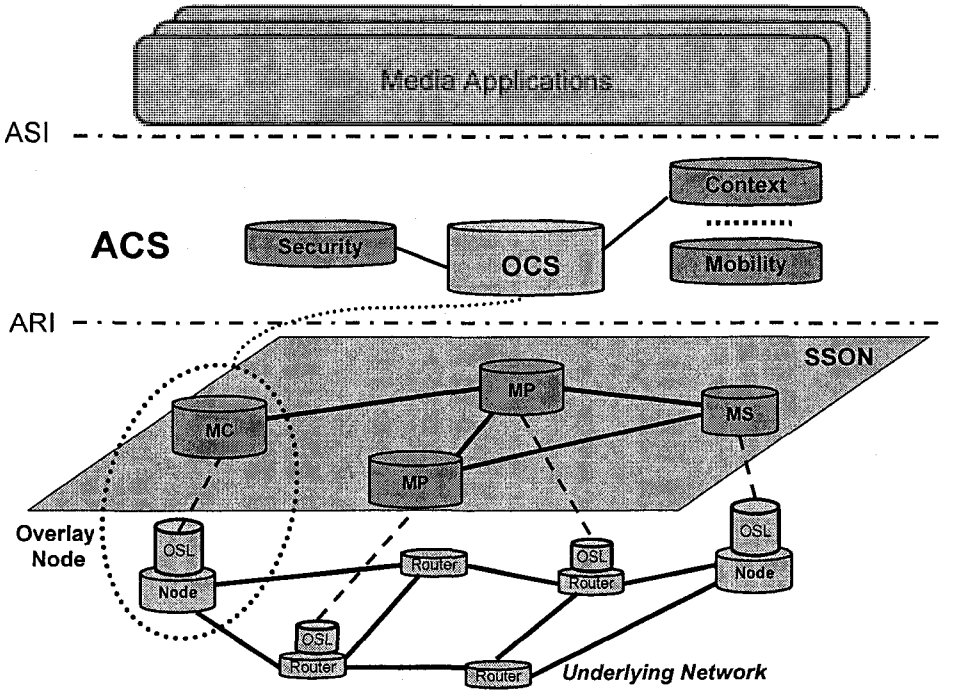

Fig. 2.1 SMART Architecture within the Overall Ambient Networks Architecture

2.3.3 Service-Specific Overlay Networks

A Service-Specific Overlay Network (SSON) is defined through the set of Overlay Nodes (ONodes) that are part of a particular service (or collection of services that are

combined to one composed service) and the virtual links that connect the individual ONodes to each other. In SMART, a different virtual network is deployed for every media delivery service (or group of services), which allows for the configuration of appropriate, high-level routing paths that meet the exact requirements (for example, QoS, media formats, responsiveness, cost, resilience, or security) of a media service. Moreover, the exploitation of overlay network techniques also facilitates the transparent inclusion of network-side media processing functionalities (such as caching, adaptation, and synchronization) into the end-to-end data paths. Besides, the overlay network is able to react dynamically to a changing environment, that is, modifications in the overlay might be triggered due to changes in user preferences, mobility, QoS, or the underlying network. Finally, to provide maximum flexibility, SMART supports all these actions separately for each flow of the media service within a SSON.

Fig.2.1 (redrawn from [2]) illustrates how the SMART architecture relates to the overall Ambient Network architecture. The figure also shows the Ambient Control Space (ACS) as well as its control interfaces, namely the Ambient Service Interface (ASI) and the Ambient Resource Interface (ARI). Roughly, the ASI provides the service and user profile to the Overlay Control Space (OCS) in case of a request for a media delivery service. The ARI is the interface to the connectivity layer, and manages the underlying connectivity resources.

2.3.3 Overlay Node (ONode) Architecture

An ONode is a specialized Ambient Network node that implements the functionality required to join the SSONs by, for example, provisioning network-side media processing functionalities, such as caching, media adaptation, synchronization, and Media aware inside the network. ONodes (see Fig 2.2, redrawn from [2]) can be described from the user perspective and the control perspective. For each SSON of which the ONode is part of, MediaPorts (MPs) are instantiated. MPs are responsible for Media Routing in the control plane and, in the user plane, host the so-called application modules, each responsible for a particular network-side media processing functionality.

Furthermore, and depending on the required media processing functionality, overlay nodes can take one or more of the roles of MC, MS, and MP. Note that a physical ONode can be part of many SSONs at the same time.

The *control plane* of the ONode includes the ONode Control entity, which is responsible for the general management of the ONode and the signaling exchange. The ONode Control consists of several components, which can be classified into those that deal only with the local control and management of the ONode, and those that logically belong to the OCS, which is the Functional Entity residing in the ACS that controls the SSONs on Ambient Network wide basis.

The *user plane* of the ONode encompasses the Overlay Support Layer (OSL) and the application modules that take part in media processing actions. The OSL sits on top of the underlying network; it embodies the basic overlay network functionality required in every ONode for the handling of packets at the overlay level. As such, the OSL is responsible for the sending, receiving, and forwarding of SSON-level packets. The OSL provides a common communication abstraction (overlay level network protocol and addressing) to all ONodes of a SSON, so that they can communicate with each other independent of their differences regarding the underlying protocol stacks and technologies. On top of the OSL, and using its services, there are application modules that implement the behavior of a MC, MS, or MP in regard to data handling. MCs act as data sinks and send the multimedia data to the end-point media applications; whereas MSs act as data sources and receive the multimedia data from the end-point applications.

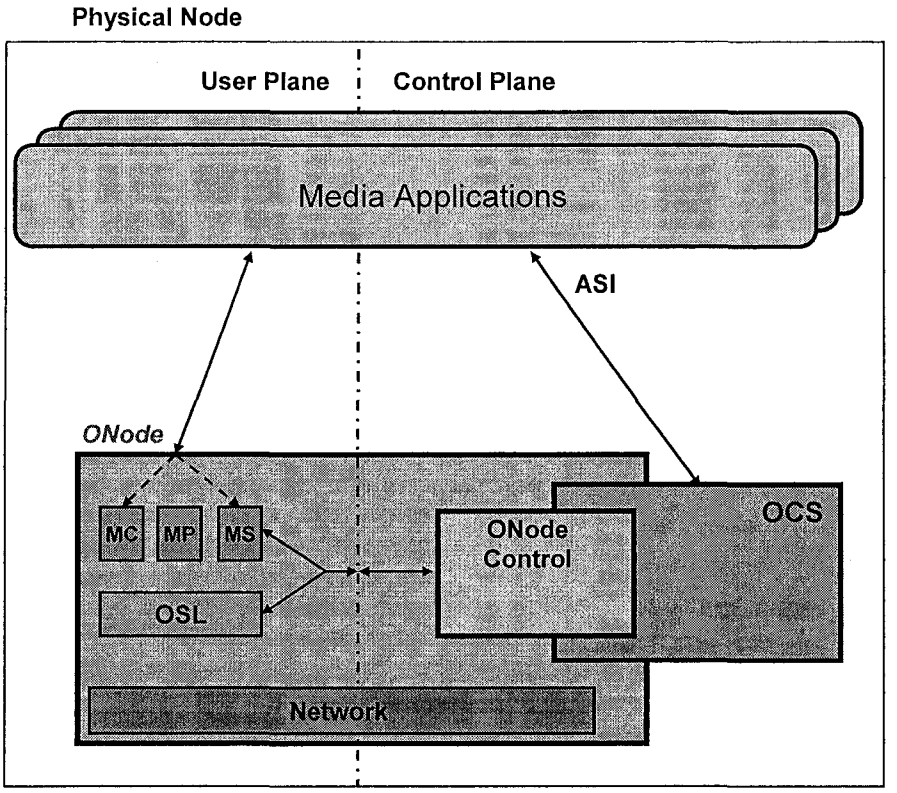

Fig. 2.2 Implementation of an ONode on a Physical Node

2.4 Summary

In this chapter, we discussed various approaches that have been proposed to address the issue of mobility and multimedia delivery. The development of mobility management schemes such as Mobile IP (MIP) and Session Initiation Protocol (SIP), and the multimedia delivery schemes such as IP-based Multimedia Subsystem (IMS) has enabled mobile users to seamlessly traverse different networks while maintaining their connectivity to their home network, and continuing their sessions at their new locations. Nevertheless, it has been generally difficult to support all types of mobility, and still deliver multimedia to users without relying on network side functions that has the ability to adapt media to meet the user's, network's, and service provider's needs.

In order to satisfy the contradictory needs of different applications and services, Smart Media Routing and Transport (SMART) introduced the concept of Service-Specific Overlay Networks (SSONs). This multimedia delivery method enables the flexible configuration of virtual networks on top of the underlying physical network infrastructure. SSONs have the ability to customize the virtual network topology, and the addressing as well as the routing at the overlay level according to the specific requirements of a media delivery service. In addition to that, SSONs transparently include network-side functions into the end-to-end communication path from the MediaServer (MS) to the MediaClient (MC), thus making it possible to support media routing, distribution, adaptation, and caching over complex communication mechanisms like peer-to-peer communication, multicasting, and broadcasting.

However, SMART does not specify the means by which SSONs are constructed and managed. Creating an SSON for each media delivery session implies that a numerous number of SSONs will co-exist and thus, if left unmanaged, they will not only degrade the performance of each other, but also of the underlying network. In addition, it is essential to have suitable mechanisms to discover the required media processing functions, and to seamlessly integrate them in the multimedia delivery session. Moreover, once SSONs are created, there should be a mechanism to adapt them dynamically to the ever-changing conditions of the network, users, and service providers.

Chapter 3

Related Work

Requirements posed by autonomous overlay management cause certain problems to emerge; problems that our architecture proposes to resolve, but we still need to characterize these problems, and that is what we do before presenting our solution. To achieve that, we present a survey of current research efforts related to overlay management in this chapter, which is organized as follows: Various definitions and different overlay networks used in literature are first presented in section 3.1. Some of the overlay management models and standardization efforts that have been proposed are then highlighted in section 3.2. Section 3.3 reviews existing research work that has been carried-out in the area of resource discovery. Finally, section 3.4 summarizes and concludes the chapter.

3.1 Overlay Networks

An overlay network is a virtual network of nodes and logical links that is built on top of an existing network, with the purpose of implementing a network service that is not available in the existing network. For example, overlays can be used to increase routing robustness and increase security, reduce duplicate messages through multicast, and provide new services for mobile users. They can also be incrementally deployed on end hosts without co-operation from ISPs, and without the need to deploy new equipment or modify existing software/protocol [48], [49], and [50]. Frameworks that have been developed for this purpose fall mainly into one of two configurations: Static and Automatic. They can be further classified into Application Specific Overlay Networks and Generic Supporting Diverse Applications. Moreover, overlays can be layered—one kind of overlay built on top of another. An overlay network is thus an application layer

internet which separates the physical layer from the applications, and supports customization to meet and optimize specific functionalities. Peer-to-peer networks are a common example of overlays.

3.1.1 Application Specific Overlay Networks

Application specific overlay networks have been tailored to a specific application. Such as multicasting [51], [52], content distribution networks [53], and peer-to-peer file sharing [54]. Application layer multicasting focuses greatly on using strategically placed fixed nodes to support overlay multicast service. Overcast [49] provides wide-area content distribution and bandwidth sensitive multicast services while utilizing the network bandwidth efficiently. Resilient Overlay Network (RON) [48] is based on strategically placed nodes in the Internet domains. It is proposed to quickly detect and recover from path outages and degraded performance. However, RON is designed for applications with a small number of participating nodes and cannot be scaled to a big number of nodes. In [55], overlays are used to achieve fast fail-over and traffic load balancing in the Border Gateway Protocol (BGP). A set of policy agents installed in each participating autonomous system enforces necessary changes in the local BGP. The policy agents communicate through the overlay. Our work differs from these approaches in that it allows (a) a number of overlays to be managed at the same time, and (b) policies to be generated dynamically from the context information. Peer-to-peer networks are another example of application-specific overlay networks. It is primarily used for resource discovery and can be classified into structured and unstructured overlays. Because of its importance, we devoted Section 3.3 to discuss it.

3.1.2 Generic Overlay Networks

In generic overlay networks, knowledge is shared through an intermediate layer that measures a number of network properties. In [56], an underlay with a multi-tier overlay routing scheme is proposed. AS-level Internet topology and routing information is

acquired by a topology-probing kernel from nearby BGP routers, thereby overlay services can share this information without the need to individually probe the internet. A more generic approach is described in [57], where a number of quality metrics (such as low latency, low hop count, and high bandwidth) are acquired from end-to-end network measurements, and used to construct overlays.

Yoid [58] is a generic overlay architecture which is designed to support a variety of overlay applications that are as diverse as net-news, streaming broadcasts, and bulk email distribution. Another similar effort is the Planet-lab [59] experiment that aims at building a global test-bed for developing and accessing new network services. A similar approach was proposed in OPUS [60], which provides a large scale, common overlay platform and the necessary abstractions to service multiple distributed applications. It automatically configures overlays nodes to dynamically meet the performance and reliability requirements of competing applications. X-Bone [61] is a system for automated deployment of overlay networks. It operates at the IP layer and is based on IP tunnel technique. Its main focus is to manage and allocate overlay links and router resources to different overlays and avoid resource contention among the overlays. OverQoS [62] can be employed to provide Internet QoS such as differentiated rate allocations, statistical bandwidth, and loss assurance, and can enable explicit-rate congestion control algorithms. Third-party providers can utilize OverQoS to provide QoS services to the customers using Controlled Loss Virtual Link (CLVL) technique, which ensures that the loss rate observed by aggregation is very small as long as the aggregate rate does not exceed a certain value. Service Overlay Network (SON) [63] is designed to use overlay technique to provide value-added Internet services. A SON can purchase bandwidth with certain QoS guarantees from ISPs, and use that bandwidth to build a logical end-to-end service delivery overlay. The authors have formulated the problem of QoS provisioning by considering various factors like SLA, service QoS, traffic demand distribution, and bandwidth cost.

Although generic overlay networks are efficient in reducing the cost of acquiring the shared knowledge, they lack the flexibility to support specific application overlay

networks. Moreover, they do not take into account specific demands for individual services such as user or terminal mobility. More importantly, they do not explicitly address the use of policies to configure overlays dynamically, which our work does in addition to addressing the use of intelligent network side functions in the overlay path, which permits additional services to be deployed.

3.2 Overlay Networks Management

The overlay's attractive benefits come at the cost of increased overhead and complexity. Overhead is increased because of the additional packet header and the redundant work at the overlay and IP layers. The constantly increasing traffic carried by the overlays also tends to overload the network and consume its resources [3]. In addition, overlays are usually designed independently, which increases the chances that they will negatively affect each other: Bottlenecks are created, reducing the performance of both the overlays and underlying network. Overlays therefore need to incorporate a management mechanism that reduces this complexity and hence keeps them operating correctly.

Overlay management is challenging for several reasons. First, the dynamic changes in network conditions and topology quickly renders management information obsolete. For example, network nodes may fail, links may get congested, and routing information may change. In addition, any changes in the routing path are affected by the required QoS [64], bandwidth, latency, and the existence of other overlays. Second, overlay members are dynamic, as new users may join or leave. Finally, each overlay node possesses limited knowledge of the network with that knowledge varying among overlay members, and with a large number of overlays, management by traditional methods becomes harder to achieve. Moreover, the management scheme must account for the different phases that overlays go through during their lifetime: Creation, optimization, adaptation, and termination. Creation requires the setup of a routing table in each overlay node along the end-to-end path—a path that must optimize the QoS metrics. Adaptation produces a new behavior that reflects a change in the overlay

environment. Adaptation may be necessary to assist mobility, to deal with the failure of an overlay node, or to control congestion. Termination means claiming the reserved resources and updating routing tables.

Since our focus is on Service Specific Overlay Networks (SSONs), it should be noted that these networks pose additional challenges. In large distributed and heterogeneous networks, media content usually requires adaptation before it is consumed by clients. For example, video frames must be dropped to meet QoS constraints: A client with a PDA requires a scaled-down version of the video; a mobile user requires the content to be cached for viewing. When SSONs are used, a first step in any of these applications is for them to learn that the services exist. In other words, they need to know "what are the services needed?", "where are these services located?" and "how are they found?" This is clearly a resource discovery problem.

3.2.1 Policy-based Management

Policy-based management has been introduced as an efficient solution for managing network entities. The use of policies offers an appropriately flexible and customizable management solution that allows network entities to be configured on the fly [4], [5]. Usually, administrators define a set of rules to control the behavior of network entities. These rules are translated into component-specific policies that are stored in a policy repository, to be retrieved and enforced as needed. Policies have been widely supported by standard organizations such as the IETF and DMTF to address the needs for network management. It was first introduced by Sloman [65] as a tool for management. His work was the trigger for other research activities focusing on policies: Sloman's work introduced policies and illustrated the power of this concept particularly in the context of distributed systems. However focus was put on the general aspects of policies such as Policy Specification [66], Conflict Analysis [67], Policy Domains [68], and Hierarchies [69]. Policies were mainly used for specific applications in networks [70] and Collaborative Systems [71].

The Policy Working Group [72] is chartered to define a scalable and secure framework for policy definition and administration. This group has defined a framework for policy-based management that defines a set of components to enable policy rules definition, saving, and enforcing. In the IETF model, the policy management system consists of a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP) [73]. The PDP evaluates the request sent by a PEP as a result of policy event against a corresponding set of policy rules. The policy decision is then sent back to the PEP using a communication protocol such as Common Open Policy Service (COPS) [74]. Figure 3.1 depicts the policy-based network management architecture defined within the IETF and DMTF framework, and used as the primary policy architecture by many research and commercial communities. The PEP is a network entity where the policy is enforced. Enforcement of policy decisions is carried out by the specific hardware/software features residing in the device such as packet filtering, marking, shaping, policing, bandwidth reservation, etc... A PDP retrieves policies from the policy repository, makes decisions based on retrieved information, and translates them into device specific configurations. These configurations are then sent to the PEP at the network entity. The policy management tool is the interface between the network administrator and the system. It allows administrators to specify policies to be enforced in network entities, and then translates them into a format compatible with the policy repository. The policy repository is a database that stores policies provided by the policy management tool, which in most cases is a Lightweight Directory Access Protocol (LDAP) directory. However, most implementations use static policy configurations built a priori into network elements. This may not be sufficient to handle changes.

Peer-to-peer systems construct an overlay to allow resource sharing; therefore they are designed with a specific application in mind. In [75], policies are used to control the topology growth of peer-to-peer systems. Policies are distributed to all hosts in the system with each host able to adopt only one policy at a time. But human interaction is still required to define the policies, and to inject them into the system. In [76], peer-to-peer concepts are used for wearable mobile devices to protect users from one another. A policy client resides in the kernel of the system, as well as a policy manager that stores

and dispatch policies. Unfortunately, the policies are static and built a priori. [55] proposes another application-specific overlay network to achieve fast fail-over and traffic load balancing in Border Gateway Protocol (BGP). A set of agents is installed in each participating autonomous system to enforce necessary changes to the local BGP. An overlay is constructed between the policy agents to facilitate their communication.

Our work differs from those specific application overlay networks in that it allows many overlays to be managed at the same time. Moreover, peer-to-peer policies so far are static and lack flexibility, while polices in our work are generated dynamically.

In the proposed work, we envision policies as a very powerful tool that can be used in automating the management of Service Specific Overlay Networks (SSONs). Policies are persistent; once a policy is applied, it remains active during its lifetime. Moreover, changing system behavior without modifying underlying software/hardware can be easily accomplished by changing the previously applied policies or by enforcing a new set of policies. Existing management systems usually direct the management task to physical entities such as routers, switches, and gateways. In our proposed scheme, the task is assigned to the overlays and their logical elements. This furthers the use of policies by automating the creation, assembly, and selection of the applied policies at a given instance of time, thereby generating policies dynamically and automating the adaptation in the behavior of the overlays without human interaction.

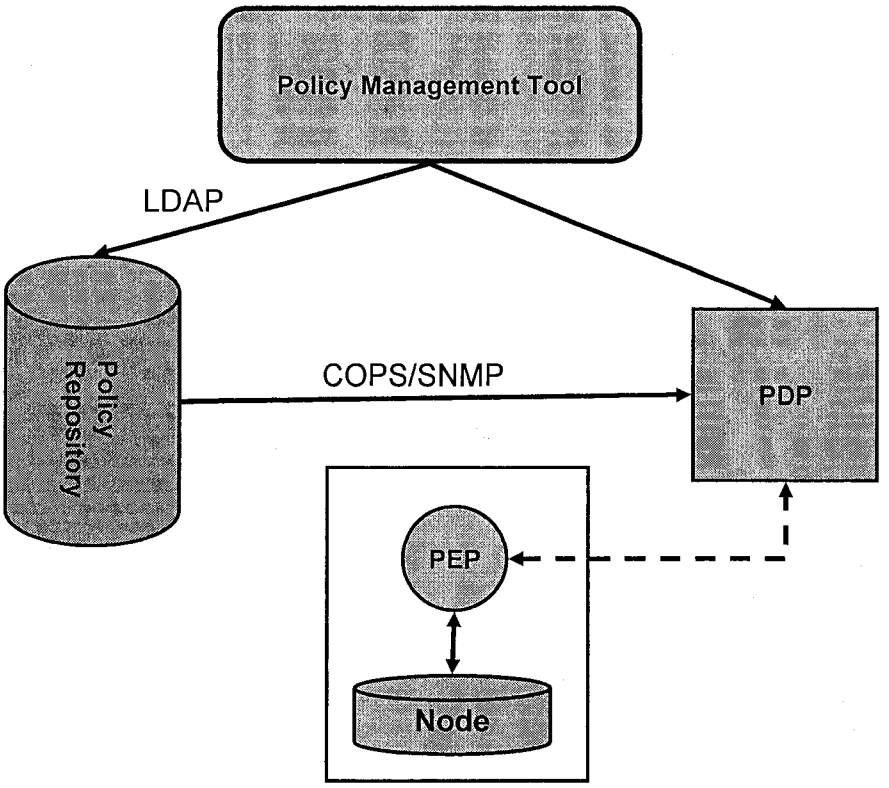

Fig. 3.1 IETF/DMTF Policy-Based Management Architecture

3.2.2 Overlay Management Using Active Networks Technology

Active networks are frameworks where network elements, primarily routers and switches, are programmable. In active networks, programs are injected into the network, and executed by the network elements to achieve higher flexibility and to present new capabilities. Each active node runs a Node Operating System (NodeOS) and one or more Execution Environments (EEs). The NodeOS is responsible for allocating and scheduling the node's resources (link bandwidth, CPU cycles, and storage), while each EE implements a virtual machine that interprets active packets arriving at the node. Each EE defines a distinct virtual machine or "programming interface" on which Active Applications (AAs) can be built to provide a particular end-to-end service [77].

There are two major approaches to service deployment in programmable and active networks [78]: In-band and out-of-band deployment schemes. In the former, the code of the services is transmitted together with the data in active packets, called capsules, which the service can execute with the co-located data on the appropriate nodes along the data path in the network. This scheme is suitable for on-demand deployment of small and simple services. In the the out-of-band scheme on the other hand, the service code is separated from the actual data, and is processed during the deployment phase. This scheme is more applicable to high-level, application-oriented services; the FAIN [79] project belongs to the second scheme. The deployment architecture [80] defines how and when service components are invoked and installed on selected network nodes, such that that the service deployment requirements are fulfilled, and the runtime management architecture deals with the installation of service components in execution environments, and with the management of component instances. The architecture uses the component as the main abstraction. From the management viewpoint a component instance represents two aspects: The functional aspect in which the component is seen as a (part of a) service instance; and the non-functional aspect in which the component is seen as a resource.

Recently, active networks technology has been geared toward aiding network and service management functionalities. In [81], a new layer—the application environment (AE) layer—has been added to the active network framework, to offer high-level services desired by applications. These applications are called user-defined processing modules (UPMs). They are greatly simplified because they leverage services offered by the AE layer.

Although active networks-based management seems to provide some promising solutions, introducing more programmability into network devices also implies adding more complexity to their management functionalities. In addition, excessive utilization of active packets results in network performance deterioration due to their high utilization of network resources—one solution to this problem is to restrict the functionality of the programs carried by the active packets, thus resulting in

architectures with decreased capabilities. Furthermore, the dispatched active packets or programmable codes introduce new safety and security concerns.

There also exists a serious resource discovery problem. Active routers will not be deployed everywhere at the same time. Rather, they will be deployed individually or in isolated pockets. Given the unavoidable extra overhead in applying intelligent processing to packets, active routers will be deployed at the network periphery rather than in the network backbone. How are applications to find these isolated resources and put them together for a single purpose?

3.2.3 Automated Management for Overlay Networks

As illustrated earlier, current approaches in the literature present simple adaptation algorithms which offer sub-optimal solutions to the management problem. Dynamic self-adaptation in response to changing QoS needs; resources availability; service cost; perceived performance of the network components or even neighboring networks, will become an essential operation in future networks. In the following, we investigate some of the few trials for automating one or more of the overlay network management functionalities.

The CADENUS (Creation and Deployment of End-User Services in Premium IP Networks) project [82] attempts to automate network service delivery. The focus was on how QoS technologies can be controlled and managed via standard interfaces in order to create, customize, and support communication services for demanding applications. Mediation components are used to represent the main actors involved, namely users, service providers, and network providers, and define their automated interactions. By defining roles, responsibilities, and interfaces, the service deployment process is decomposed into a set of sub-processes whose mutual interactions are standardized. The model brings novel contributions to automated management. Nevertheless, it lacks scalability and does not discuss impacts of network heterogeneity on system performance.

The DHARMA (Dynamic Hierarchical Addressing, Routing and naming Architecture) [83] proposes a middleware that puts no constraint on the topologies of the overlays, and defines a distributed addressing mechanism to properly route data packets inside the overlay. It separates the naming and addressing of overlay nodes, and so can be used to enable network applications to work over the Internet in an End to End mode while exhibiting mobility, multicasting, and security in a seamless way. The routing is greedy and follows the closest hierarchy to the destination node. The middle ware achieves reasonable results for network dynamics <=10% and restricts overlays to End to End communications.

The ADCCS (Autonomous Decentralized Community Communication System) [84], [85] provides a framework for large-scale information systems, such as content delivery systems. It forms a community of individual members having the same interests and demands at specified time. It allows the members to mutually cooperate and share information without loading up any single node excessively, and organizes the community network into multi-levels of sub-communities. ADCCS's is concerned with reducing both the communication delay of a message that is broadcasted to all community nodes (while considering latency among them), and the required time for membership management.

In [86], a distributed binning scheme is proposed to improve routing performance by ensuring that the application-level connectivity is harmonious with the underlying IP-level network topology. In the binning scheme, nodes partition themselves into bins such that those nodes that fall within a given bin are relatively close to one another in terms of network latency. To achieve this, a set of well known landmark machines are used and spread across the Internet. An overlay node measures its distance, i.e. round-trip time, to this set of well known landmarks, and independently selects a particular bin based on these measurements. The scheme is targeted at applications where exact topological information is not needed, such as overlay construction and server selection; however it provides no support for the application specific demands.

In [87], [88], and [89], a social-based overlay for peer-to-peer networks is proposed. The social-based overlay clusters peers who have similar preferences for multimedia content. A similarity between two peers exists if both share common interests in specific types of multimedia content, hence peers sharing similar interests can be connected by shorter paths so that they can exchange multimedia content efficiently. Specifically, whenever a peer requests an object of interest, it can locate the object among its neighboring peers, i.e., the peers that have high similarity and which are more likely to hold the requested object. Some of these approaches [87] model a distance measure that quantifies the similarity between peers, and uses random walk technique to sample the population and discover similar peers from the randomly selected samples. In [88], the similarity of peers is measured by comparing their preference lists, which record the number of the most recently downloaded objects. However, a new user who has only made a few downloads cannot get an accurate similarity measure. In [89], a central server collects the description vectors of all users, and establishes overlay links based on the distance between each pair of users. The central server does not explicitly define the description vector however, which has a significant effect on the accuracy of the similarity measure.

3.2.4 Autonomic Management

Autonomic Computing (AC), launched by IBM in 2001 [90], is an emerging technology that aims to allow users to traverse transparently and dynamically between different providers and service domains. IBM identified the complexity of current computing systems as a major barrier to its growth [90], and as a result, automated selection of service configuration, relocation, and monitoring must be carried out with minor intervention of users and system administrators. AC simplifies and automates many system management tasks traditionally carried out manually. Systems that manage themselves are able to adapt to changes in their environment in accordance with business objectives; the result is a great savings in management costs and IT professionals' time, thus freeing the latter to focus on improving their offered service

rather than managing them manually. Some of the main scientific and engineering challenges that collectively make up the grand challenge of autonomic computing were outlined in [91]. Also, a set of characteristics required by AC were identified and explained in [92].

According to the IBM vision [93], an AC system is one that knows itself and its environment, configures and reconfigures itself under varying and unpredictable conditions, heals itself, provides self-protection, and keeps its complexity hidden. Although the IBM vision is a holistic approach to designing computer systems, much of the research in this field focuses on a few specific aspects of this vision.

Autonomic communications was proposed in [93]. It has a similar concept to IBM's autonomic computing, differing in that it focuses on the individual elements of the network, how their behavior is learned and altered, and how they interact with their peer elements. A generic architecture for autonomic service delivery was proposed in [94]. It defines a resource management model based on virtualization, but it is service-independent, and is unlikely to achieve the specific QoS requirements for each service dynamically without human intervention. A model for dynamic fault tolerance technique selection for grid work flow, which allows the system to configure its fault tolerance mechanism, was developed in [95].

Pattern classification and clustering techniques that support online decision making and incremental learning in autonomic systems were proposed in [96]. The use of policies to configure autonomic elements to enforce the required behavior in an Apache web server was presented in [97]. A set of UML-based models were developed and used in [98] to specify autonomic properties and to deploy policies as an executing system based on composition and model modification. A policy-driven model based on multi-agent systems was also proposed in [99]; in that model, Web services are represented as agents, and agent behavior is controlled using high level policies. A mapping of biological systems to PBMS was introduced in [100]; this system is hierarchical and relies on mechanisms for organism regulation, which supports self-management at different levels of the hierarchy. Humans in an organization thus specify policy at a

level of abstraction that reflects their specific needs. The difference between our work and all these approaches is that the above approaches consider a particular service to which their design is appropriate. In addition, policy generation is not a fully automatic process and human intervention is still needed.

Projects such as Service Clouds [101], Autonomia [102], GridKit [103], Auto-Mate [104], and Unity [105] utilize the autonomic concept in different ways. Service Clouds provides an infrastructure for composing autonomic communication services. It combines adaptive middleware functionality with an overlay network to support dynamic service reconfiguration. Autonomia provides dynamically programmable control and management to support the development and deployment of smart applications; primarily, it achieves the self-healing property for failed entities. GridKit proposes a middleware that offers a consistent programming model across different communication types. AutoMate enables the development of autonomic Grid applications by investigating programming models, frameworks, and middleware services that support autonomic elements. Finally, Unity designs both the behavior of individual autonomic elements and the relationships that are formed among them, in order to create computing systems that manage themselves. A detailed survey on autonomic computing is available in [106]. Although, in theory, AC seems to provide the ultimate solution for the complex management problem, in general, research efforts towards Autonomic Management are still in their infancy and are still faced with many challenges.

Our work focuses on service-specific overlay networks; thus, the interaction between the network and computing entities is based on a service request/offer concept in which each entity is responsible for its internal state and resources. An entity may offer a service to other entities. The offering entity responds to a request based on its willingness to provide a service in its current state. Our work is concerned with all possible phases of the service delivery in SSONs—from the instance of requesting a service to terminating it. As a result, we present an integral approach to SPs that wish to deliver services over their infrastructure.

3.3 Resource Discovery

In large, distributed networks, media content usually requires adaptation before it is consumed by clients; for example, video frames must be dropped to meet QoS constraints. A client with a PDA requires a scaled-down version of the video; a mobile user requires the content to be cached for viewing. Therefore, we need to discover the required resources before we construct the media flow path. Resource discovery techniques can be classified into centralized, distributed, and semantic approaches. This section provides an overview of the most established resource discovery techniques.

3.3.1 Centralized Approaches

In centralized approaches, all resource information (resource description, node address, etc...) are kept in a centralized server. Each arriving node needs to actively notify this server about its kept resource information; consequently, nodes only need to consult node address from the server about its needed resources. This type of architecture is very simple and easy to deploy, but has the problem of single point-of-failure. *Napster* [107], a peer-to-peer system, adopts this approach. Alternatively, directory servers in which all the services offered in the network are registered can be used. Either nodes know how to direct queries to all these servers, or the servers know how to communicate with each other. For example, the centralized approach [108] is suitable for networks with stable topology and for applications that do not require frequent service updates. Though it consumes bandwidth, has a high message overhead, and suffers from single point-of –failure (in the servers), this approach has been used in the Internet [109] for web services and other applications [110]. Needless to say that such a centralized approach is not well suited to the dynamic topology of SMART, where services on offer must be updated frequently.

3.3.2 Distributed Approaches

Flooding is the simplest approach to resource discovery. A query is broadcast to all nodes. A requesting node contacts its neighbors, which in turn contact their own neighbors until the resource is found. Each receiving node determines independently how to process and respond to the query. Although this approach is flexible and requires no topology awareness, it consumes bandwidth and suffers from an exponential number of overhead messages [111], [112], [113], and [114]. In [115], a path-directed approach that is explicitly targeted to media stream processing services is proposed: The query is sent to nodes that move it progressively closer to the destination. While this is more efficient than flooding, queries are still sent to nodes where answers may not be available.

Dynamic Hash Table (DHT) approaches are decentralized, and are proposed mainly for P2P systems. They can be classified, based on their inter-connection architectures, into flat or hierarchical. Flat approaches Chord [54], CAN [116], Pastry [117], Tapestry [118], and Kademlia [119] provide a uniform distribution of peers and resources. They support scalable and distributed storage and retrieval of $(Key, Data)$ pairs on the overlay network, and they do this by associating each node in the network with a portion of the key space; all data items whose keys fall into a node's key space are stored at that node. DHT systems differ in the details of the routing strategy as well as in the organization of the key space. In a network of N nodes, where each node maintains $O(\log N)$ routing entries, DHTs generally perform lookups using only $O(\log N)$ overlay hops (CAN [116] is an exception).

Chord is a decentralized P2P lookup service that stores $(Key, Data)$ pairs for distributed data items. It assigns keys to its peers using consistent hashing [120], where consistent hash functions assign peers and data keys an m-bit identifier using SHA-1 [121] as the base hash function. Given a key k, the node responsible for storing k's data can be determined using a hash function that assigns an identifier to each node and to each key (by hashing the node's IP address and the key). Key k is assigned to the first peer whose identifier is the successor of k in the identifier space. Chord nodes form a

connected Ring topology, with each node maintaining a finger (routing) table with *O(logN)* pointers to other nodes. When a new chord node joins the network, certain keys have to be moved to the new joining node from its successor. Similarly, when a node leaves Chord, all its keys are assigned to its successor. This operation costs $O(log^2N)$ messages. Chord supports only a lookup operation; given a key k, it maps k into the node responsible for storing the data associated with k. In the steady state, Chord performs lookups in *O(logN)* messages to other nodes.

CAN is designed to be scalable, fault-tolerant, and self-organizing for internet scale applications. It is built on a virtual d-dimensional cartesian coordinate space on a d-torus (for some fixed integer d). Every node in CAN owns a distinct zone from the virtual overall space. A CAN node maintains a routing table that holds the IP address and virtual coordinate zone of each of its neighbors in the coordinate space. Using a uniform hash function, any key k is mapped onto a point p in the coordinate space. K and its *data* are then stored at the node that owns the zone that contains p. Routing messages follows a greedy forwarding pattern; when a node joins, it will randomly select a point of d-dimensional space, and then becomes responsible for half of the zone that this point belongs to, and hold all keys whose IDs belong to this zone. A CAN node maintains a coordinate routing table that holds the IP address and virtual coordinate zone of each of its immediate neighbors in the coordinate space. A node sends the message to a neighbor node that is closest to the destination coordinate. The routing table size at each CAN node is $2 \times d$, and lookups cost $O(d \times N^{1/d})$ messages. Thus, in contrast to Chord, the routing table maintained by a CAN node does not depend on the network size N, but the lookup cost increases faster than *O(logN)*. If $d = logN$, CAN lookups match Chord's.

Pastry nodes form a decentralized, self-organizing, and fault-tolerant overlay network within the Internet. Each node in the Pastry system is assigned a nodeID, a 128-bit node identifier that is used to indicate the position of the node in circular nodeID space in the range $[0 - (2^{128}-1)]$. When a new node joins the system, it is assigned a randomly generated nodeID from the uniformly distributed nodeID space.

Routing in Pastry is a prefix-based routing. A node forwards the message to another node whose nodeID shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the present node's ID. For a network of N nodes, Pastry routes to the numerically closest node to a given key in less than $log_2^b N$ steps under normal operation (where b is a configuration parameter with typical value of $b = 4$).

Each Pastry node maintains a *routing table*, a *neighborhood set,* and a *leaf set.* The neighborhood set is not normally used in routing messages; it is useful in maintaining locality properties. It contains a set of nodeIDs and IP addresses that are closest (according a proximity metric) to the local node. The leaf set is used during the message routing, and contains a set of nodes with half of those nodes being the numerically closest larger nodeIDs, and the second half being the numerically closest smaller nodeIDs, relative to the present node's nodeID. Each node maintains a routing table of $[log_2^b N \times (2^b - 1)]$ entries. Each entry in the routing table contains the IP address of one of potentially many nodes whose nodeID have the appropriate prefix. Therefore, lookups cost between any pair of nodes is $(log_2^b N)$.

Tapestry shares similar properties with Pastry, but the main difference between them lies in the handling of network locality and object replication; Tapestry' is based on Plaxton [122]. The core location and routing mechanisms of Tapestry are similar to those of Plaxton, but Tapestry's goal is to improve the capability to detect, circumvent, and recover from failures through maintaining periodically updated cached content. To avoid a single point of failure, Tapestry uses multiple roots for each data object— Routing is longest prefix routing. Tapestry uses local tables at each node, called *neighbor maps*, to route overlay messages to the destination ID, digit by digit. Each node in Tapastry maintains routing maps, which are organized into routing levels, each level containing entries that point to a set of peers closest in distance that match the suffix for that level. The routing method guarantees that any existing unique peer in the system can be located within at most $\log_B N$ logical hops, in a system with N peers using nodeIDs of base B. Since the peer's local routing map assumes that the preceding digits

all match the current peer's suffix, the peer needs only to keep a small constant size (B) entry at each route level, yielding a routing map of size $B \times \log_B N$.

Kademlia is similar to many peer-to-peer systems. Keys are opaque, and each peer is assigned a NodeID in the 160-bit key space, with <key,data> pairs stored on peers with IDs close to the key. Kademlia uses a novel XOR metric for distance between points in the key space. XOR is symmetric and allows nodes to receive lookup queries from precisely the same distribution of nodes contained in their routing tables. Each node in the network stores a list of {IP address,UDPport,NodeID} triples for nodes of distances between 2^i and 2^{i+1} from itself. These lists are called k-buckets. Each k-bucket is kept sorted by last time seen. Therefore, the maximum state kept by any node is k, a typical value for k being 20.

The Kademlia routing protocol consists of the following steps: 1) PING probes a node to check if it is active; 2) STORE instructs a node to store a <key,data> pair; 3) FIND_NODE takes a 160-bit ID and returns {IP address,UDP port,NodeID} triples for the *k node* it knows that are closest to the target ID; 4) FIND_VALUE is similar to FIND_NODE: It returns {IP address,UDP port,NodeID} triples, except in the case when a node receives a STORE for the key, in which case it just returns the stored value.

Despite their efficiency, current Dynamic Hashing Tables (DHTs) are limited to pure lookup of unique Keys, which introduces a problem: A user will not always be aware of a Key`s value. Large routing tables incur costs, in addition to the traffic maintenance needed to keeping them up to date (in order to avoid stale entries that may cause timeout delays). DHT systems also exhibit dramatic latency growth when subjected to increasing churn, where nodes continuously join and leave the network. This may lead to network partitions, causing subsequent lookups to provide inconsistent results [123], [124].

Multi-attribute searches have been proposed to solve the limitations of unique IDs. The main approaches include Reverse Hash Tables [125], [126], [127], [128], [129], [130], and [131], and Keyword-fusion [132], [133], and [134]. Reverse Hash Tables are

based on inverted indexes, in which (Resource ID, Node) is replaced by the inverted list: (keyword, List of resources/List of nodes). Each resource is described by a list of keywords. Then, each keyword is indexed separately. Therefore, the inverted index is distributed among peers by keyword; hence a query with n keywords can be answered by n nodes. All the results are collected by the requesting node that computes the intersection of all the responses as the final result. The scalability limitations of this technique and its existing optimizations, in terms of high bandwidth consumption, have been demonstrated in [135]. In Keyword-fusion, the resource identifiers are obtained by hashing an attribute or a list of attributes using a consistent function. This list of attributes defines a single key that identifies the resource uniquely. It solves the problem of common keywords—those keywords that frequently appear in the keyword lists of a large number of files—by generating a new keyword (referred to as *synthetic)* through concatenating a set of keywords in the Alphabetic order. The value part of the mapping for the synthetic keyword is an intersection of all file lists in the original mapping, i.e. a list of the files that contain the set of keywords in their keyword lists. While Reverse Hash Tables introduces a significant load in the network, and Keyword-fusion reduces this traffic, they both require the keywords to be known beforehand.

Hierarchical DHTs can be further classified into vertical and horizontal approaches. Vertical approaches [136], [137], and [138] ensure that the nodes in any domain form a DHT routing structure by themselves. The DHT containing all nodes in an internal domain is obtained by merging all the DHT "children" into a larger DHT, and then by applying this recursively at higher domains. This has many advantages; for one, local traffic does not affect other layers. Other advantages lie in network proximity and efficient caching. However, the creation of several DHTs assigned to sub-domains can affect the scalability and the total number of connections in the network. Furthermore, a routing table is needed for each DHT, thus increasing the maintenance cost. In horizontal approaches [139], [140], leaf overlay networks are connected using a single DHT that contains the conceptual hierarchy, and which optimizes the routing in the whole network, thus reducing the number of connections that build the hierarchical infrastructure at the expense of more complex routing tables. In addition to the

traditional DHT benefits, hierarchical DHTs provide fault isolation, effective caching, and bandwidth utilization. The limitations of traditional DHTs do, however, still exist.

Smart Media Routing and Transport (SMART) is a highly dynamic environment. MediaPorts (MPs) resources change frequently, and frequent re-hashing is not feasible. Consider the example of a service that requires MPs caching at least 100MB in the network; using DHT to find all caching MPs results in a huge message overhead—finding those with less than 100MB cache is not useful. Using a multi-attribute search, we can retrieve only those MPs with an available cache greater than 100MP. However, after selecting a specific MP, its cache size will be less than before. This implies that we need to re-hash this MP and its cache. Different caching MPs belong to different domains. Hierarchical DHTs will therefore not be able to group them into one cluster without the clustering becoming costly. More importantly, if multiple MPs are needed for a specific media flow, they cannot be discovered all at once because their number and types are not known beforehand. In a dynamic distributed environment, discovering the first MP and trying to discover the rest recursively is costly. This is because the discovery time will be substantial and, before reaching a solution, the network might have changed, which in turn might require beginning the search again.

One common way to improve the performance of a network is to increase its connectivity and decrease its diameter, a feat that can be achieved by adding links. However, we want to add as few links as possible since their cost has practical implications on the design. Additionally, the number of links going out of a node must be small to allow for fast maintenance. Also, the links must be added in a homogeneous way so that nodes can be easily inserted and messages can be routed systematically.

3.3.3 Semantic Approaches

Semantic approaches [141], [142] have been proposed as an enhanced search mechanism. Peers with similar content become members of the same Semantic Overlay Network (SON). Queries are then forwarded to the SON that satisfies the query, thereby reducing their communication cost. A major problem of SONs is to construct efficient

overlays. In [141], SONs are presented as groups of peers, which share common interests. In [142], a similarity-based, pre-computed binary relation among peers is encoded in SON. Each peer becomes directly connected to a small number of other peers that are likely to be good routing targets. Bloom filters [143] or hash sketches maintained in a directory based on DHT have been used as a brief summary technique for query filtering and routing. In [144], a probabilistic algorithm based on bloom filters is first used to discover content. If it failed, a deterministic algorithm is used. This is motivated by the assumption that the probabilistic algorithm finds resources quickly when it can, and fails quickly when it cannot. In [145], a DHT maintains a global key-to-document index. The key-index only contains single terms and term sets that are rare and thus discriminative with respect to a document collection. A particular instantiation of the key-indexing creates keys by combining terms appearing in well-defined contexts. Their work assumes that peers are cooperative and provide documents for indexing that will become searchable through a global index. At the same time, they offer computing and storage resources to build and maintain the global index and the underlying P2P network.

A different notion of SONs [146] is related to schema mappings and peers that are logically interconnected through schema mappings; the approach is a two-layer model: A physical layer based on the P-Grid access structure, and a logical semantic overlay layer. Peers in Grid-Vine create (and possibly index) translation links, mapping one schema onto another. These links can then be used to propagate queries in such a way that relevant data items annotated according to different schemas can also be retrieved. Query forwarding can be implemented using iterative forwarding, where peers process series of translation links repeatedly, or recursive forwarding, where peers delegate the forwarding to other peers.

Other SON examples are globally available term statistics about the peers' contents [147], gossiping strategies [148], locality in the underlying network [149], and resource shortcuts that group peers into clusters according to their contents [150], [151].

However, many of those methods involve directory lookups, statistical computations, and multi-hop messages.

Semantic approaches have been also proposed for ad hoc networks [152], [153], and [154], and Grid technologies [155], [123], [156], [157], and [158]. In ad hoc networks, nodes are considered equals, in effect acting like a special kind of P2P network. Grid systems allow the sharing of heterogeneous, distributed resources that are potentially numerous and dynamic. Resource discovery is achieved by either using broadcasting, or advertising services to the entire network or through special structures.

Routing queries on top of a semantic overlay will result in more efficient resource discovery. But problems remain: These approaches are not fault-resilient, the overlay is difficult to maintain, and the message overhead is considerable. Moreover, assuming the lack of knowledge of both global content and network topology, the actual construction of these overlays is challenging. In a Peer-to-Peer architecture, each node is initially aware only of its neighbors and their content. Thus, finding other peers with similar contents to form a SON becomes a tedious problem.

Resilient Overlay Network (RON) [48] allows distributed Internet applications to detect and recover from path outages and periods of degraded performance. However, RON overlay does not scale for more than 50 nodes.

Geographical routing [159], [160] is a routing method that uses geometrical reasoning for forwarding packets. Typically, a greedy approach is used: This means that a packet is forwarded to the node in the neighborhood that is closest in Euclidean distance to the destination. Since MPs may not fall exactly in the path between MediaClient (MC) and MediaServer (MS), geographical routing does not guarantee that the needed MPs will be discovered. Furthermore, since these MPs are not known beforehand, geographical routing becomes impractical.

3.4 Summary

In this chapter, we have extensively discussed various approaches that have been proposed to address the issue of managing overlay networks. The development of management schemes, such as policy-based management, active-network management, and autonomic computing have made it possible to provide some management operations. Nevertheless, it is generally difficult to manage service-specific overlay networks while maintaining the service specific requirements, since numerous overlays exist, each dictating its own requirements. However, it is vital to construct, configure, and manage these overlays to prevent them from consuming network resources, and to make them efficient. By investigating current research contributions in literature, the following key conclusions have been reached:

1. Static network components configuration is inefficient to manage overlays in the aggregate levels; overlays have their own logical components that should be configured and managed.
2. The costs of maintenance of existing management models are high, due to the reliance on human operators.
3. Current management mechanisms mostly cover only a single part of the global overlay life cycle management problem. An adequate management mechanism should cover all the phases that overlays pass through during their lifetime. Furthermore, it should incorporate management actions between different classes and types of overlays.
4. Segregation between resource discovery and overlay management leads to an inefficient usage for both. Resource discovery mechanisms should be efficient and accurate, in addition to providing a high success rate. Moreover, a resource discovery mechanism should be integrated in the management scheme such that the frequent requests do not generate great overhead on the network resources.
5. The deployment and management of overlays is a serious issue, and in order to support large-scale, distributed applications, overlays must be deployed and

managed in an automated manner without any manual intervention, or unnecessary communication. Ideally, no modifications to applications or operating systems should result from this process.

Chapter 4

Autonomous Management Infrastructure

As mentioned earlier, this dissertation is focused on developing novel approaches that can be used to achieve an autonomous management of SSONs. To this end, the following chapters will develop schemes to automate SSONs management. For that purpose, this chapter gives an overview of the proposed framework for autonomous SSONs management, it proceeds as follows: Section 4.1 introduces overlay management challenges. Section 4.2 reviews SMART modeling for overlay networks and its limitations. Section 4.3 presents an overview of the proposed architecture, and section 4.4 discusses the proposed architecture components in details. In Section 4.5, we present a use case scenario that shows the steps used in creating, adapting, and terminating SSONs. In Section 4.6, we present simulation details and results. Finally, the chapter is concluded with a brief summary.

4.1 Introduction

As discussed earlier, an overlay network is a virtual network of nodes and logical links that is built on top of an existing network in order to implement a service that is otherwise, not originally available. Overlays can be used to increase routing robustness and security, to reduce duplicate messages, and to provide new services for mobile users. They can also be incrementally deployed on end hosts without the involvement of ISPs, and they do not need new equipment or modifications to existing software or protocols [48], [49], and [50]. These attractive benefits come at the cost of increased overhead and complexity. Overhead is increased because of the additional packet header and the redundant work at the overlay and IP layers. The constantly increasing traffic

carried by the overlays also tends to overload the network and consume its resources [3]. In addition, overlays are usually designed independently. This increases the chances that they will negatively affect each other; bottlenecks are created, and they reduce the performance both of the overlays and of the underlying network. Overlays therefore need to incorporate a management mechanism that reduces this complexity and keeps them operating correctly.

Overlay management is challenging for several reasons. First, the dynamic changes in network conditions and topology quickly renders management information obsolete. For example, network nodes may fail, links may get congested and routing information may change. In addition, any changes in the routing path are affected by the required QoS [64], bandwidth, latency, and the existence of other overlays. Second, overlay members are dynamic since new users may join or leave. Finally, each overlay node has limited knowledge of the network, and the knowledge varies among overlay members. With a large number of overlays, management by traditional methods becomes harder to achieve and, a new management scheme must be supplied. This new scheme must account for the different phases that overlays go through during their lifetime: creation, optimization, adaptation, and termination.

Creation requires the setup of a routing table in each overlay node along the end-to-end path, a path that must optimize the QoS metrics. Adaptation produces a new behavior that reflects a change in the overlay environment, and may be necessary to assist mobility, deal with the failure of an overlay node, or control congestion. Termination means claiming the reserved resources and updating routing tables.

The use of policies offers an appropriately flexible and customizable management solution that allows network entities to be configured on the fly [4], [5]. Usually, administrators define a set of rules to control the behavior of network entities. These rules are translated into component-specific policies that are stored in a policy repository, to be retrieved and enforced as needed. However, existing management systems usually direct the management task to physical entities such as routers, switches, and gateways. In our architecture, the task is assigned to the overlays and their

logical elements. Policies are generated dynamically, and no human interaction is required.

We propose a new approach to the autonomous, context-aware, policy-based management of overlay networks. The approach's novelty lies in that sets of policies, specifically adapted to the current availability of resources and users' demands, are dynamically generated from the available context information and enforced on the fly. Policies also control the various construction phases harmoniously. Our goal is to automate overlay management in a dynamic manner that preserves the flexibility and benefits that overlays provide.

4.2 SMART Modeling for Overlay Networks

To recap, in SMART, A Service Specific Overlay Network (SSON) is constructed for each media delivery service or group of services. An SSON is a virtual network composed of a set of overlay nodes and links, which customizes the network to the particular requirements of the service (such as QoS, media formats, responsiveness, cost). SSONs have the ability to transparently include network side entities called MediaPorts (MP) in the communication path, thereby providing the flexibility to modify the content and services such as caching, adaptation and synchronization [47].

Overlay nodes are physical Ambient Network nodes that have the capabilities needed for them to become part of the SSONs; these are a control plan and a user plan. The control plan is responsible for the creation, routing, adaptation, and termination of SSONs. The user plan contains the Overlay Support Layer that receives packets from the network, sends them to the network, and forwards them on the overlay. Overlay nodes implement a sink (MediaClient, or MC), a source (MediaServer, or MS) or a MediaPort (MP) in any combination. MPs are special side components that provide valuable functions to media sessions such as special routing capabilities, smart caching and adaptation. MPs, MCs, and MSs are managed by the control plan. The control plan also

contains a MediaPort Directory Service (MPDS) to maintain information about the available MPs, such as location, load and cost.

SMART's architecture is described in detail in [2]. But SMART does not specify the means by which SSONs are constructed and managed, nor does it specify how SSONs can be adapted dynamically according to the users' context; our architecture addresses these drawbacks. First, the control plan is equipped with a new entity called the Overlay Policy Enforcement Point (OPEP). The OPEP is designed to control node resources and functionalities by enforcing configuration changes based on context information. This in turn is used to dynamically generate policies. Second, an SSON Overlay Policy Decision Point (OPDP) is used for each SSON or group of SSONs to make the appropriate decisions about the creation, adaptation and termination of SSONs. In addition, a set of System PDPs (SPDPs) is used to coordinate the actions of OPDPs. COPS protocol [74] is used to exchange policy objects between the OPEP, OPDP, and SPDP.

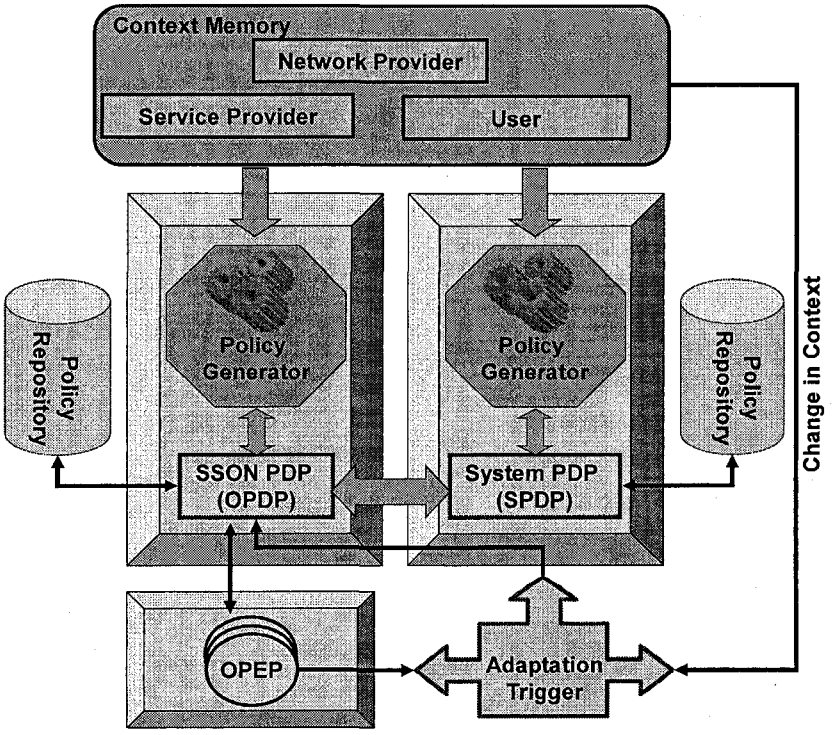

Fig. 4.1 Context-aware overlay policy architecture

4.3 Architecture Overview

A schematic description of the main components of our proposed architecture is shown in Fig. 4.1. The central components are the OPDP, SPDP, OPEP and the Policy Generator (PG). The OPEP is a component in the overlay nodes while the OPDP and the SPDP are remote entities that may reside at a policy server. The PG generates and adapts policies using the available context information. The OPEP is the point at which policy decisions are actually enforced. Policy decisions are made primarily at the PDP1. The PDP receives policies from the PG, evaluates them and distributes them appropriately. The OPEP requests decisions and enforces them. With any change in the context information, an adaptation process is triggered by first generating the policies that reflect the new context and then by proactively sending them to be dynamically enforced.

We distinguish between the sources of context information, such as user context, service provider context and network provider context. All these types of context must be considered when building a comprehensive management system. We assume that the context information has been gathered in a context memory [161], [162], which feeds it to the PG. The PG generates different types of policies: user policies, application policies, service provider policies, network provider policies, and service-specific policies. Any change in the SSON environment triggers an adaptation process in which new policies are generated dynamically and sent to the appropriate PDP.

The policy repository saves all the policies generated for each SSON, and also contains other information relevant to the management task. This may include the SSONs constructed so far and the Media Port Directory Service (MPDS) that lists the available MPs and their capabilities, user registration, and accounting information.

There are two different types of PDP: The SSON PDP (OPDP) and the System PDP (SPDP). Since the number of overlay nodes expected in each SSON is small, each OPDP is assigned one or more SSONs. The OPDP is responsible for automating the task of creating, adapting, configuring, and terminating its designated SSONs. It

[1] We use PDP to refer to both OPDP and SPDP unless it is necessary to make a distinction.

communicates directly with the participating overlay nodes to achieve its tasks. Typically, its tasks are the following. 1) It makes configuration decisions in response to the system policies received, and uses these decisions to configure the overlay nodes participating in a given SSON. 2) During construction of an SSON, it is responsible for optimizing the service path to meet the required QoS metrics of the high-level system policies as well as the context of the service. 3) It monitors the QoS metrics for the multimedia session and continuously adapts the service path to the changing conditions of the network, the service, and user preferences. 4) It also monitors the participating overlay nodes, and finds alternatives in case any of the nodes do not conform to the required performance level. OPDPs receive goal policies from SPDPs in order to decide the types of actions required.

A single OPDP is able to automate the management functions only for the SSONs that it manages. If a network contains a large number of SSONs, it may be that they are not really isolated. On the one hand, each overlay node can be part of many SSONs if it offers more than one service or if it has enough resources to serve more than one session. On the other hand, the SSONs' service paths may overlap, resulting in two or more SSONs sharing the same physical or logical link. For example, if two SSONs share the same routing MP with the same goal to maximize throughput, the result will be race conditions on the shared resources. Therefore, in order to achieve a system-wide balance, the OPDPs need to coordinate their actions. This is achieved using SPDPs.

SPDPs interact with one or more OPDPs. They pass the high-level system policies, such as for load-balancing, to the OPDPs. Whenever they find shared goals between two different SSONs, they send information that avoids conflicting actions. The OPDPs then contact each other and create a Virtual Management Overlay (VMO) as illustrated in Fig. 4.2. This VMO coordinates their actions before they are passed to their overlay nodes.

Sharing goals is not the only reason to create VMOs. SSONs that share common links and SSONs that belong to the same policy domain (same service class, ISP, etc.) may also create VMOs among themselves. Additionally, SSONs that share common

nodes or links affect each other's performance as they compete for the shared resources. This can result in degraded performance as the competition causes them to frequently evaluate their decisions in an attempt to reach the desired performance goals. All SSONs in a given domain (ISP) are also expected to achieve the domain-wide policies together. VMOs allow these policies to be sent and adapted to each SSON in a way that achieves the desired goals. VMOs also allow the sharing of control and information between different SSONs. A set of SSONs co-located in a given vicinity (such as an area, domain, AS) usually has independent routing decisions based on its observations for its environment. Sharing this information results in reduced overhead for each overlay and allows policies to be adapted and generated in order to achieve better performance.

When VMOs are created, each OPDP can obtain information of two types, the first related to the coordination actions, the second to the common metrics in which each OPDP is interested. Goal policies are passed from SPDPs to the OPDPs they manage. The context information of the network, users, and services is used primarily to aid in generating suitable policies at each level.

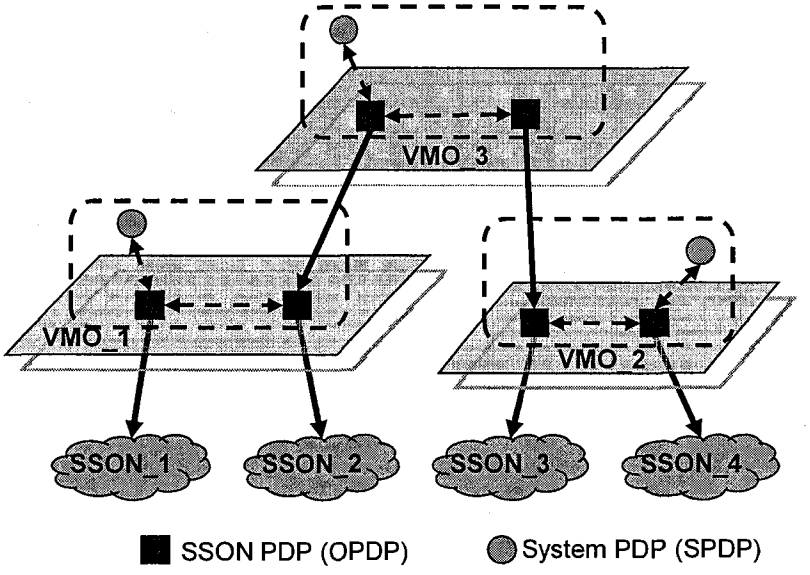

Figure 4.2. Virtual Management Overlay (VMO) hierarchy

4.4 Proposed Architecture Details

This section describes, in detail, the central components of the proposed architecture, and the various steps in the construction, adaptation, and termination of SSONs.

4.4.1 Policy Generator (PG)

The central component in the PG [163] is the automated policy adaptor (APA). The key feature in APA design is to separate the mapping of abstract higher-level goals to network-level objectives from the functionality that adapts the behavior of network components. Although we used the same concepts as [163] to generate policies, our PG goes beyond those concepts as follows. First, policies for system and business objectives are derived from the relevant context information, as are policies for users and applications. While in [163], network administrators and users/applications specify these policies using a graphical user interface and register them with the APA, we derive these policies from the context, thereby further automating the process. Second, instead of sending the policies directly to the managed resources, we send them to the PDPs. This separates the process of generating policies from the process of making decisions. This is done because adapting an existing policy may not be sufficient to adapt an entire SSON. In addition, adapting one policy may require adaptations in other policies in order to achieve an SSON-wide adaptation. The separation of policy generation from decision making allows for more flexibility to adapt SSONs dynamically based on their specific requirements as units.

As a result, overlay management is seen as a process of learning from current system behavior by creating new policies at runtime in response to changing requirements. The PG generates and adapts five kinds of policies and sends them to PDPs. These are user policies, network policies, application policies, service provider policies, and service specific policies. The adaptation process is either triggered at pre-set intervals or by events received from the OPDP and network monitors in the OPEP.

These events are in response to user-related or application-related events such as low battery level, or changes in a user's location or an application's QoS requirements.

The PG considers policy adaptation to be a process of learning from current system behavior. This learning process assembles new policies at runtime. The policy-making passes through three main phases: Stage setting, consideration of alternative decisions, and reassessment of the applied decision. As a final step, a feedback mechanism ensures that the new policies are correct.

In the first phase, all necessary information is obtained from the context information. In the second phase, the PG selects one or more actions from the action space that best attains the specified change. The selection is made by calculating an expected loss value for each action. Choosing the optimum policy is simply a matter of choosing the action that minimizes the expected loss values. The third phase involves the assembly of a new policy as a result of the actions selected in the previous step. The newly assembled policy consists of a triggering event translated by the PG from higher-level policies such as user location. Conditions are specified by the characteristics of the satisfied objective and the selected action. The new policy can also be associated with a lifetime, a duration after which it expires and is deleted. Once a policy is assembled, it is sent to the appropriate PDP.

The final step is performed by the reassessment module that evaluates the success of the new policy. Network monitors in OPEPs measure the average values for the actual QoS parameters. For example, an SSON's actual throughput of traffic is calculated and compared with the objective. The difference between the values measured by the monitors and the required objectives is fed back to the first stage as a new objective change. If the difference is substantial, the adaptation process is repeated.

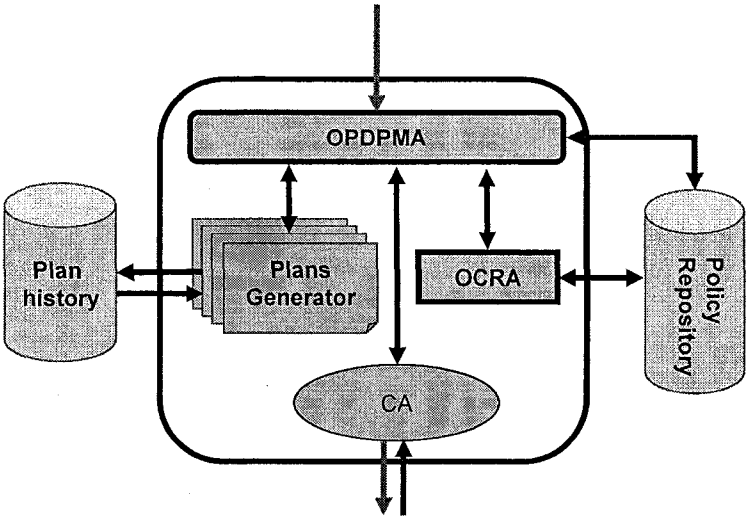

Fig. 4.3. OPDP architecture

4.4.2 Overlay Policy Decision Point (PDP)

As shown in Fig. 4.3, the Overlay Policy Decision Point is composed of a Management Agent (OPDPMA), an Overlay Conflict Resolution Agent (OCRA), a Plans Generator, and a Communication Agent (CA).

The management agent receives the policies from the PG, analyses them, and makes appropriate decisions. It assigns a unique ID to each SSON and to each flow. Flows can then be routed independently when, for example, it is necessary to meet the required QoS. To create or adapt an SSON, the management agent sends the IDs, the SSON's performance requirements and the requested MP capabilities to the Plans Generator.

The Plans Generator is responsible for constructing the topology of the SSON that meets the requested performance properties (such as low overlay path latency or a specific overlay path bandwidth) and the requested MP capabilities (such as a caching MP with at least 300Mb disk space). It searches for the path that best meets the QoS constraints. For this to be done, a set of QoS metrics has to be available. These metrics (such as link costs, delay, jitter, and bandwidth) are either part of the context information available in the plan history or are obtainable by reusing measurement techniques similar to those presented in [164], [165]. These costs are updated using a link state

update protocol that is outside the scope of this paper. To facilitate the process of finding the optimal path, and to allow the network to interactively contribute to successful media delivery, the Plans Generator includes the suitable MPs whenever necessary.

The MP location is chosen to be as close as possible to the shortest path between the source and the sink. The Plans Generator either chooses the most suitable MP, or any suitable MP and then searches for the optimal path from the source to the MP, and from the MP to the sink. The former choice ensures that the MP is as close as possible to the shortest path, yet it does not guarantee that an MP will be found. To avoid this problem, we expand the search parameters and run the search again, accepting that this increases the time needed to find an optimal path and consumes more resources. The latter choice allows a parallel search for the optimal path, and reduces the management overhead, but does not guarantee that the MP closest to the shortest path will be found. Any MP can be chosen at random from the Media Port Directory Service, in which SMART assumes that MPs register their locations and capabilities. But since geographically closer nodes are expected to have fewer hops between them [166], [167], it is more efficient to choose a MP that is geographically close to the sink. This is the approach taken in this paper.

Once the Plans Generator finds the optimal path, it constructs a connection matrix that represents the SSON topology and sends it to the management agent, and to the PG. The PG generates the policies that construct or adapt the SSON and sends them to the Management Agent.

The OCRA (is listed here for completeness and it is an object of future work) receives the policies from the management agent, and checks them for any conflict with previous policies. This ensures that new or adapted SSONs do not negatively affect the operation of those already deployed. If a conflict is found, the policies are rejected and the SSON has to enter an adaptation process. Conflicts are generally divided into two types: Static and Dynamic. In our model, a static conflict is one that is detected at the time a new policy is generated; a dynamic conflict is one that occurs at runtime. If no conflicts are found, the management agent either sends the policies to the appropriate

OPEP through the Communication Agent (CA) for immediate enforcement or retains them in the policy repository to be activated at a future time. The CA is responsible for sending policy objects to the appropriate OPEP as well as for receiving policy objects from OPEPs.

The SPDP's main tasks are to coordinate the actions of two or more OPDPs, and to distribute system-level policies that guarantee system-wide performance. These policies are derived from the network and service provider context information. SPDPs consist of the same components as the OPDP, except that they do not contain the Plans Generator module. They therefore receive the system policies such as load balancing from the Policy Generator, analyze them, and send them to a conflict resolution module. This module checks the consistency of the new policies against those already installed. In case of conflicts, the new policies are fed back to the Policy Generator for re-adaptation. If no conflicts are found, the policies are sent to their OPDPs through the communication agent.

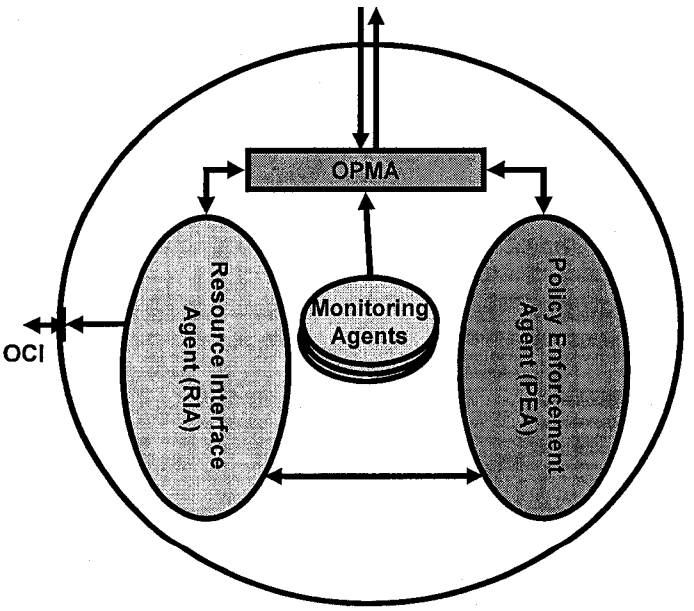

Fig. 4.4 OPEP Architecture

4.4.3 Overlay Policy Enforcement Point (OPEP)

The OPEP is the point at which policies are actually enforced. As shown in Fig. 4.4, the OPEP is composed of four cooperating agents. It receives notifications or messages from the monitoring agents that require a policy decision, and then constructs a request for a policy decision and sends it to the OPDP. Once the policy decision is received, the OPEP enforces the decision by accepting or denying as appropriate.

- Overlay Policy Management Agent (OPMA). The OPMA manages the various aspects of the OPEP through two-way transmission of policy objects with the OPDP. It also receives notifications and adaptation events from the Monitoring Agents. Once a new policy object is received, the OPMA analyzes it to determine the appropriate action. If it is a decision, it is sent to the policy enforcement agent (PEA), which decides how the policy will be enforced. If it is an adaptation, it constructs an appropriate policy object, and communicates with the OPDP to acquire a decision.

- Policy Enforcement Agent (PEA). The PEA is responsible for enforcing or removing the overlay policies at the overlay node. It receives and analyzes policy decisions from the OPMA. With the assistance of the Resource Interface Agent (RIA), it enforces them at the appropriate overlay node component. It also sends a report to the OPMA describing the success or failure of the enforcement.

- Resource Interface Agent (RIA). The RIA is an interface between the OPEP and the components of the overlay node. As such, it communicates with the appropriate component to enforce a policy. For example, it communicates with an Overlay Service Layer (OSL) component to update a routing table entry and with the MP to reserve or free its resources.

- Monitoring Agents (MA). MAs are placed at various layers of the system as required. Each MA is responsible for monitoring its layer, and reporting to the main monitoring agent in the OPEP. MAs are therefore able to monitor the available resources and capabilities, as well as the connectivity of the overlay node to neighboring nodes. MAs also monitor the performance of overlay nodes and MPs. Whenever reduced

performance is detected, MAs send an adaptation event to the Management Agent so that additional resources can be freed.

4.5 Use Case Scenario

This section provides a simple illustration of how our architecture uses policies to create, adapt, and terminate an SSON.

SSON Construction: The process of constructing an SSON starts with the service provider (or user) defining the properties of the service to be offered. These properties include the required QoS, the required network side functions (such as caching), and any other requirements specific to the service. The OPEP sends the properties to an SPDP that assigns the task to an OPDP and forwards the properties to the PG. The PG then converts them into policies and sends them to the OPDP. For example, the PG would generate policy (a) for a user requesting a video from a streaming video server:

```
If (User = "x") and (Application = "video") Then
    Max_Bandwidth < 128 kbps,
    Aggregate_Bandwidth < 1024 kbps,
    OneWay_Delay < 200ms,
    Special_Functions = "caching",
    Priority = 3
```

(a)

Once the OPDPMA decides, with the help of the policy repository, that this is a new service, it assigns a unique SSON_ID to the service. If the service has multiple flows, it assigns a distinct FLOW_ID to each of them. The video session in our example has two flows (video and audio). Each flow can be routed on a different path if necessary, but our initial assumption is that both are routed on the same path. The OPDPMA therefore constructs the following policy object and passes it to the Plans Generator.

Action	SSON_ID	FLOW_Ids		Application	User	Policies
Create	3432	Audio = 1	Video = 2	video	x	Policy (1)

The Plans Generator searches for the optimal path, including the suitable MPs. It then

```
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
   and (User = "x") Then
     Connection_MAT = {ONodeA (client), ONodeB (caching MP),
                       ONodeC (server)}
     Client = ONodeA,
     Next_HOP = ONodeB,
     Server = ONodeC
```

(b)

sends back a policy object (b) containing the proposed topology in the form of a connection matrix.

This plan is sent to the PG that uses the relevant context to generate the policies (c) that will make up the SSON. The first policy instructs the server's OPEP to mark each packet with the session information. The second policy updates its routing table to route the packets to the next overlay node in the SSON topology. The rest of the policies update the routing tables of overlay nodes. In OnodeB, a caching media port is configured to cache the data and to deliver it to ONodeA (the client). At that point, the OSL component is instructed to deliver the packets to the requesting application.

Once the OPDPMA receives these policies, it sends them to the conflict resolution agent to check for conflicts with policies and SSONs that are already installed. If there are no conflicts, the OPDPMA constructs a policy object for each participating overlay node. This policy object contains information about its neighboring overlay nodes in order to facilitate routing and the enforcement of applicable policies. If a conflict is detected, the policies are rejected and an adaptation process is triggered so that the conflict can be overcome.

Once the policy object is received by the OPEP, it is analyzed and enforced. In our example, the caching MP at ONodeB starts caching the video content and sending the video to the client from the cached version. The OPEP reports to the OPDP informing it about the success or failure of enforcing the decision. Assuming success at all overlay nodes, the SSON is now constructed.

```
Server = ONodeC
If (User = "x") and (Application = "video") Then
    SSON_ID = 3432,
    Audio_FlowID = 1,
    Video_FlowID = 2
IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1))
Then
    Next_HOP_ONodeID = ONodeB,
    Next_HOP_ONodeIP = xxx.xxx.xxx.xxx,
    Media_Ports = none

Target = ONodeB
IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1))
Then
    Media_Ports = "Caching",
    Caching = "ok"
IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1))
   and  Caching = "ok") Then
    Disk_Space = 300 MB,
    Freshness_Factor = 50.0,
    Expiration_Time = now + 10h,
    On cache miss refer to: ONodeC_IP = "xxx.xxx.xxx.xxx",
    On overload refer to: ONodeC_IP = "xxx.xxx.xxx.xxx"
IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1))
   and (User = "x") Then
    Next_HOP_ONodeID = ONodeA,
    Next_HOP_ONodeIP = xxx.xxx.xxx.xxx,
    Sending_Rate = default

Target =ONodeA
IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1))
Then
    Application = "video"
```

(c)

SSON Adaptation: If another user requests the same service, the overlay must be adapted to include the new user. In our example, the new user's OPEP constructs a policy object containing the request and user information. After authenticating the user for security and accounting purposes (operations outside the scope of this work), the OPDPMA checks if there is an SSON for the requested service, and when one is found, it triggers an adaptation process by sending a message to the PG requesting any policies specific to the user. Assuming that the new user's context information has already been fed to the context memory, the PG generates policy (d):

```
If (User = "y") and (Application = "video") Then
    Available_Bandwidth = 16 kbps,
    One_Way_Delay < 300ms,
    Special_Functions = "routing, scaling",
    Priority = 2
```

(d)

Based on this policy, the OPDPMA decides to scale down the video frames so that they may be routed to the new user. It therefore includes another MP with routing and adaptation capabilities. Along with the SSON information, this is sent to the Plans Generator that invokes the plan and decides which Media Port to include. It sends back a policy object (e) containing the proposed topology in the form of a connection matrix.

```
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
   and (User = "y") Then
    Connection_MAT = {ONodeB (caching), ONodeD (media adaptation
                      MP),  ONodeE (client)}
    Client (y) = ONodeE,
    Next_HOP = ONodeD,
    Scaling,
    Server = ONodeB,
    caching MP
```

(e)

In our proposal, the caching Media Port is used to route the data to the new user rather than the original streaming server. This adaptation saves network bandwidth and resources because the same content is distributed only once for each SSON, rather than once for each user.

The PG generates policies (f). The sending rate of the caching MP is chosen to match the user preferences, the adaptation MP is configured to scale down the video frames, and the video data are buffered at the client side before being forwarded to the requesting application. The rest of the adaptation steps are the same as those in the section on creation.

```
Target =ONodeB
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
   and (User = "y") Then
    Next_HOP_ONodeID = ONodeD,
    Next_HOP_ONodeIP = xxx.xxx.xxx.xxx ,
    Media_Ports = "caching" ,
    Sending_Rate = 10p/100mls

Target = ONodeD
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
Then
    Media_Ports = "Scaling",
    Scaling = "ok"
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
   and (Scaling ="ok") Then
    Frame_rate = 10fps,
    Frame_Size = 320x240
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
Then
    Next_HOP_ONodeID = ONodeE,
    Next_HOP_ONodeIP = xxx.xxx.xxx.xxx

Target =ONodeE
IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1))
Then
    Buffer_Size > 2 MB,
    Application = "video"
```

(f)

SSON Termination: If a user leaves a session normally, (or even unexpectedly because of a crash), the SSON must be adapted accordingly. If the session is ended normally, the OPDP receives a notification of leave from the user; it then sends a policy object to the overlay nodes to uninstall the existing policies. If the notification is received from the last user of the SSON, the session is terminated by sending policy objects to the overlay nodes that are part of the SSON. In our example, a leave request from user y causes the following policies (g) to be sent by the OPDP to ONodeB, ONodeD, and OnodeE. Once these policies are received by the respective OPEPs, they are deleted immediately and all ongoing packets are dropped.

To deal with the failures that may occur in a dynamic network, we adopt a fault recovery mechanism similar to the one described in [168], which guards against failures with a checkpoint technique. Each overlay node that is part of an SSON sends the checkpoint back to the OPDP. The OPDP caches any checkpoints obtained. If an overlay node fails, the OPDP can receive no more checkpoints, upon which, it decides whether to re-adapt the SSON or terminate the SSON if it has no users.

```
Target = ONodeB
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
    and (User = "y") Then
    Delete_Policy

Target = ONodeD
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
Then
    Media_Ports = "None", Scaling = "No",
    Delete_Policy
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1)
Then
    Delete_Policy

Target =ONodeE
IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1))
Then
    Delete_Policy
```

(g)

4.6 Simulation Details and Results

This section summarizes simulation results of the proposed scheme. In our simulation, the topology was constructed using the BRITE [169] topology generator, and the network was simulated using the J-Sim network simulator [170], a simulator with a Java(tm)-based engine. We conducted two experiments to test our architecture; the first simulated a moderate-sized network to test a mobility scenario, the second a large network to test the response to heavy demand.

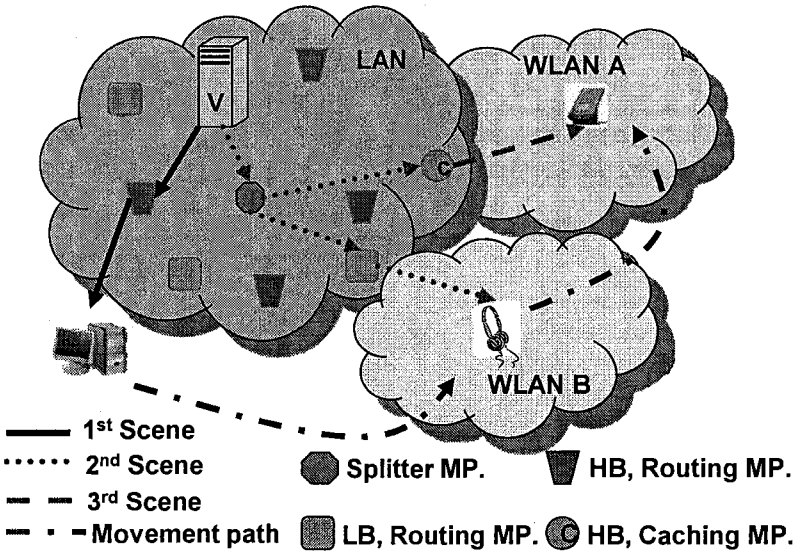

Fig. 4.5 Mobility Scenario

4.6.1 Experiment 1: Mobility Scenario

In the first experiment, three interconnected networks were simulated as shown in Fig. 4.5. One was a LAN with randomly generated topology, and the other two were WLANs with different bandwidth capacities. WLAN A had a higher bandwidth capacity (15 Mb/s) than WLAN B (5 Mb/s). The scenario consisted of three scenes, the first showing the creation of the overlay, and the second and third showing the dynamic adaptation and routing. In the first scene, a user tunes her office PC into a video server and starts to

view a video. A SSON that requires a high bandwidth MP is created. In the second scene, the user moves to a cafeteria during her lunch break. When she enters the cafeteria's coverage area, the network detects her PDA and its wireless headset. The SSON adapts by choosing a splitting MP that splits the audio and video of the session into different flows. The audio flow is sent through a routing MP to the headset. Assuming that the user previously established her context by stating that she has a meeting in a conference room after lunch, a caching MP close to the conference room's wireless LAN is selected and automatically configured to catch the video flow. In the third scene, the user moves to the conference room and the SSON is automatically reconfigured. The video flow from the caching MP is resumed, thereby reducing transmission delays. The throughput for each scene appears in Fig. 4.6.

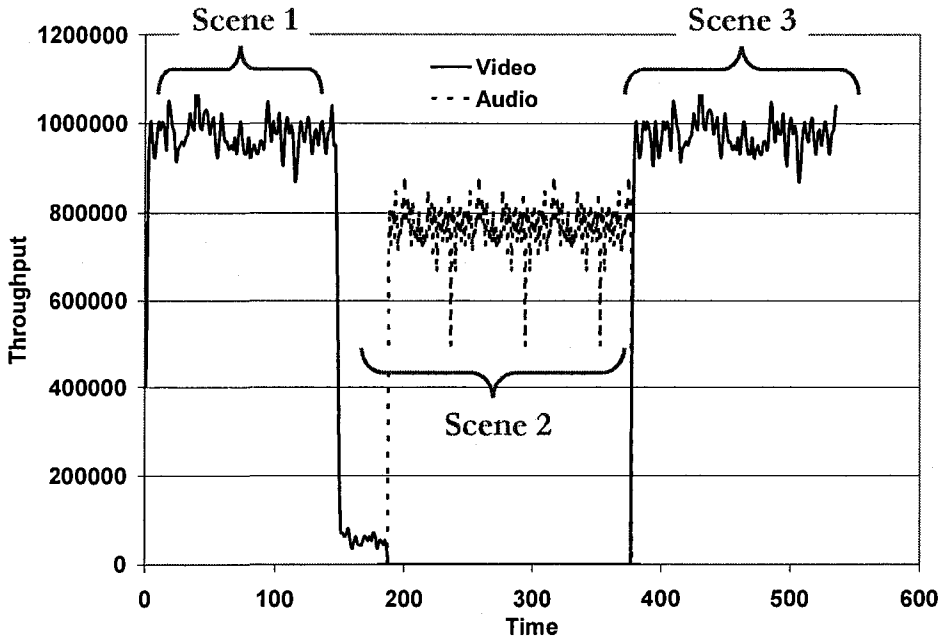

Fig. 4.6 Mobility Scenario Result

We observe that as the user moves to the cafeteria, the throughput decreases until it reaches its minimum. As the throughput starts decreasing, the OPEP sends an adaptation request to its SSON OPDP. The OPDP then decides, based on user and network context, to split the session into video and audio flows. It then it adapts the SSON to route the

audio flow to the new location. This is shown as Scene 2 in Fig. 4.6 where the audio throughput is less than the original video and audio throughput. When the user moves to the conference room, the monitoring agents in the user OPEP detect the move and report to the SSON OPDP. The OPDP then readapts the SSON to the new context and resumes the transmission of the session from the cached version. Our architecture dynamically handles the adaptations of SSONs to the available context information since all adaptations are done transparently and with no explicit interaction from the user.

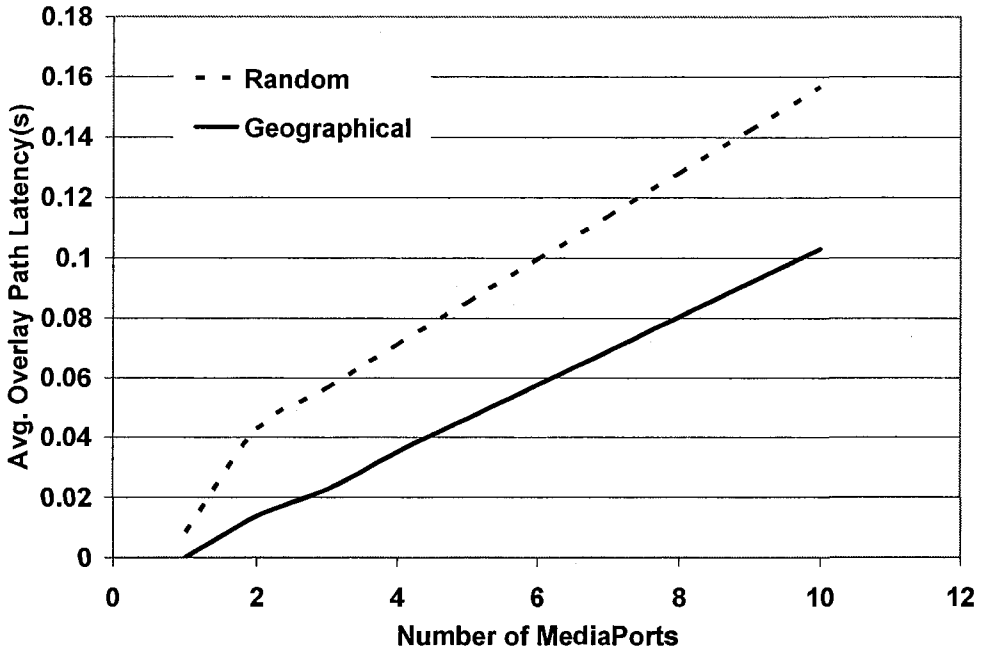

Fig. 4.7 Average Overlay Path Latency

4.6.2 Experiment 2: Large Scale Network

The topology used in the second experiment has 1000 nodes. The bandwidth assigned to each node is randomly selected between 128 and 512 Kbits/s, and the links propagation delay is fixed at 1 ms. Each source generates packets according to a Poisson process with a bitrates of 3400 kbit/s, and a uniform random selection of destination nodes. Following a flash crowd characteristic, all nodes request their sessions at a random point

during the first 2 seconds while the simulation lasts for another 1000 seconds. We ran the simulation 10 times and collected the results after each run. The first run simulated 100 SSONs with each subsequent run adding 100 SSONs. To reach steady state behavior, each SSON issued one adaptation request randomly 30 seconds after the start of the simulation. As previously described, each SSON has one or more MPs when created and different MPs when adapted. In our selection of MPs, we compared two approaches. In the Geographical approach, we selected MPs that were geographically close to the shortest path between the source and the destination. In the Random approach, MPs were selected at random. We measured the overlay latency, packet stretch and management overhead.

1) *Overlay Latency*: Figure 4.7 illustrates the average latency incurred by 1000 overlays with varying numbers of MPs. MPs join the overlays until the desired number is reached; the measurements are taken after the system stabilizes. The Random approach has the worst performance, especially for large sessions. Three factors contribute to the latency overhead. First, the encapsulation and decapsulation time depends on overlay node capabilities such as CPU speed and memory. Second, as overlay packets add more information to the header of the normal IP packets, the packet size increases, thereby increasing the time needed for delivery. The solution is to equip devices, especially small ones such as PDAs, with faster CPUs and more memory in order to maintain, or if possible to reduce, latency level. Third, the processing time needed at MPs, for example to record the data into caches. The average overlay path latency increases linearly with the number of MPs in the path. Although each MP adds its own latency depending on how fast it can provide its service, the average latency incurred by each MP is 0.01s. Therefore, in order to compensate for the delay in multimedia transmission at the source and its presentation at the destination, we need to set a buffer size at the destination relative to the number of MPs used in the SSON.

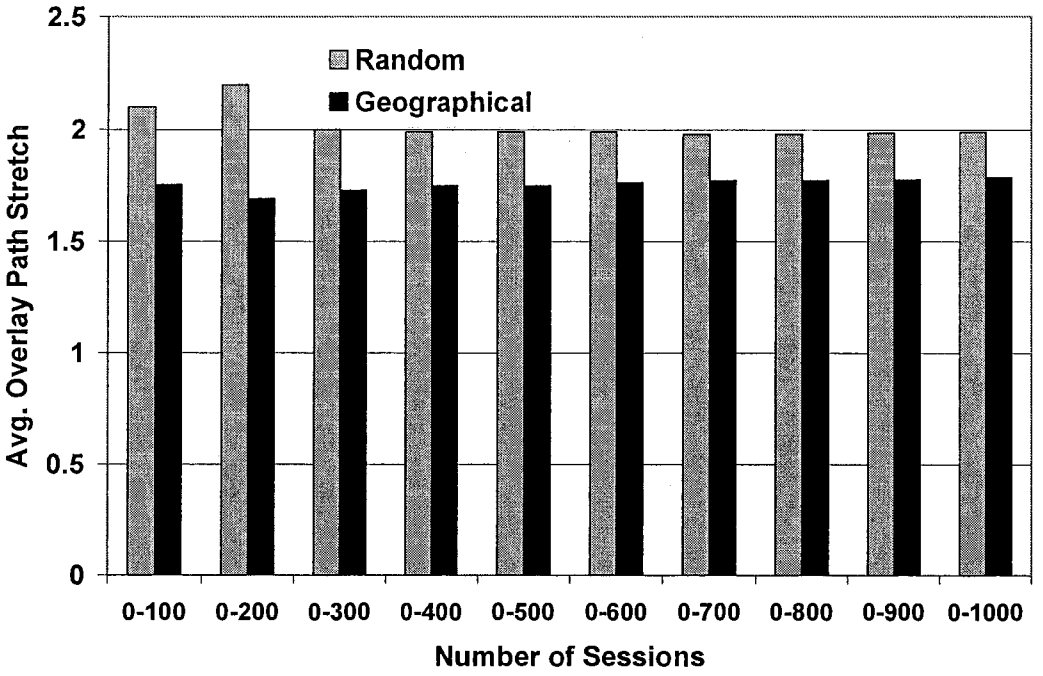

Fig. 4.8 The Average Overlay Path Stretch

2) *Packet stretch*: The stretch of the SSONs' topologies is defined as the number of physical hops taken by an overlay packet divided by the number of hops a packet takes when using an IP-layer path between the same source and destination. A high stretch value indicates an inefficient SSON topology as their packets have longer routes and delays. Fig. 4.8 shows the simulation results for the average overlay path stretch and Fig. 4.9 shows the distribution of the actual stretch values for the geographical approach. The results show that the stretch for the geographical approach ranges from 1.73–1.79. This low stretch value is not significant considering the gains obtained by using policies. The distribution of the actual stretch values shows that 3.6% of sessions have a stretch greater than 3 and 76.8% have a stretch less than 2. Although the geographical approach improves the overlay path stretch, the improvement is not significant compared to the random approach. This is due mainly to the limitation that results from using an MPDS. The MPDS is a centralized entity that, in addition to its disadvantage of being a single point of failure, frequently registers MPs services and capabilities. In a dynamic

network, this is not sufficient as the services change over time. The capacities of MPs are also dynamic as they change when sessions are added and removed. In order to decrease the stretch, therefore, there is a need to design a resource location mechanism. This would integrate the search for an optimal overlay path that satisfies a certain QoS metric with the search for the MPs needed to construct the SSON. The resource discovery approach should be decentralized and should exploit the semantics of the services offered by MPs.

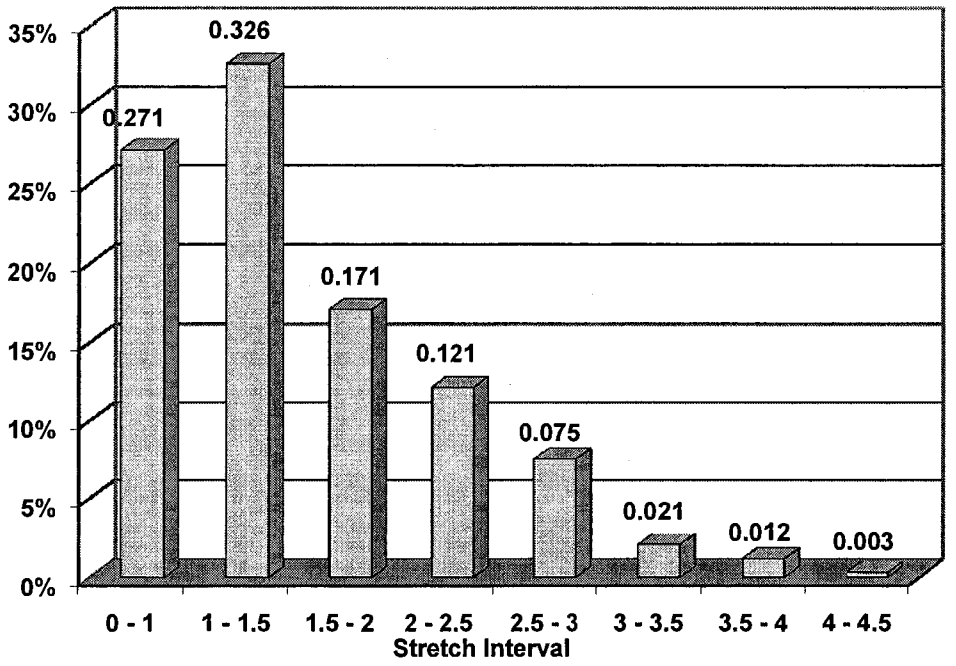

Fig. 4.9 The Distribution of the Actual Stretch

3) *Management Overhead*: The management overhead of the architecture consists of (a) the time needed to generate and enforce policies, (b) the time needed to exchange messages between the SSON OPDP and OPEPs, as well as between the SSON OPDP and the System OPDP, and (c) the time needed to access information in the policy repository and plan history. Fig. 4.10 shows the average management overhead and the 95% confidence interval. For the geographical approach results show that, while the number of sessions increases, the mean management overhead increases only slightly. For a small number of sessions, stretch is a significant factor as it results in larger delays and thereby increases the management time. For example, the second simulation run shows that the average management overhead is nearly 0.278s and the stretch is 1.75 (see fig. 4.8). As more sessions are added, therefore, the average stretch decreases and the time needed to generate policies and to access the repository and plan history outweighs the effect of the stretch. We also observe a similar behavior in the random approach. The exception is that in the random approach the management time increases

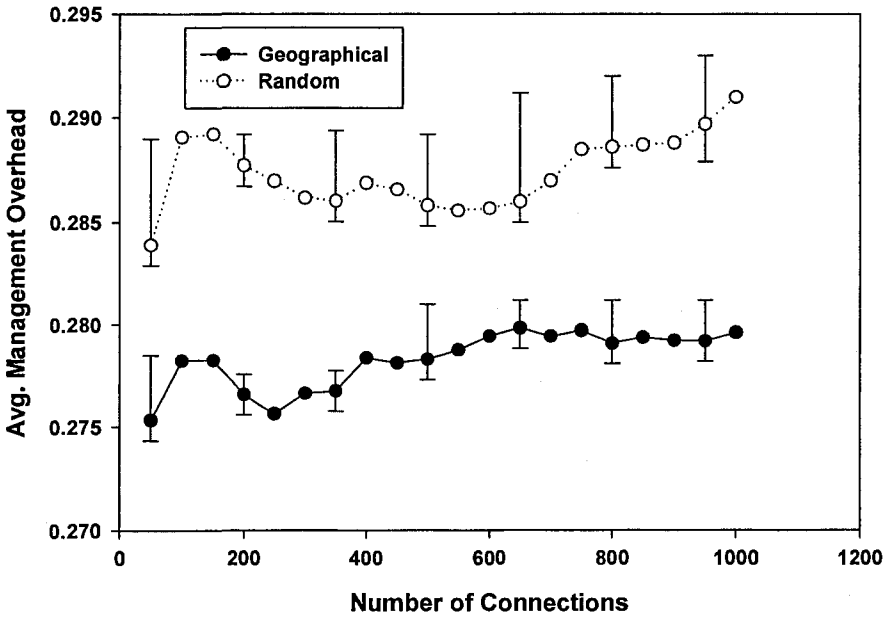

Figure 4.10. The Average Management Overhead

faster than the geographical approach where the management overhead time slightly increases. However, the average increase is insignificant compared to the gain achieved by the architecture itself.

Fig. 4.11 shows that the time needed to process and enforce policies is nearly 42% of the total time consumed in creating or adapting an SSON. Exchanging messages between the OPDP and OPEPs takes 32% of the time, and the remaining 26% is used to access information in the policy repository and plan history. This indicates that the overhead caused by introducing policies is compensated for by the gain achieved by the context-aware architecture and the dynamic deployment of SSONs. Since the extra time is needed only once to create an SSON and once for each adaptation, it is insignificant for overlays that do not require adaptation, or that have a long operation time between creation and adaptation.

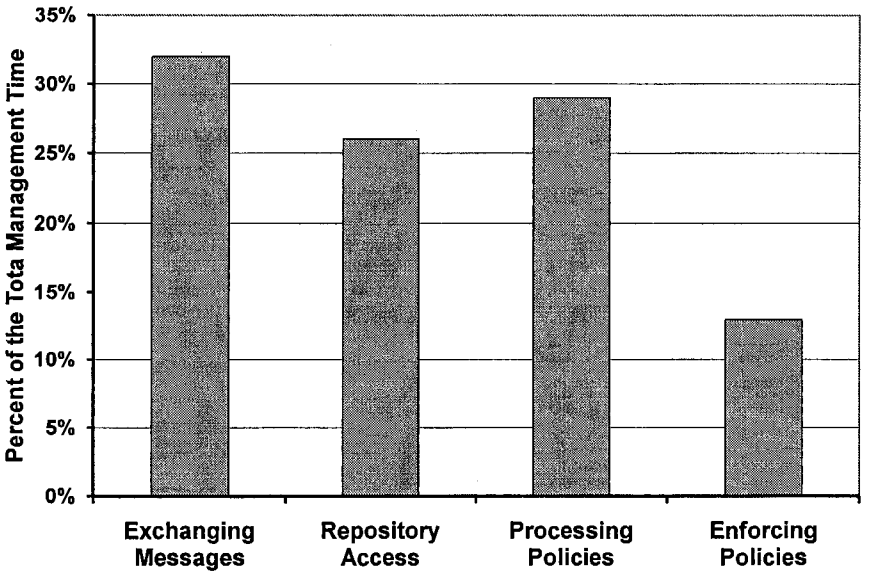

Figure 4.11. The Time Needed to Create or Adapt an SSON

4.7 Summary

In this chapter, a novel scheme for SSONs network management has been presented. The context-aware management architecture automates the task of managing overlay networks through flexible and policy-based adaptation. The creation, adaptation, and termination of overlay networks were controlled by the OPDP and the OPEP, which were used to dispatch and enforce policies. Simulation results show the flexibility and the efficiency of constructing SSONs using our scheme; it shows that on average, SSONs will be composed of 2-4 MPs. This contributes to the scalability of the scheme as low management overhead is needed by SSONs. However, MPs discovery is curried out using a MPDS. All MPs registers their services in it, and whenever a service is needed, a request is sent to the MPDS. This is clearly inefficient and centralized, and a more flexible and distributed solution is needed to cope with the dynamicity of networks, users, and applications.

Chapter 5

Semantic Overlay for MediaPorts Resource Discovery

As illustrated in the previous chapter, resource discovery represents a key component in assisting SSONs construction and adaptations. This chapter presents a novel scheme for MediaPorts resources discovery that can locate the needed MPs accurately and efficiently. The rest of this chapter is organized as follow: Section 5.1 introduces resource discovery challenges. Design goals are discussed in section 5.2. Section 5.3 introduces MPs modeling, while section 5.4 reviews the optimal chordal ring that represents the main building block for the proposed resources discovery mechanism. Section 5.5 discusses the semantic overlay construction, and section 5.6 discusses the routing of service replies on the constructed SORD. In section 5.7, we present algorithms to query, join, leave, and break SORD. Degrees of freedom in constructing SORD are discussed in section 5.8. In section 5.9, we present simulation details and results. Scalability of the resource discovery scheme is discussed in section 5.10. Finally, the chapter is concluded with a brief summary.

5.1 Introduction

Given the many sources of heterogeneity (of networks, users and applications), SSON construction uses network side functions called MediaPorts (MPs) to provide the flexibility to modify the content and services such as caching, adaptation, and synchronization [47].

In large, distributed networks, media content usually requires adaptation before it is consumed by clients. For example, video frames must be dropped to meet QoS constraints. A client with PDA, for example, requires a scaled-down version of the

video; a mobile user requires the content to be cached for viewing. We illustrate the type of applications we are targeting with the following example. Consider a user (MediaClient, or MC) trying to view a movie from a streaming video server (MediaServer, or MS) on his PDA, where the MC terminal can accept Mpeg and English subtitles, and the movie at the MS is available in DivX and French subtitles. Since the available video format is not directly usable at the MC side, an MP (or possibly more than one MP) is needed to convert the video to the needed format. SMART creates an SSON for this video flow, which consists of the MC, MS and the set of MPs that are needed to establish the service. A first step in any of these applications is for them to learn that the required services exist. In other words, they need to know "what are the services that are needed?", "where are these services located?" and "how are they found?" This is clearly a resource discovery problem. This example represents a large category of applications that pose the following challenges when designing a resource discovery system.

1. The required resources (MPs) are not known beforehand. In our example, there might be no single MP that converts DivX into Mpeg, and French subtitles into English. However, the SSON can be constructed using three MPs: The first converts DivX into RM, and the second converts RM into Mpeg. A third buffering MP is needed to remove the jitter introduced by processing the media.
2. To construct an efficient SSON with multiple MPs, the selected MPs locations should avoid looping in the overlay path. In our example, if two MPs are used to construct the SSON, the MP that provides the final acceptable video format should be closer to the MC.

There are various approaches to resource discovery. Centralized approaches maintain a mapping between the resources and the nodes offering them, but this creates bottlenecks and is not scalable in dynamic networks. De-centralized approaches, such as the popular P2P DHT approaches, improve scalability by avoiding dependency on centralized entities. But they offer limited functionality by supporting exact lookups only. They are also inefficient in that they produce a large message overhead, especially

if nodes fail. To reduce the message overhead, queries need to be routed efficiently, which leads to the proposal of semantic approaches. Generally, semantic approaches create an overlay network that connects resources based on predefined criteria. Queries are routed on the overlay only; response time is improved and message overhead is reduced. But these approaches are designed with a specific application in mind, and overlays are hard to maintain. They also broadcast service descriptions on the overlay, thereby adding to the message overhead.

We believe that these systems are not flexible enough in dynamic networks where resource properties (such as computational power, memory and available storage) vary very frequently over time. SMART's many SSONs, for example, commonly require updated resource information, both in the construction stage and during their life time in order to adapt to the ever-changing topology. Resource discovery techniques therefore need to be both resilient to the dynamic topology (i.e. made up of multiple paths between network nodes and service nodes), and efficient in terms of query responses, network communication, and accuracy. This is particularly important in SMART, where the discovery algorithm is used repeatedly to obtain updated information with which to construct and adapt SSONs. Resources should not therefore depend on other nodes to advertise or register their services. Unfortunately, existing service discovery techniques are not well suited to SMART; they are either centralized, or they produce an enormous message overhead. Nor are they resilient when failures occur.

In this chapter, we propose a novel resource discovery service for MPs, which we have named the Semantic Overlay Resource Discovery (SORD). SORD meets the challenges described above and considers not only the semantics of the services offered by MPs, but also their physical location. It provides SMART with an overlay that can be efficiently queried without using service announcements. SORD ensures that nodes without services, or those not located on the route to the desired resource, are not involved in the discovery process. Importantly, SORD is based on a widely-studied family of chordal rings called the optimal chordal ring. The result is a fault-resilient and efficient structure.

5.2 Design Goals

1) Decentralization. SORD should not depend on central entities because, in a dynamic network, these entities may not always be available.

2) Adaptability. SORD should adapt to a changing topology with low overhead. If faults occur, or as nodes leave or join, SORD should maintain its normal operations.

3) Semantic Overlay. The semantic properties of Media-Ports should minimize the overhead of routing queries.

4) Optimal Routing. SORD should avoid flooding, yet provide correct results wherever they exist.

5) Efficiency. SORD should be efficient in query response time, message overhead, and accuracy.

5.3 MediaPorts (MPs) Modeling

SSON construction involves the following main tasks:

1) Expressing in objective terms the media end points (MC, MS).

2) Discovering the MPs needed to process the media flow so that it is usable at the MC. This step requires a suitable MP service description.

3) Routing the media stream through the selected MPs.

Our work does not assume a specific MP service description. Services can be described using standard Web Service Description Language (WSDL) [171], for example, and extended with semantic metadata. For simplicity, an MP service can be described using a service identification ID, an input I, an output O, and the function f that the service provides. Using this simple representation, a service S always receives

I and produces O as a result of applying f on I. Each service used incurs a cost and each MP provides one or more services. We assume that the media end points do not alter the media flow. Therefore, they are described using their I and O only. For an MC, I refers to the possible input format and O refers to the content output channel (ex. Display). For an MS, I refers to the content input and O refers to the encoding scheme. Using this simple description scheme, an MC requesting content from an MS, can be served directly only if the input I of the client is compatible to the output O of the server. In the case of non-compatibility, an MP (or perhaps more than one) has to be inserted between the MS and the MC to establish the media delivery. Discovering these MPs is the focus of this paper. Given an input media I and a requested output media O, the problem is to find a set of services (or MPs) that transforms I into O and minimizes or maximizes a cost criterion. The result is that the MPs are chained to process the media flow.

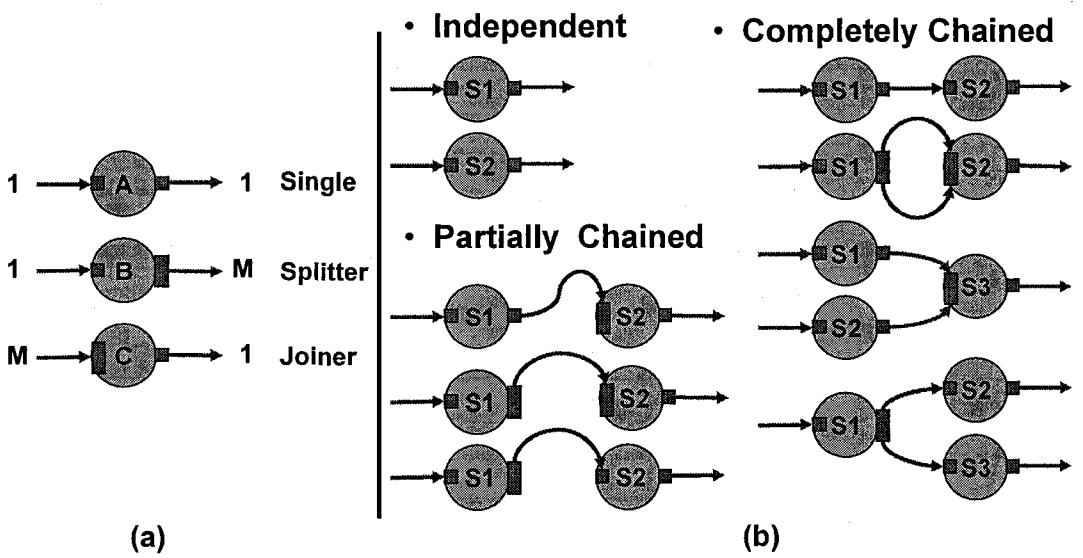

Fig. 5.1 (a) Types of MPs, and (b) MPs Chaining

To facilitate this chaining, as shown in Fig. 5.1a, MPs can be described according to their input and output ports: single, splitters, or joiners [115]. Single MPs have only one

input port and one output port. They take a media flow as input and transform it into a different output flow according to the service function that they offer. Splitters have one input and several outputs. They take one media flow as an input and produce a number of output flows. A splitter might, for example, take a video as an input and produce audio and video as an output. Joiners have several inputs that they merge into one output. Similarly, a joiner might take an audio and a video flow as input and produce a video flow as its output. MP services can therefore be independent, or partially or completely chained. As shown in Fig. 5.1b, independent MPs can perform a service without help from other MPs. Partially chained MPs consist of at least two MPs where the output and inputs of the first can be composed with some of the inputs and outputs of the second. They are partially chained in the sense that they need other MPs to provide a complete service. Completely chained MPs are made up of at least two MPs where all the output and input ports of the first are composed with all the input or outputs ports of the second. Also completely chained are MPs where all the outputs of the first are composed with some of the inputs of the second, and where the remaining inputs are composed with all the inputs of a third MP. In other words, completely chained MPs are those that provide a complete service. Since media descriptions have been well studied [172], we will not attempt to provide a complete description.

5.4 Optimal Chordal Ring

Because of their simplicity, expandability, and regularity, chordal rings have been studied for many years as an interconnection architecture for parallel and distributed systems [173]. In this chapter, we focus on a widely-studied family of degree 4 chordal rings. They are called optimal chordal rings because of their network properties of symmetry, high fault-tolerance, low broadcast time, and ease of routing. All these properties contribute productively to the efficiency of our discovery mechanism. In an optimal chordal ring, each node knows about 4 neighbors, specifically two ring nodes and two chord nodes [174]. We prefer the optimal chordal ring because the low reliability of traditional rings makes them highly vulnerable. For example, the

connectivity of a unidirectional ring of N nodes is 1 because the failure of any node i would break down the direct path from node $(i-1)$ to node $(i+1)$. Moreover, the diameter of unidirectional rings (the maximum distance between any pair of nodes) is as big as $(N-1)$. This negatively affects the performance: A large diameter would contribute to the latency between these two nodes. The extra chord connections in the optimal chordal ring are an additional overhead when compared to a traditional ring. But this overhead is offset by the low diameter, the ease of routing, and the resiliency. An optimal chordal ring can be defined as follows.

Definition 1: Optimal chordal ring of degree 4

Given two positive integers N, C, *where* $2 \le C \le N$. *The graph* $R_k(N,C)$ *is an optimal chordal ring of degree 4 whose node set is* $\{0,1,...,N-1\}$ *and*

edge set $\{\ [i,(i+1) \bmod N]\ ,[i,(i+c) \bmod N], i \in \{0,1,...,N-1\ \}\ \}$

$$If\ \begin{cases} N = 2k^2 + 2k + 1 \\ C = 2k + 1 \end{cases}$$

Where k is the diameter of the network, the ring $R_k(N,C)$ *is regular and of degree 4*

$$If\ \begin{cases} N\ \ is\ \ odd \\ N\ \ is\ \ even \& C \neq N/2 \end{cases} \qquad OR$$

Each node in an optimal chordal ring is connected to its two nearest neighbors: node i, for example, is connected to nodes $(i-1)$ and $(i+1)$, and node 0 is connected to nodes 1 and $(N-1)$. In addition, each node has two other chordal connections defined by the edges connecting the nodes at distance C in the ring to other nodes. The symmetry makes all nodes equivalent. For any $k>1$, the ring $R_k(N,C)$ has a diameter equals to k [175]. See R_2 in Fig. 5.2.

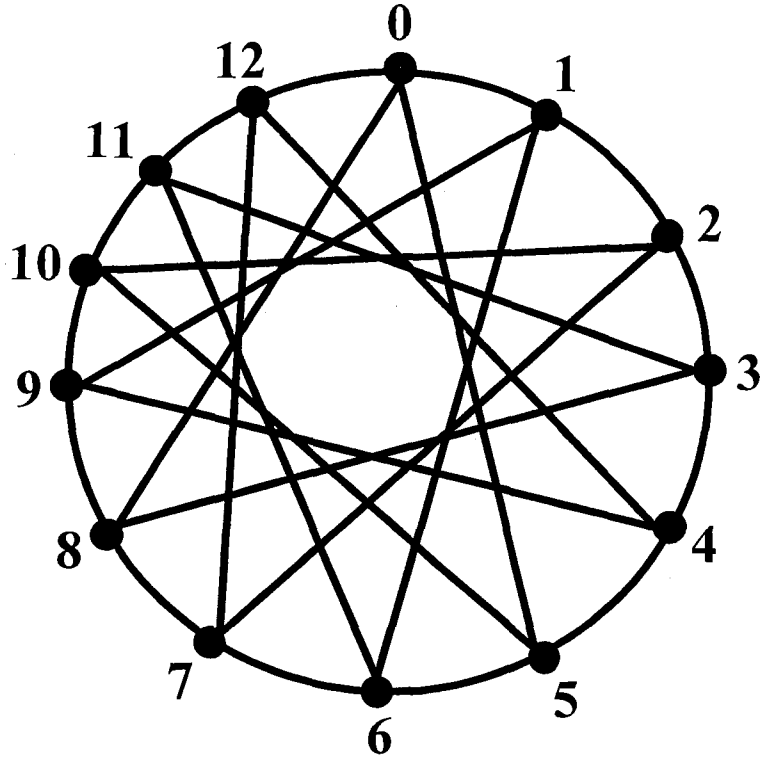

Fig. 5.2 An Optimal Chordal Ring $R_2(13,5)$

5.5 SORD Construction

MPs either provide value-added services such as caching, media adaptation, flow splitting, and synchronization, or they provide special routing capabilities. During the setup phase of a media delivery service, MPs are selected to be in the optimal location in the end-to-end path. SORD searches only those MPs most likely to have a positive answer. By constructing semantic rings for each type of service offered, efficiency is increased. Requests are routed through the semantic rings only. Studies [166] and [167] have shown that link latencies are extremely affected by geographical locations, demonstrating that geographically closer nodes are expected to have fewer hops and

latencies between them because a path on the overlay network consists of a series of application-level (not IP-level) hops between the source and destination nodes. This can lead to inefficient routing because routing on the overlay usually uses the neighboring nodes on the overlay to forward messages. These overlay neighbors might be considerably far from each other in the IP-level. To improve performance, it is necessary to avoid placing distant nodes as neighbors on the overlay. To this end, we have used nodes' Geographical locations to ensure that neighboring nodes on the overlay are also neighbors in the underlying IP-level network topology. The semantic rings are thus composed of a set of local rings connected to a global ring. The local rings group semantically similar MPs in a geographical sub-area, and one of the MPs represents the access point for the local ring. The set of access points forms the global ring, an optimal chordal ring of degree 4. SORD construction therefore consists of two steps: 1) Classifying MPs semantically, and 2) Constructing global and local rings.

5.5.1 Classifying MPs

MPs can be easily grouped into hierarchies, each of which representing a semantic overlay network. MPs can be classified, based on the services that they offer, into caching, adaptation, synchronization, and routing. Those MPs with semantically similar services in a given sub-area are connected. For example, MPs offering a caching service establish a connection among themselves. This organization improves query performance while maintaining a high degree of node autonomy. MPs can belong to more than one classification if they offer more than one service.

One important aspect of MP classification is the level of granularity. For example, caching MPs can be classified as "caching only". Or granularity can be increased by properties such as connectivity, location, cost, and capacity. Excessive granularity implies that queries will be answered with a small number of messages, but at greater maintenance cost. Poor granularity implies more message overhead and less maintenance cost; the granularity of MPs should therefore be a tradeoff between the number of messages at a given SORD and the maintenance cost at each MP. Simply

put, it should be straightforward to look at the properties of each MP and distinguish between dynamic properties that change over time (such as capacity, cost, and location) and static properties that are maintained through the MP operational time (mostly related to the type of offered services such as caching and synchronization). We believe that dynamic properties should not be included in the granularity of the overlay semantic segmentation. Including them implies that, each time they change, their logical location in the overlay should change too which is costly in a dynamic environment. Our choice, therefore, is to decrease the MPs' granularity which results in an overlay that has well-defined service semantics with fewer dynamic properties. The choice to provide a service is left entirely to the MP that has the most up-to-date knowledge about its availability, cost, capacity, and other properties.

Another important and related challenge is how to perform comparison and logical transformations on media endpoint descriptions [115]. Using the description in Section 5.3, it should be straightforward to look at MS and MC descriptions and to tell whether or not the content is in a form that can be received by the client; it should also be possible to determine the effect that a MP will have on a content description. We therefore assume the existence of a function *sim(MD1,MD2)* that computes the difference between two media descriptions. For instance, we can use a modified X-Diff [176] algorithm to analyze the similarity between two media endpoint descriptions as well as between a media description and a MP description. For example, *sim(MC,MS)* = ϕ implies that the input of the MC is compatible to the output of the MS. There is therefore no need to insert MPs in the media flow. On the other hand, if *sim(MC,MS)* = λ, then λ represents the mismatch between MC and MS description. It also represents the set of required adaptations for the media flow to be viewable at the MC. The same function can be used to compute the similarity between MPs. For example, if a media flow is passing through MP1, then *sim(MS,MP1)* = γ represents the set of required adaptations after passing the media flow through MP1. If MP1 is going to be used, γ should be less than λ. In other words, the set of required adaptations after passing the media flow through MP1 is less than the original set.

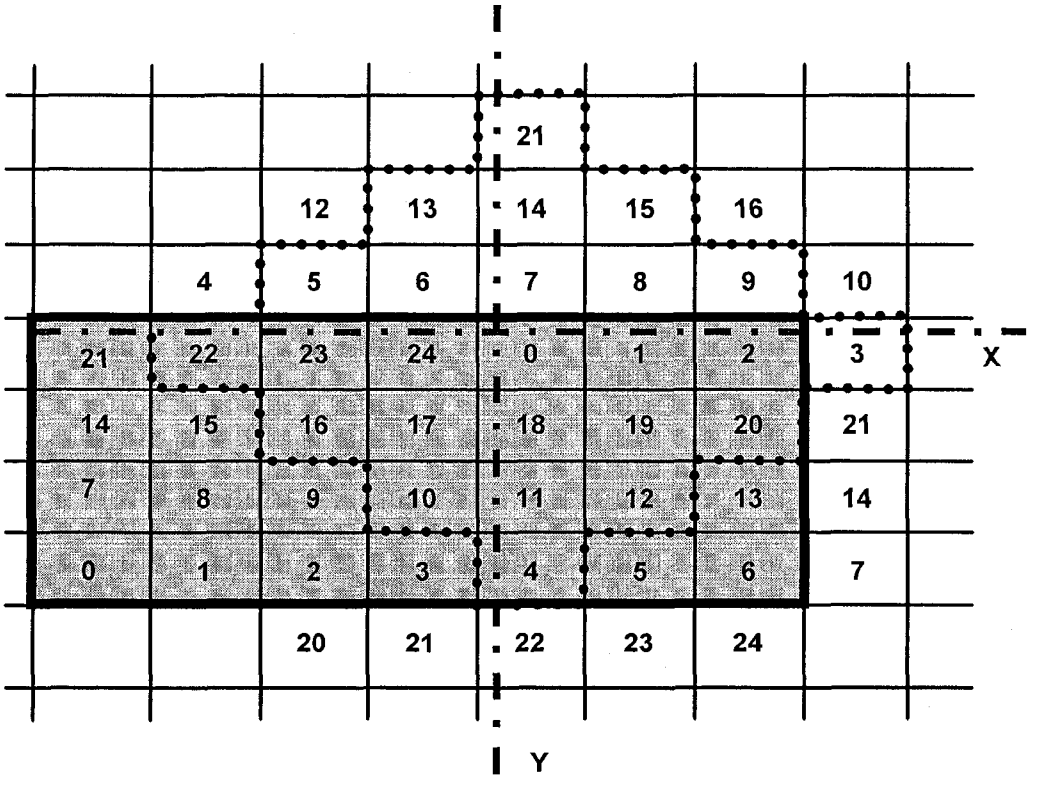

Fig. 5.3 Geometrical Representation of R_3

5.5.2 Constructing Local and Global Rings

5.5.2.1 R_k Geometrical Representation

We assume that each node knows its geographical location, and that the geographical area is two-dimensional. If we consider a geometrical representation of R_k already used in [175], we define a representation in the Euclidian plane that is divided into squares of size 1 and centered at integer coordinate. Each square is labeled with a node of R_k as follows. The square in coordinate (0,0) is labeled 0; for any square with label l, the square to the right is labeled $(l+1)$, and the square to the left is labeled $(l-1)$. The square above is labeled $l+2k+1$, and the square below is labeled $l-2k-1$ (all operations are

performed mod N). A ring edge between nodes u, v in the chordal ring is represented by a horizontal line of length 1 between the centers of the adjacent squares labeled u, v. The chordal edges are represented by a vertical line of length 1 between the centers of the adjacent squares. Fig. 5.3 shows a tile (dotted lines), a collection of contiguous squares such that every node of the optimal chordal ring appears only once as a label of a square. This tile includes all shortest paths from node 0 to all other nodes in the R_3 geometrical representation. From the tile, we see that the shortest path is $O(k)$ hops in the worst case. At node 0, we see that 4 nodes are one hop away, 8 nodes are 2 hops away, 12 nodes are 3 hops away, etc. Therefore, the average lookup cost is

$$AvgCost = \left(\sum_{i=1}^{k} 4i^2 \Big/ N - 1 \right) = \frac{2k+1}{3} \tag{5.1}$$

The average cost in (5.1) is equivalent to:

$$AvgCost = \sqrt{2 * 2^{\log N} - 1} / 3 \tag{5.2}$$

Since the optimal chordal ring is symmetric, the same tile can be used to find all the shortest paths from any node V to all other nodes by re-labeling the plane with node V in the center. The geometrical representation is complete; different tiles with different properties, or routing between ring nodes, can be represented in the same plane.

5.5.2.2 Global Ring

By dividing the geographical area into sub-areas, the sub-areas match the R_k geometrical representation. The result is that routing paths between geographical sub-areas correspond to the routing paths in the matching geometrical representation. Since the network geographical area is two-dimensional, the geographical area fits in the R_k geometrical representation. To do so, the x coordinate is divided into sub-areas equal to C (the distance of chords in R_k). Fig. 5.3 shows a tile (the solid rectangle) in the geometrical representation of R3 that is equivalent to the two-dimensional geographical area. Each square in the tile represents a geographical sub-area and, at the same time, a

node in R_k. From this tile, we observe the following: 1) All ring edges are adjacent in the tile, and they are also adjacent in the geographical plane, except on the left and right edges. For example, node 6 connects to node 7, while sub-area 6 is distant from sub-area 7. That means in SORD, global ring nodes labeled {0, C, 2C, 3C ...} are connected to nodes {(C-1), 2C-1, 3C-1 ...} respectively. We should therefore avoid using those connections whenever possible as they result in long latency. To solve this problem, we can also connect nodes labeled {C-1, 2C, 3C-1 ...} to nodes {2C-1, 3C, 4C-1 ...} respectively, 2) All chord edges are adjacent in the tile as well as in the geographical plane. The exception is chords on the top/down edges, where one chord is adjacent and the second is not. This type of connection is not considered a problem in the two-dimensional network, as the nodes are actually some distance from each other and the communication cost is paid anyway. However, these connections can be used efficiently to deliver messages to distant nodes.

Using this representation, we preserve the geographical proximity of ring nodes. As a result, a small number of hops are expected to connect global ring nodes, and less network latency is expected for the communication between them.

Each sub-area should be represented by one node. Therefore, one MP in each sub-area is identified as the access point. The selection of the access point may be based on criteria such as highest bandwidth, connectivity, or connection life time. All access points connect to each other to form an optimal chordal ring of degree 4 referred to as the global ring.

An important aspect of this matching between the geographical sub-areas and R_k nodes is that we need the same number of sub-areas during the life time of SORD. It is therefore important to carefully choose k that decides the number of nodes in R_k and to provide a solution for network evolution. k is a system parameter that depends on: 1) the actual number of MPs present in the network (n), 2) the expected growth rate (r) of MPs, 3) the expected operational period (p). Therefore, k is given by:

$$k \geq \left\lceil \sqrt{2(n+nrp)-1} - 1/2 \right\rceil \tag{5.3}$$

Since $N = (n + nrp)$, the total number of MPs is greater than the actual n, the extra $(N - n)$ are virtually hosted by other MPs.

When network size dramatically increases, it is advantageous to increase the number of sub-areas, thus decreasing granularity. To do so, we can choose an initial R_k, such that a large number of sub-areas is present to support network evolution. The extra sub-areas can be virtually hosted by existing nodes, and when enough nodes are present in these areas, they can be split from the hosted nodes. This corresponds to a larger k in (5.3).

5.5.2.3 Local Rings

In SORD, semantically similar MPs are connected to each other if they belong to the same geographical sub-area. One of these MPs in each sub-area is the access point for its sub-area. Since multiple MP types can be present in the same geo-graphical sub-area, different rings are constructed for each type. The node that is serving as the access point for a given sub-area should be aware of all different rings; it should know at least one node from each ring, and the MP types for each ring. To increase robustness, this knowledge can be replicated within the rings.

In its basic form, SORD has two levels: Global ring that connects sub-areas, and a local ring at each sub-area. To increase its scalability, we can recursively construct R_k rings in each sub-area. The maximum number of hierarchical levels is set to $(k-1)$, where k is the parameter used to construct the global ring. Therefore, the hierarchy of the local rings is of the form: $R_{k-1}, R_{k-2}, \ldots, R_2$.

The choice to build more or less hierarchical levels in a given sub-area depends on the number of MPs in it. The lowest hierarchical level is R_2 that contains 13 nodes. For a lower number of MPs, we either create a traditional ring or a star topology.

```
Algorithm 1: Routing on SORD
    Once a MP is discovered by S, S uses this algorithm to rout the query on SORD
Status: = {INITIATOR, GlOBALRING, LOCALRING, ASLEEP, WAITING,DONE}
  Sinit = {INITIATOR, IDLE} ' initial states
  Sterm = {Done} 'termination state
INITIATOR ' the source node that request the session
  Spontaneously
    { Send(Q, MCd,MSd, α) to MP ' empty chain history
     Become ASLEEP }
GlOBALRING ' any node in the optimal chordal ring that receives a query
  Receiving(Q, MCd,MSd, α)
   { if (withinα) { Send(Q, MCd,MSd, α) to LOCALRING, Become WAITING}
    else { do ROUT(Q, MCd,MSd, α), Become DONE } }
LOCALRING 'any node in a sub-area rings
  If Receiving(Q, MCd,MSd, α)
    { If (Withinα) { Process(Q), send(Q) to L.succ, become DONE } else send(Q) to L.succ }
WAITING 'optimal chordal ring node receiving Q from its local ring.
  do ROUT(Q, MCd, MSd, α)
procedure ROUT(Q, MCd, MSd, α) ' used by optimal chordal ring node to rout Q
  { if (InMSArea) Send (Q, MCd,MSd, α) to MS ' this prevents Q from being sent beyond the MS
    else {
     Candidates ← {WithinScopeAngle(G.succ,G.pred, CH1, CH2, MC, MS, α) ' choose ring neighbors
          that is within α and if a candidate is known to be failed remove it from Candidates
     Send(Q, MCd, MSd, α) to Candidates(x) – [received] } }
```

Fig. 5.4 Routing in SORD Algorithm

5.6 Routing of Service Replies

When a node sends out a query, results must be sent back to that node. Replies can be routed back to the requesting node in two ways: 1) MPs can use the shortest path algorithm to route the reply without a major use of SORD. 2) Using a reverse routing technique, the reply can retrace the query path. In the former case, the new route to the requesting node generates an additional network load. This load is reduced and system

efficiency is increased [177] if the path that already exists from the querying node is reused. In the latter case, the reverse routing is either *symmetric* or *asymmetric*. In *symmetric* routing, we can retrace the exact original routing path from the requesting node to the MP. This is possible by forcing each message to save the last address of the optimal chordal node that sent it. But failure is possible if any number of nodes in the previous path have disappeared or moved. In *asymmetric* routing, the service reply is routed on SORD, but along a path different from the original one since the optimal chordal ring owns a number of paths between any two nodes. Our strategy is to combine both symmetric and asymmetric approaches. Service replies in SORD are routed using the reverse routing (symmetric) approach. Whenever a failure is detected, the asymmetric approach is used until the reply reaches the requesting node.

5.7 Algorithms In SORD

In this section, we present the algorithms that ensure the validity of SORD, and we demonstrate how they can be efficiently used for resource discovery.

5.7.1 Querying SORD

To find the service path between source S and destination D, all the required MPs must be located. The most suitable MPs are located in the shortest path between S and D. Querying SORD therefore consists of two stages, 1) discovery, and 2) routing.

If S has no previous knowledge of the existing SORD, or if its knowledge has expired, it has to run a discovery algorithm, which is based on flooding. Since each sub-area has its own local ring of MPs, discovery takes at most a number of messages equal to the message cost of flooding in a single sub-area. In most cases, especially in sub-areas with many MPs, this cost can be expected to be very low. This is because any MP can complete the algorithm successfully. After discovering a SORD node, the information is cached locally for subsequent queries, and the actual routing for the query starts at stage two, as shown in Fig. 5.4. When a node wants to search for a service, it

sends the query to the closest MP (established in stage one). The query is then sent along the global ring to all the destination sub-areas. The query consists of the MC and MS descriptions, the search scope angle (α) and a chain history. The chain history (initially

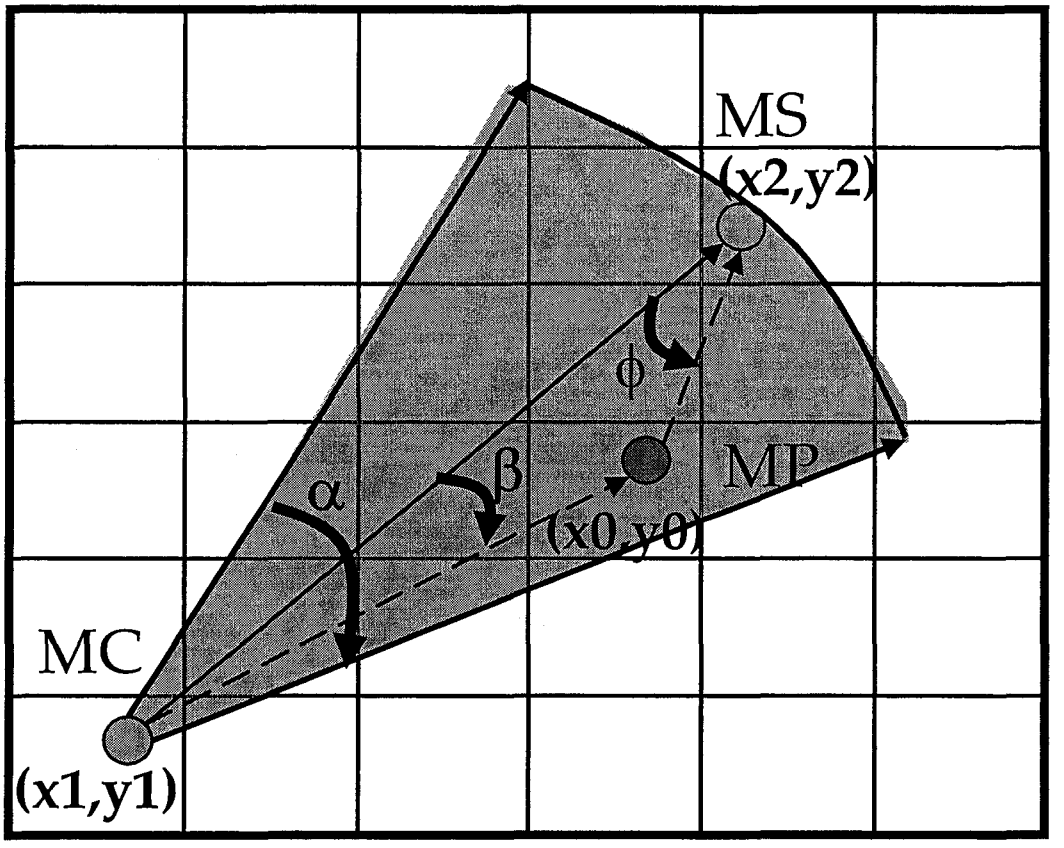

Fig. 5.5 Network Geographical Area and Search Scope Angle

empty) accumulates all the possible adaptation from all MPs that this query visits, which basically results in a list of paths; each path representing a possible solution for the media flow.

Any global ring node that receives the query sends it to its local ring nodes only if it is within α.It waits until it receives the query again, then it forwards the query to each neighbor in the global ring if it falls within α.

To know that it is within the search scope angle, the shaded area in Fig. 5.5, a global ring node A computes the angles β and ϕ using the following formula:

$$\beta = \sin^{-1}\left(\frac{|(x2-x1)(y1-y0)-(x1-x0)(y2-y1)|}{\sqrt{(x2-x1)^2+(y2-y1)^2}*\sqrt{(x1-x0)^2+(y1-y0)^2}}\right) \tag{5.4}$$

Where $(x1, y1)$ is the MC location, $(x2, y2)$ is the MS location, and $(x0, y0)$ is the location of A. The computation of ϕ is similar to the computation of β except that the MC and MS locations will be switched in (5.4).

A is within the scope angle if $\beta < \alpha/2 \wedge \phi < 85^{O}$

When a local ring node receives a query, it processes the query only if it is within the search scope α. Processing the query involves: 1) Retrieving incomplete paths from the chain history, 2) the current node adding itself along with its costs, if it can provide a service for the path, 3) the current node creating a new path and adding that path to the chain history, if it can provide a complete or partial service for the MC and MS. The node then forwards the query to its next neighbor in the local ring. Query forwarding will be terminated when the query is received by the MS. Since the MS might not be a MP, each global ring node checks to see if it is in the same area as the MS, and if so, it will serve the query and send it to the MS. At this point, the MS investigates the chain history and computes the best path for the media flow, then sends a constructFlowPath message through it to the media client. If no solution is found, the MS sends a failure message to the MC, and the MC can then reinitiate the query with a greater α.

Generally, the algorithm avoids sending the query to large parts of the network where answers are not likely to be found, and also, it considers only MPs in the direct path between the source and the destination. The algorithm also exploits the resiliency of the optimal chordal ring by routing around failed nodes. This is particularly important in a dynamic network, since it ensures that resources are discovered even in the event of failures.

```
Algorithm 2: Joining SORD
   Node S is a MP joined the network. Therefore it has to Join SORD.
Status: = {INITIATOR, ASLEEP, DONE}
   Sinit = {INITIATOR} ' initial states
   Sterm = {Done} 'termination state
INITIATOR  ' the node S
  Spontaneously
   { send(JoinRequest, TTL) to S(x) 'all nodes reachable within one hop
     Become ASLEEP }
ANY 'any node independent from its  current status
   Receiving(JoinRequest,TTL) {
      If (localRingNode) reply(JoinOffer) 'if not an optimal chordal ring node
        ' contains its access point, Succ, Pred, and Service Type
      else if (globalRingNode) {
           if (globalRingNode.SubArea = S.SubArea) reply(JoinOffer) 'same sub-area as S
                'JoinOffer contains globalRingNode's Succ, Pred in local ring
           else (globalRingNode.SubArea = S.SubArea -1)
                reply(JoinAccessPoint) ' if S is the first in its sub-area
                'JoinAccessPoint contains the 4 global ring connections
    } else if (TTL>0) send(JoinRequest,TTL) to its S(X) – [received from] }
ASLEEP ' the node S
  { collect(JoinOffer) messages
  if (∃ JoinAccessPoint  Λ  ~∃ JoinOffer ε S.SubArea)
     become an access point for its sub-area
  else {
   S chooses a node V that offers the same (or close) Service type
   Send(OfferAccept) to V
   V sends a message to V.Succ informing it to change its pred to S
   V sets V.succ = S
   S.succ and S.pred sends(ConfirmJoin) to S
   S sets its succ and pred. accordingly and become DONE }}
```

Fig. 5.6 Joining SORD Algorithm

```
Algorithm 3: Leaving SORD
   Node S is a MP that wants to leave the network.
Status: = {INITIATOR, DONE}
   Sinit = {INITIATOR} ' initial states
   Sterm = {Done} 'termination state
INITIATOR ' the node S
  Spontaneously
   If (s ∈ LocalRing) Sends(LeaveMessage) to S.succ. And S.pred
    'S.succ and S.pred connect to each other
  else { 'S an optimal chordal ring node or an access point
    If (∃ LocalRing) {
          S.succ become the new Access point
          send (newAccessPoint) to S.succ, S.pred, S.chord1, S.chord2
    } else { 'S.GSucc in the global ring virtually hosts this subArea because it is empty
       Send(HostVirtualAccessPoint) to S.Gsucc ' contains all S connections
       S.succ acquires all the connections of S }
Become DONE
```

Fig. 5.7 Leaving SORD Algorithm

5.7.2 Joining and Leaving SORD (Intentionally)

When a new MP joins the network, it does not become part of the existing SORD, and does not receive search messages. To correct this, it has to join SORD using the Join algorithm in Fig. 5.6.

The new MP sends a join request to its neighbors, which will forward that request to their neighbors, until the request reaches a SORD node or a node that is aware of a SORD node. The SORD node that receives a join request sends a join offer only if it is a local ring member, otherwise it sends its known local ring members to the MP. The MP then collects join offers and selects the suitable local ring type and joins it. According to the algorithm, an MP is only allowed to join the local ring in its sub-area. This minimizes the messages exchanged and ensures the stability of the global ring. If a MP joins from a sub-area where there is no local ring, it becomes the access point for its sub-area and acquires its connections from its global ring successor that was virtually hosting the empty sub-area. When an MP wants to leave the network, it calls the Leave algorithm in Fig. 5.7; it simply sends a leave message instructing its successor and predecessor to connect to each other. If it is an access point, its successor in the local

ring becomes the new access point. If it is the last node in its sub-area, the global ring successor virtually hosts it. In the worst case, the Leave algorithm costs 4 messages.

```
Algorithm 4: Broken Global Ring
Status:= {INITIATOR, IDLE, ASLEEP, WAITING, DONE}
        MPinit = {INITIATOR, IDLE}
        MPterm = {Done}
INITIATOR ' an optimal chordal ring node
        Spontaneously {
                Send(GlobalRingCheck,TTL) to MP(x) 'to 4 ring neighbors
                Set timer T
                Become ASLEEP ' until T expires or receive OK }
IDLE
        Receiving(GlobalRingCheck,TTL) {
                Return(OK) ' to the sender
                if (received from CH1 or CH2)
                   Send(GlobalRingCheck,TTL-1) to S(x) – {sender}
                Else if (received from a ring peer i.e not CH1 or CH2)
                  If (TTL-2 != 0)
                  Send(GlobalRingCheck,TTL-1) to ring peers – {sender}
                Set timer T
                Become ASLEEP }
ASLEEP
        If Receiving(OK) Become DONE
                ' from all peers that this node sent messages to.
        Else If  T > λ
          If Not ping(predecessor(k))'assume a broken link
                Do BrokenLink(k)
ANY 'any node independent from its status
        Receiving(BrokenCheck) {
                If Ping(predecessor) send BrokenCheck to predecessor
                Else return predecessor(v) }
        Receiving(BrokenCheckOnChord){
                send BrokenCheck to successor
                ' successor pings its CH1 peer and save the result and
                ' sends the message to its' successor}
WAITING
        Receiving(predecessor(v)) {
                If (v = k) do singleRepair(k)
                Else do multipleRepair(k,v) }
        receiving(BrokenCheckOnChord) {
                'assign a node for each failed node. And instruct them to
                                'take their rolls.}
        Become DONE
procedure BrokenLink(k)
        { Send(BrokenCheck) to CH1 peer
          Become WAITING }
procedure singleRepair (k)
        { 'take the rolls of k by connecting to its successor and chords.
          Become DONE }
procedure multipleRepair(k,v)
        { Send(BrokenCheckOnChord) to CH2 peer
          Become WAITING }
```

Fig. 5.8 Broken Global Ring Algorithm

5.7.3 Broken SORD

In a dynamic network, nodes not only join and leave the network; they may also unexpectedly disappear, which causes SORD to break. If a MP unexpectedly leaves SORD, its negative effect depends on the ring it belongs to.

```
Algorithm 5: Broken Local Ring
  MP didn't receive a periodic check message from its Local Peer.
Status:= {INITIATOR, DONE}
      MPinit = {INITIATOR}
      MPterm = {Done}
INITIATOR ' the MP
      Spontaneously
      Begin
            if ( NOT Ping(MP.pred)) { 'if unable to ping its pred. i.e MP.pred failed
              if (Ping(MP.pred.pred)) { ' known from previous LocalRingCheck
                if (AccessPoint(MP.pred)) MP uses algorithm 2 to join global ring
                else { set MP.pred = MP.pred.pred, send(changeSucc) to MP.pred }
           } else { repeat until (Ping(Mp.pred.?)),
                        'ping the next pred until we reach a live one
                        set MP.pred = ?, send(changeSucc) to MP.pred.? }
            }
            Become DONE
      End
```

Fig. 5.9 Broken Local Ring Algorithm

If the node is a global ring member, its local ring may become disconnected. MPs in that sub-area cannot be reached by search messages. However, all the other sub-areas are reachable. This is because of the fault-resilient property of the optimal chordal ring—the multiple paths between any two nodes. However, the problem must be detected and corrected in order to restore SORD to its normal operations. The detection is done by a periodic GlobalRingCheck Message sent on the global ring. If a MP does not receive this message after a certain amount of time, it assumes a broken ring and runs the algorithm in Fig. 5.8.

If the node is a local ring member, the global ring is unaffected, and search messages are routed normally. The local ring, however, is disconnected. Using the LocalRingCheck message and the algorithm in Fig 5.9, local ring nodes are able to detect and correct broken links.

The Broken Global Ring algorithm makes use of the optimal chordal ring's low broadcast time. Traditional algorithms forward the periodic ring check message to one node after another. This makes use of all possible links, and forwards the check message through a maximum subset of links that guarantees $\Theta(i)$ time units. The result is a substantial decrease in time complexity compared to traditional methods that requires $\Omega\left(2\left(i^2+i\right)\right)$. The algorithm also checks for group failures, by sending a BrokenCheckOnChord message to the opposite direction of the ring. It can therefore detect up to $c-1$ failures using $c-1$ messages for periodic checks, and $4c-5$ messages for failure checks. Since the optimal chordal ring is symmetric, and the edge and the node are transitive, the algorithm can be initiated by any node. The algorithm therefore alternates between all nodes in each consecutive N periodic check to guarantee that all links are being tested.

5.8 Degrees of Freedom for SORD

All the algorithms presented here assume that each local ring member knows only its predecessor and successor in the ring, and the four neighbors for each access point in the global ring. While this establishes the locality of updates, it restricts the ring structure. Not all nodes in a local ring are likely to bring the media closer to its destination. We can assume that the access point knows all its sub-area members, but it can choose to send the query message only to a subset of them (though not if a sub-area contains a large number of MediaPorts).

Alternatively, we can subdivide each sub-area, which allows each local ring to be constructed in such a way that geographically close nodes are connected to each other. This is similar to the construction of the global ring, except that each local ring is

traditional. The degree of freedom this provides is most appropriate if there are a large number of MediaPorts in each sub-area. A hybrid approach can also be used to construct a star-like structure in sub-areas with a small number of MediaPorts and a ring structure in sub-areas with a large number.

Another degree of freedom lies in choosing the access points to the sub-areas. Access points have an upper arm connecting the global ring and a lower arm connecting the local rings. Using both rings, they should therefore be able to handle a large number of requests. Additionally, their location inside their sub-areas should allow for and contribute to global ring construction. Techniques for choosing super-peers in P2P networks can therefore be reused without the need to modify SORD algorithms.

Another degree of freedom is to use the number of failures (f) to enhance SORD's robustness. If the number of failures is high, the value of f is increased. When f exceeds a certain threshold value, the number of nodes that have a backup of the global ring node information is increased. This is done by requiring neighboring network nodes to host the same type of connections and information. For example, when $f = 0$, no immediate network node is required to host the global ring node information, and when $f > 2$, the direct neighbors of the global ring node host the information. Although this duplicate information is a clear overhead, it provides SORD with the following advantages: 1) The global ring becomes more robust to node failures, 2) the discovery cost (Fig. 5.5) is significantly low, as many nodes in a given sub-area have knowledge of SORD, and 3) the load on the original global ring node may be distributed to neighboring nodes when the original node is unable to handle the incoming requests.

Another degree of freedom is to improve the way queries are processed in SORD. Due to the lack of knowledge about the needed MPs, queries in SORD are processed in a sequential-chain. This can be parallelized by sending the query to global ring nodes (access points) all at once. Each ring node sends the query to its local ring, based on whether it provides a service or not. Each access point then sends replies to the MS, either using SORD or a shortest path algorithm. The MS retrieves the path history and builds a service graph that can be searched for the best solution.

Lastly, since each MP may belong to more than one SORD, and may provide more than one service, a multi-dimensional routing algorithm can be used to route queries efficiently. This is achieved by using different SORDs, though only one SORD is used for the answer. This knowledge can be also used to discover a specific SORD by routing the discover message to all known SORDS in a given MediaPort.

5.9 Simulation Details and Results

We used a discrete event simulator to evaluate the performance and efficiency of SORD. The topology was constructed using the BRITE [169] Topology Generator, and the network was simulated using the J-Sim network simulator [170], a simulator with a Java(tm)-based engine. SORD's structure is built in a way that supports the construction of SSONs. Constructing SSONs has been proposed by several methods, while SORD's structure is similar to DHT approaches. Thus, evaluation of SORD consists of assessing its efficiency in constructing SSONS compared to existing methods; determining the effect of SORDs' specific parameters and their relation to each other; and comparing SORD to DHT approaches. To this end, we conducted three experiments. The first compares SORD's discovery mechanism with Limited-Flooding (LF) and Path-Directed (PD) approaches [178], [179]. Limited-flooding has been predominantly used to discover services in environments such as ad hoc and pervasive networks. The path-directed protocol starts from the source and expands along the end-to-end routing path towards the destination node, with a sideway expansion of a given distance (e.g. based on the number of hops, delay, etc...) After visiting the nodes defined by the protocol, it contracts towards the source node, gathering the requested information (depending on the resources/services we are looking for). The sideway expansion parameter of the protocol controls the scope of the search and thus limits the number of nodes to be probed. To examine the effect of granularity, $R_3(25,7)$, $R_5(61,11)$ and $R_{10}(221,21)$ optimal chordal rings were simulated. None of these approaches have advertisements for the offered services. In the following, we use SORD to denote R_3, R_5 and R_{10} unless the distinction is necessary. The second experiment evaluates the efficiency of SORD under

various parameters such as mobility, search angle, and service density. And the last experiment compares the message cost of SORD to two popular DHT based approaches.

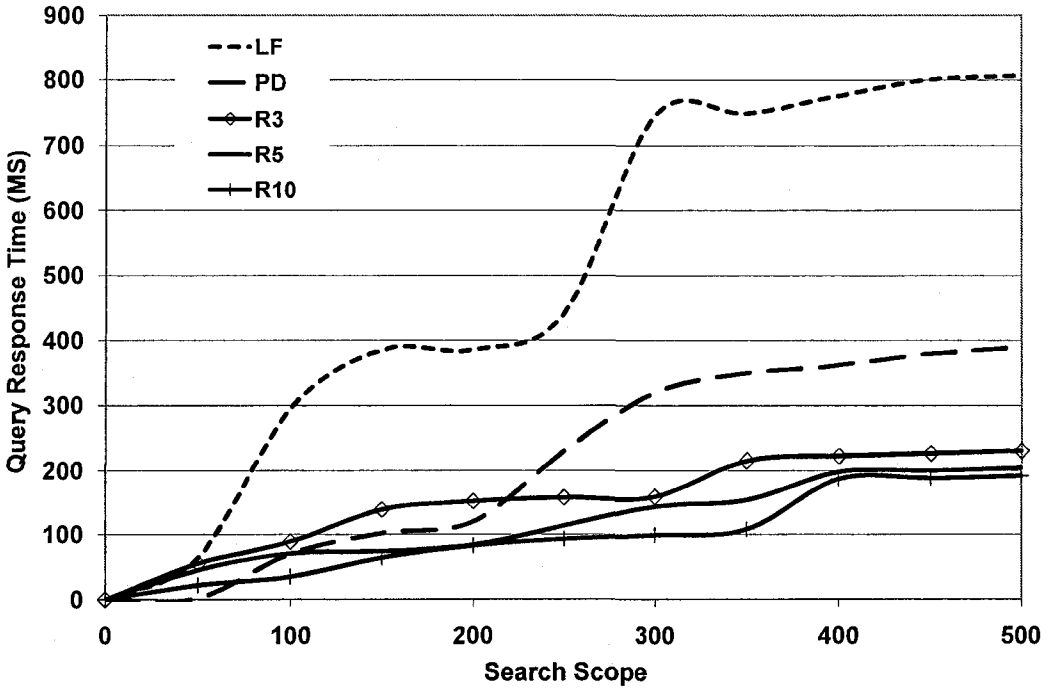

Fig. 5.10 Average Response Time

5.9.1 Simulation Setup

The topology used hosted 3000 nodes in a 5000 X 5000 node two-dimensional overlay space. The bandwidth assigned to each node was randomly selected between 128 and 512 kbits/s. Each node had a random geographical location. To follow a flash crowd characteristic, all nodes issued their queries at a random point during the first 30 seconds, with the simulation lasting for another 1000 seconds. We ran the simulation a number of times with different search scope values, which can be any metric useful to measure the network distance of an end-to-end service path between source and destination. Examples are the number of hops or the aggregate delay. In our case, we have used the end to end delay, measured as the Round Trip Time (RTT) in

milliseconds. For each run, a random number of queries (between 2000 and 4000) were requested. The results were collected after each run.

5.9.2 Experiment 1

In the first experiment, we measured response time, query cost, and success rate. In this experiment, the search scope angle in SORD, α, is fixed at 35 for all queries.

5.9.2.1 Average Response Time

The response time for discovery requests is the difference between the starting time of the search and the arrival of the complete result set. Fig. 5.10 shows that the average response time of LF approach is at least one and a half times higher than the average response time of the PD approach. The average response time observed in R_3, R_5, and R_{10} is much lower still. We believe that the decrease in response time is primarily due to two factors: First, the decrease in the overhead of service replies (see Fig. 5.12). This is because in LF and PD, service replies travel a greater number of hops than in SORD where the number of hops is reduced. Second, the decrease in the overhead due to search messages (see Fig. 5.11). This is mostly because service requests were routed directly to the nodes where answers were most likely to be found. Furthermore, the average response time observed in R_{10} is lower than that of R_3 and R_5. The reduced granularity in R_{10} results in a small local rings at each sub-area. Thus service requests and replies take less time to go through the entire local ring.

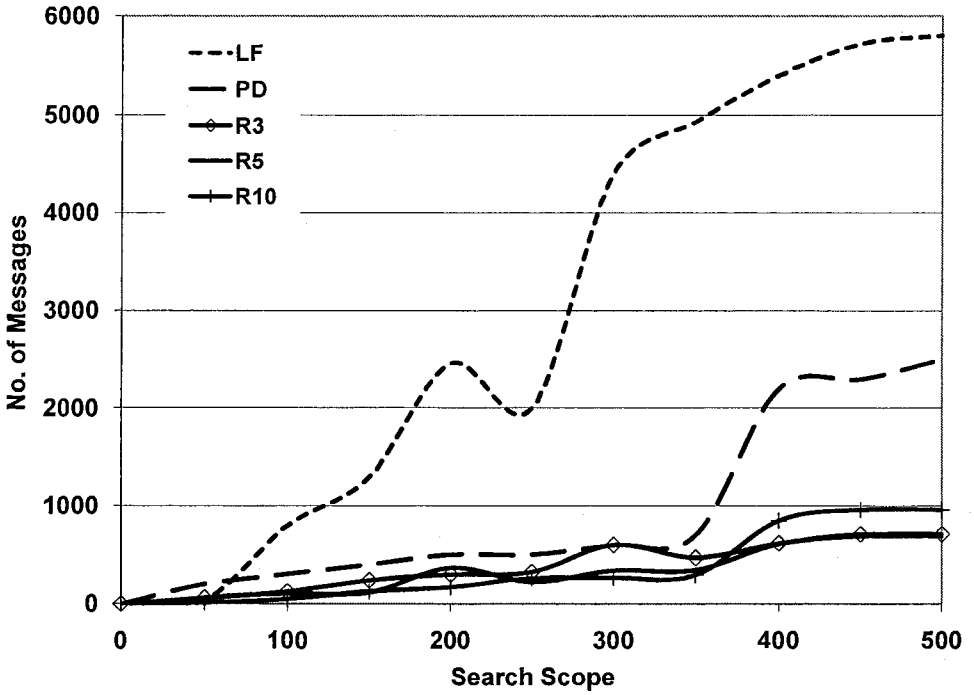

Fig. 5.11 Overhead Due to Search Messages

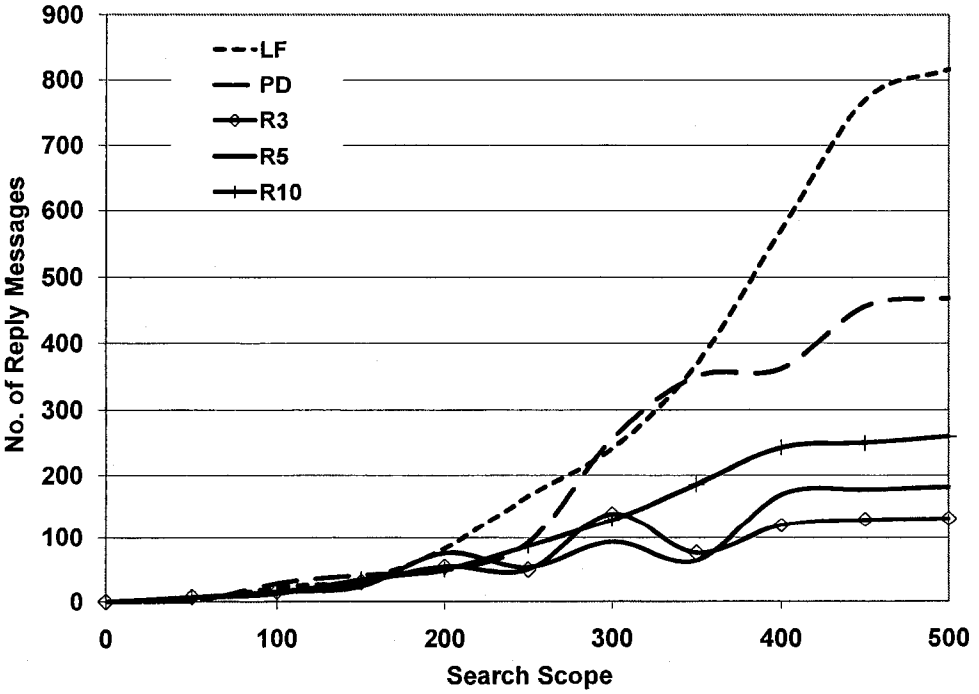

Fig. 5.12 Overhead Due to Query Responses

5.9.2.2 Query Cost

This quantifies the cost of searching SORD for services. Query cost is composed of 1) the total number of search messages (the total number of hops taken by all queries divided by the number of queries) and 2) the total number of reply messages (the total number of hops taken by all reply messages divided by the number of replies). In SORD, the total number of hops is the sum of the total number of hops in the discovery stage and the total number of hops taken by the query. The discovery stage is present only in SORD and is required only once for each node. Fig. 5.11 shows that LF has the worst performance: It produces a greater number of search messages, except in searches with small scope values. This is because we need to discover SORD first before we can use it to route the query. Understandably, the discovery stage in SORD is similar to LF with small TTL values. The consequence is that, with small search scope values, there will be an overlap between the discovery and routing stages. So to reduce the message cost for small search scope values, the discover message should also be considered as a service request, which increases the chances of finding a service match before the query is routed on SORD. But as the search scope increases, the number of messages in LF and PD is at least two times higher than the number of messages in SORD. For search scope values less than 350, R_{10} produces less number of messages compared to R_3 and R_5. With larger scope values, R_{10} tends to generate more messages while R3 and R5 behave similarly. This is because in R_{10} the number of sub-areas is much higher than in R_3 and R_5. This increases the number of sub-areas between the media end points, thereby increasing the number of messages routed in the global ring.

The overhead due to query response is shown in Fig. 5.12. For small search scope values (below 230), SORD has a larger overhead. This is because SORD routes service requests to all the available MPs in targeted sub-areas, resulting in a larger number of service replies. However, for larger search scope values, SORD outperforms both PD and LF approaches. R_{10} also produces larger replies than R_5, and R_5 produces larger replies than R_3. We believe that this is primarily due to 1) the way the service replies are

routed on SORD as discussed in section (5.6) and 2) the increase in the number of the global ring nodes.

5.9.2.3 Success Rate

Success rate measures the accuracy of SORD, and is defined as the number of requests that receive positive responses, divided by the total number of queries. Fig. 5.13 shows that SORD results in a higher success rate, except for small search scope values, for which LF is more effective (though it did not reach the 100% success rate that the PD approach attains after a certain search scope value). However, SORD reaches the 100% success rate earlier. We believe that this is due to the huge network load generated by LF. For large search scope values, LF generates a large number of messages and receives a large number of reply messages. As a consequence, messages are dropped or lost due to collisions. In the PD approach, the messages are controlled by the distant

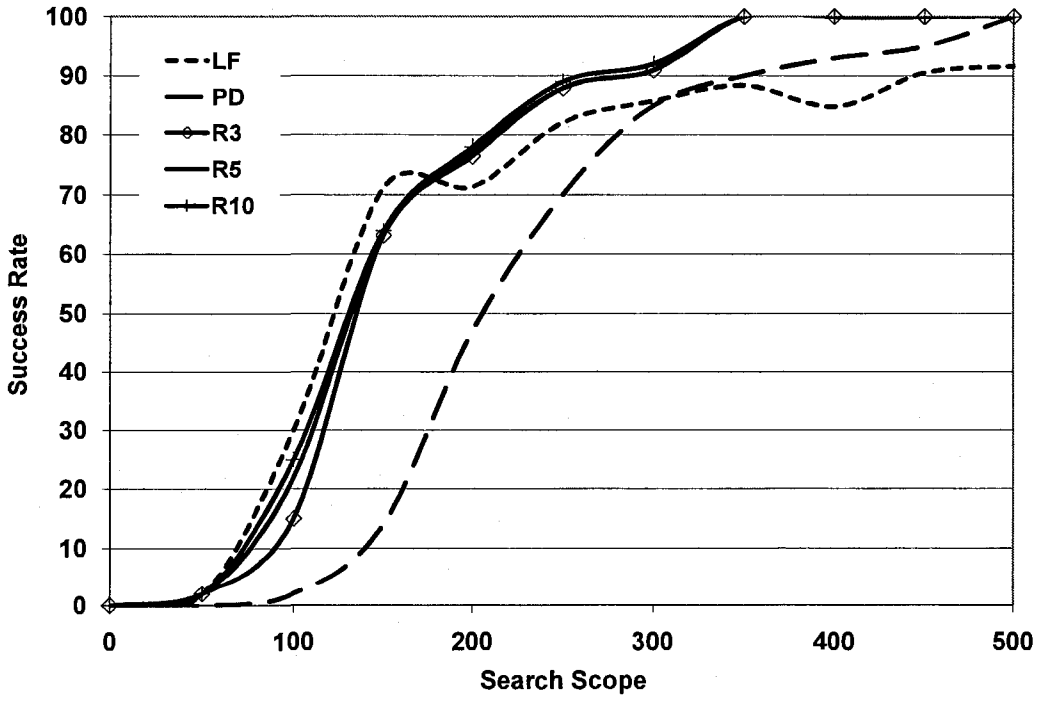

Fig. 5.13 Success Rate

function that reduces their number. By contrast, SORD generates the lowest number of messages; as we have seen, this is by routing service requests only to sub-areas located in the direct geographical path between the end points. R_3, R_5 and R_{10} have almost the same success rate. This is because all MediaPorts in the direct path between the media end points will be reached by R_3, R_5 and R_{10} regardless of their differences in the response time and query cost.

5.9.2.4 Initial Cost

Since LF and PD have no initial cost for building a structure, the results are presented without the initial construction cost for SORD as well. Generally, we can construct SORD by building a spanning tree between MPs and broadcasting the geographical sub-areas, node address in SORD, and the connections for the global ring in that spanning tree. The cost of these steps is the initial construction cost of SORD and is given by:

$$M(SORD/INI) = 4m - 2n + 3(n-k) + 2 \tag{5.5}$$

Where m is the number of links, n is the number of nodes and k is the number of MediaPorts. Then, assuming $k << n$, the cost becomes:

$$M(SORD/INI) = 4m + n - 1 \tag{5.6}$$

Therefore, the initial cost complexity is $\Theta(m)$ and the time complexity is $\Theta(d)$, where d is the diameter of the network. Generally, the cost is equivalent to about 50-60 queries, which indicates that SORD is most suitable for applications where a high number of queries is expected. The initial construction cost is compensated for by the low query cost, the improved response time, and success rate.

Alternatively we can use a mechanism similar to the bootstrap mechanism proposed by [180]. We assume that SORD has an associated DNS domain name, and that it resolves to the IP address of one or more SORD bootstrap nodes; this maintains a list of SORD nodes that are currently present in the system. To join SORD, the new node uses the DNS to retrieve a bootstrap node that will supply it with several SORD nodes currently in the system. The new node then sends the join requests to one of these nodes to be forwarded to its geographical access point.

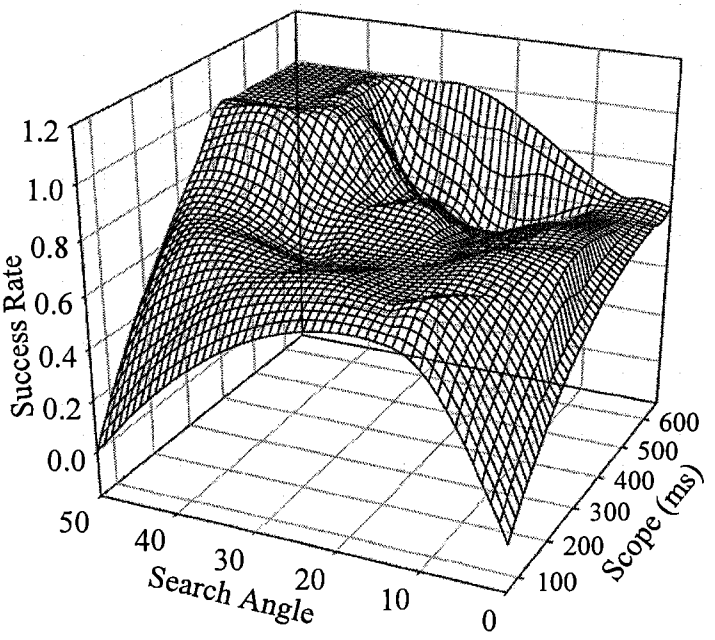

Fig. 5.14 Success Rate as a Function of Scope and Search Angle

5.9.3 Experiment 2

In the second experiment, we evaluated the efficiency of SORD with extensive measurements of the success rate and the overlay path stretch as a function of the following parameters: 1) Search scope, 2) search scope angle α, 3) service density, 4)

number of mobile nodes. The search scope is the same as in the previous experiment. Angle α defines how many sub-areas are being searched in a given search query. Intuitively, increasing α increases the success rate but also increases the query cost. It is therefore essential to decide on the best initial α to be used. Service density refers to the number of distinct services in the network. We assume that each MP offers only one service drawn randomly from a set of 600. Each query searches for a service selected randomly from the same set. Even if the service is not present in the network explicitly, it can be provided by chaining two or more MPs. Finally, to test the efficiency of the SORD in the presence of churns, mobile nodes are introduced. The topology parameters in this experiment are the same as in section 5.9.1. The only difference is that each node is equipped with a wireless interface. The MAC layer uses the IEEE 801.11 protocol and the mobility model for each node is a Random Waypoint. Each mobile node moves average speed of 5 km/hour [181].

5.9.3.1 Scope vs. Search Angle α

Fig. 5.14 shows the success rate of SORD when both scope and α are variables. Service density and mobility are set to 600 and 30% respectively. Increasing both (scope and α) increases the success rate. This figure suggests that choosing α in the range [30-40] guarantees a 100% success rate for scopes greater than 3000. While α in [15-30] attains a 100% success rate for larger scope values. Although α in [40-50] has a higher success rate than [30-40], the increase is not substantial. These observations suggest that applications should use 30 as an initial search scope angle, and if the desired results are not found, α should be increased by 5.

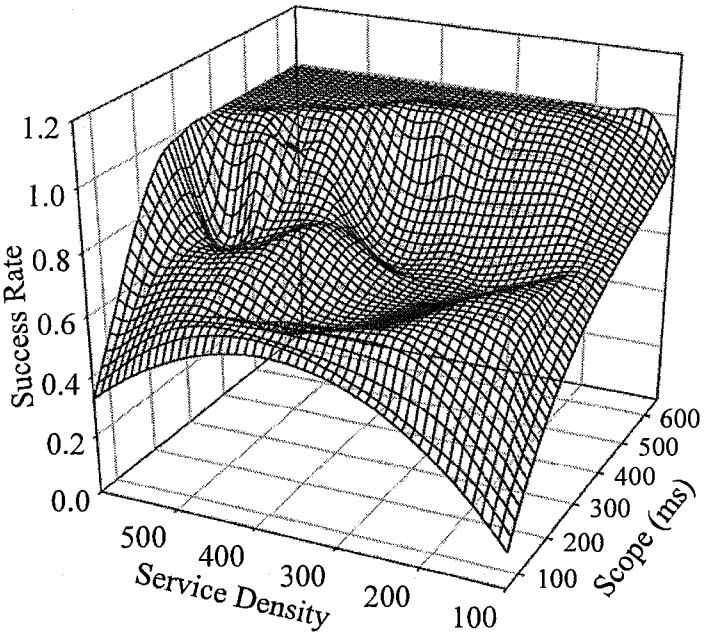

Fig. 5.15 Success Rate as a Function of Scope and Service Density

5.9.3.2 Scope vs. Service Density

Fig. 5.15 shows the success rate of SORD when both scope and service density are variables. α and mobility are set to 30 and 30% respectively. Increasing the service density increases the success rate. However, even for low service densities, SORD achieves a 100% success rate for scopes greater than 550. This observation supports the previous finding that $\alpha = 30$ is a good initial choice.

5.9.3.3 Scope vs. Mobility

Mobility is an important challenge in a dynamic network. The MC (or user) might move to another location and the MP providing the service might be mobile or become

unavailable due to a power limitation. The mobility of nodes affects SORD in that each time a node moves away from its sub-area, the Leave algorithm will be executed and each time a node enters a different sub-area, the Join algorithm will be executed. This varies depending on whether the moving node is a local or global ring member.

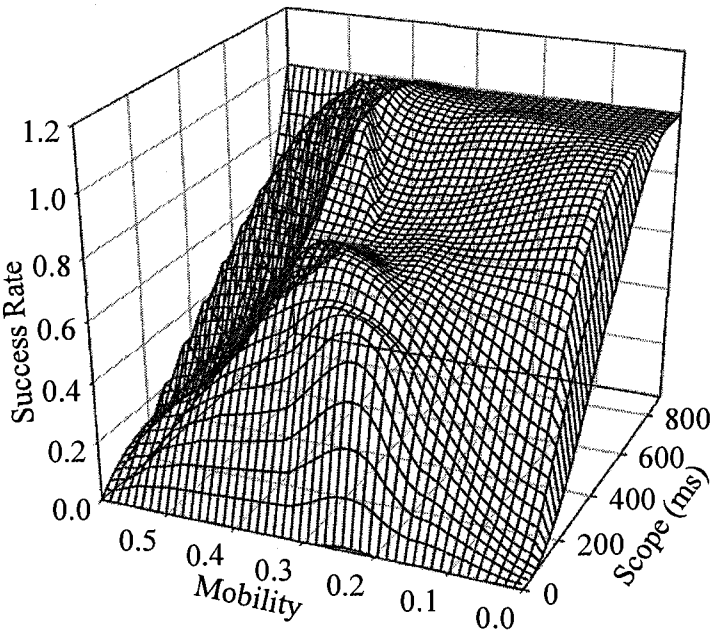

Fig. 5.16 Success Rate as a Function of Scope and Mobility

Fig. 5.16 shows the success rate of SORD when both scope and mobility are variables. α and service density are set to 30 and 600 respectively. We ran the simulation 12 times. Each run increased the number of mobile nodes by 5% by random selection. In each run and for each scope value, we issued 25 queries and computed the success rate. We observe that mobility < 35% has a limited effect on success rate. An observable effect appeared for mobility > 50, though SORD still achieves the 100% success rate. We believe that this is due to: 1) The small routing table that SORD maintains because less time and messages are needed to fix changes, 2) new nodes are only allowed to join

the local ring which leaves the global ring unchanged and operational. Only a small number of existing nodes in a very small locality are affected, 3) the ability to route around failures.

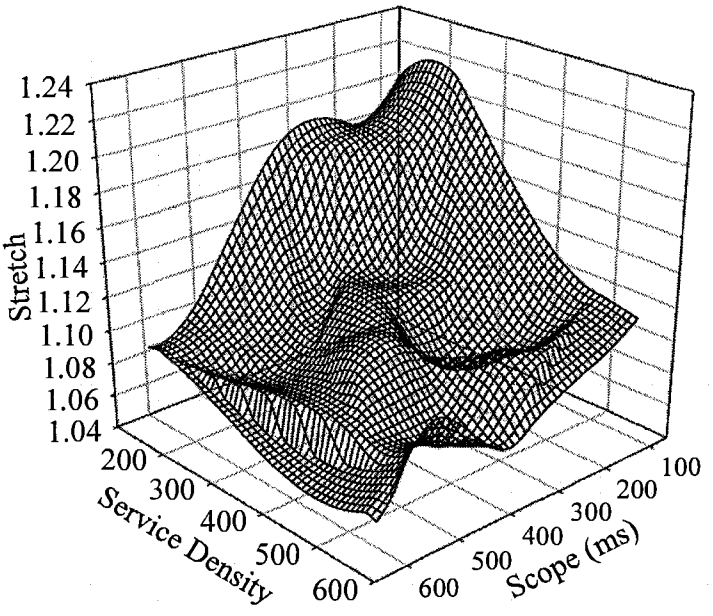

Fig. 5.17 Overlay Path Stretch as a Function of Scope and Service Density

5.9.3.4 Stretch

Stretch is defined as the number of hops taken by an overlay packet divided by the number of hops the packet takes when using an IP-layer path between the same source and destination. A high stretch value indicates an inefficient SSON topology as longer routes delay the packets. Fig. 5.17 shows the stretch when both scope and service density are variables. α and mobility are set to 30 and 30% respectively. The figure is rotated so that the higher service density and scope values are shown in the front. When the number of services increases, the stretch decreases. The results show that for search scope values > 250 and service density > 250, stretch varies from 1.04-1.1. For smaller

values it varies from 1.1-1.22. Generally, the stretch is not significant considering the gains in other measurements.

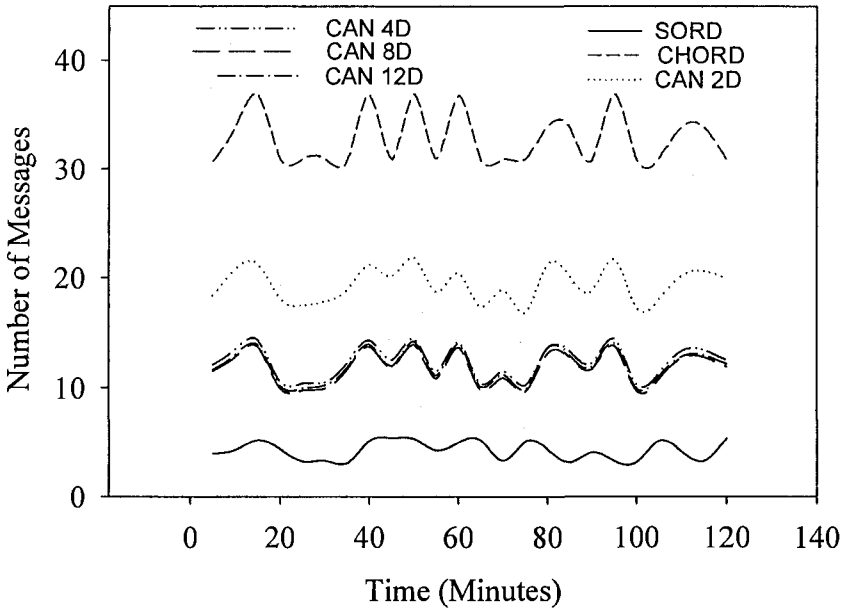

Fig. 5.18 Overhead as a Function of Time

5.9.4 Experiment 3

In this experiment, we compared flat SORD, CAN [116], and CHORD [54] protocols. For the sake of realistic comparison, SORD nodes stores <key,value> pairs and the network has 106 keys. Key lookups are generated according to a Poisson process at a rate of one per second. Joins and failures are modeled by a Poisson process with the mean arrival rate of one per 60 seconds. For CAN and CHORD, each node periodically runs the stabilization routines at randomized intervals averaging 30 seconds; all finger table entries are updated on every invocation of the stabilization routine (Both CAN and CHORD use the same stabilization algorithm proposed originally by the CHORD protocol). The stabilization algorithm maintains a successor list at each node; a successor list of size r maintains r connections at each node pointing to the first r

successors in the CHORD ring. The network is strongly stabilized when $r = 2\log N$. As a result, the total number of states maintained by each node is the sum of the routing table size and the successor list size.

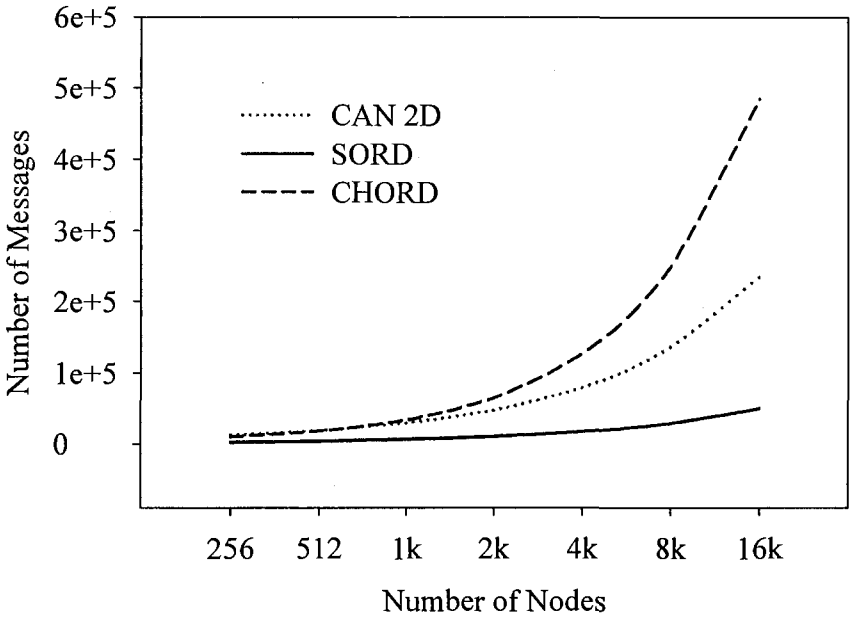

Fig. 5.19 Overhead as a Function of Network Size

Fig. 5.18 plots the total number of messages generated by each protocol during a 2 hour interval. The total number of messages is normalized by the network size (4096 nodes), and computed as the sum of the messages generated due to lookups and maintenance. CHORD maintains a successor list of size $\log N$. In addition to the periodic refreshes sent by each node to its neighbors, CAN maintains a successor/predecessor list of size 2. We observed that SORD generated the lowest message overhead. Increasing the CAN dimensions to a certain limit decreases the total number of messages. We observed that this limit occurs when $d = 12 = \log N$. (Increasing the dimension decreases the lookup cost). Total message cost is well beyond that of SORD due to the CAN large

maintenance cost. CHORD maintains more nodes in the successor list, thus incurring the highest message overhead.

Fig. 5.19 shows the total number of messages (averaged over 2 hours) as a function of network size. Although SORDs' lookup cost increases at a greater rate than CHORD, the maintenance cost in SORD is very low compared to CHORD. We believe that, in addition to the low number of connections per node, this is due to the symmetry of the optimal chordal ring of degree 4. A 2 dimensional CAN maintains 4 connections per node and 2 connections for the successor list. While this is close to the 4 connections in SORD, the lookup cost in a 2 dimensional CAN grows faster than that of SORD. These results show that the increase of lookup costs in SORD is compensated by the decrease in maintenance cost.

5.10 Scalability

One common way to improve the performance of a network is to increase its connectivity and decrease its diameter, and this can be done by adding links. However, we want to add as few links as possible since their cost has practical implications in the design. Additionally, the number of links going out of a node must be small to allow for fast maintenance. The links must be added in a homogeneous way so that nodes can be easily inserted, and messages can be routed systematically.

Dynamic Hash Table (DHT) approaches are decentralized. They support scalable and distributed storage, and retrieval of $(Key, Data)$ pairs on the overlay network. In a network of N nodes, where each node maintains $O(\log N)$ routing entries, DHTs generally perform lookups using only $O(\log N)$ overlay hops (CAN [116] is an exception). In contrast, the proposed discovery approach, SORD, has only 4 links per node independent of N (the number of nodes in the network). It routes in $O(k)$ hops (where k is the diameter of the optimal chordal ring). For relatively small N, $k < \log N$. For larger N, $k = c * \log^2 N$ (where c is a variable increasing with N). For N = 190000, c = 1

and for $N=10^6$, c= 1.8. CAN [116] is an exception in that it routes in $O(dN^{1/d})$ hops (where d is the dimension) with a routing table size $O(dr)$ which is independent of N. Setting $d = \log N$ allows CAN to match the scaling properties of other DHT systems in that it routes in $O(\log N)$ hops and requires a routing table size that is $O(\log N)$. However, CAN is not designed to vary d as N (and thus $O(\log N)$) varies, therefore this match will only occur for the "right" N corresponding to a fixed d. Setting $d=2$ allows CAN to match SORDs' routing table size of $O(4)$ but increases the lookup cost to $O(2N^{1/2})$. This implies that while SORD lookups is more expensive compared to DHT systems (except CAN with $d=2$), SORD topology is more stable in a dynamic network. This is because SORD uses a small routing table size that requires less work and less time to fix any change in the topology. These results are evident in figures 5.18 and 5.19. However, to increase scalability and fault isolation, hierarchies are introduced. It can be seen that SORD trades lookup cost for more efficiency and flexibility.

5.11 Summary

In this chapter, a novel scheme for semantic resource discovery has been presented. The proposed scheme allows services to be found without relying on centralized directory servers, and also minimizes query cost and response time. The approach is targeted for SMART [2], but it can be adapted to a wide variety of applications, such as P2P and ad hoc networks. The proposed overlay structure is based on a widely-studied family of chordal rings (the optimal chordal ring of degree 4), the semantics of the offered services, and the physical location of nodes. It is fault-resilient because of the multiple paths between ring nodes, allowing queries to be routed optimally between any two nodes. It has been shown that the proposed scheme is efficient in query cost, accuracy, and query responses. In addition, results suggest that decreasing the granularity will decrease the response time but increase the query cost. Moreover, an initial search angle, α, of 30 degrees is sufficient to give an acceptable success rate.

Chapter 6

Towards an Autonomic Service Architecture

As discussed earlier, IT professionals must reinforce the responsiveness and resiliency of service delivery, by improving quality of service while reducing the total cost of their operating environments. Yet, information technology (IT) components over the past decades are so complex that they increase the challenges to effectively operate a stable environment. Overlay networks management complexity is turn increased by the huge number of users, terminals, and services. Although Human intervention enhances the performance and capacity of the components, it drives up the overall costs—even as technology component costs continue to decline. Due to this increased management complexity, this chapter gives an overview of autonomic SSONs; it proceeds as follows: Section 6.1 introduces autonomic overlays management challenges. Section 6.2 discusses Autonomic overlays. Section 6.3 identifies required knowledge and their types, while section 6.4 proposes different policy types used to realize autonomic entities interactions. Finally, section 6.5 presents a discussion and summary for the chapter.

6.1 Introduction

A service delivered to a customer by a Service Provider (SP) is usually formed from a composition of different services. Some services are basic in the sense that they cannot be broken down further into component services, and they usually act on the underlying resources. Other services are composed of several basic services, each consisting of an allocation of resource amounts to perform a function. However, with the increasing demands for QoS, service delivery should be efficient, dynamic, and robust. Current

manual approaches to service management are costly, and consume resources and IT professionals' time, which leads to increased customer dissatisfaction; with the advent of new devices and services, the complexity is further increased. With a large number of overlays, the management task becomes harder to achieve using traditional methods. Therefore, new solutions are needed to allow SPs to support the required services, and to focus on enhancing these services, rather than their management. Autonomic Computing (AC) helps address this complexity by using technology to manage technology.

The concept of autonomic computing (AC) [93] was proposed by IBM to enable systems to manage themselves through the use of self-configuring, self-healing, self-optimizing, and self-protecting solutions. It is a holistic approach to computer systems design and management, aiming to shift the burden of support tasks, such as configuration and maintenance, from IT professionals to technology. Therefore, AC is a key solution for SSON management in heterogeneous and dynamic environments.

Establishing a SSON involves 1) Resource discovery to discover network-side nodes that support the required media processing capabilities, 2) an optimization criterion to decide which nodes should be included in the overlay network, 3) configuring the selected overlay nodes, and 4) adapting the overlay to the changing network context, user, or service requirements, and joining and leaving nodes. In AC, each step must be redesigned to support autonomic functions. In other words, in Autonomic Overlays (AO), each step imposes a set of minimum requirements. For example, the resource discovery scheme should be distributed and not rely on a central entity; it needs to be: Dynamic to cope with changing network conditions; efficient in terms of response time and message overhead; and accurate in terms of its success rate. The optimization step is mapped into a self-optimization scheme that selects resources based on an optimization criterion (such as delay, bandwidth, etc.) and should yield the cheapest overlay, and/or an overlay with the least number of hops, and/or an overlay that is load-balanced, and/or a low latency overlay network, and/or a high bandwidth overlay network. The configuration of the selected overlay nodes in a given SSON is mapped into a self-configuration and self-adaptation. Self-configuring SSONs dynamically

configure themselves on the fly. Thus they can adapt their overlay nodes immediately to the joining and leaving nodes and to the changes in the network environment. Self-adapting SSONs self-tune their constituent resources dynamically to provide uninterrupted service. Our goals are to automate overlay management in a dynamic manner that preserves the flexibility and benefits that overlays provide, to extend overlay nodes to become autonomic, to define the inter-node autonomic behavior between overlay nodes, and to define the global autonomic behavior between SSONs.

This chapter proposes a novel Management Architecture for overlay networks. There are two main contributions brought about by the Architecture: First, we introduce the concept of Autonomic Overlays (AO), in which SSONs and their constituent overlay nodes are made autonomic and thus become able to self-manage. Second, autonomic entities are driven by policies that are generated dynamically from the context information of the user, network, and service providers. This ensures that the creation, optimization, adaptation, and termination of overlays are controlled by policies, and thus the behaviors of the overlays are tailored to their specific needs.

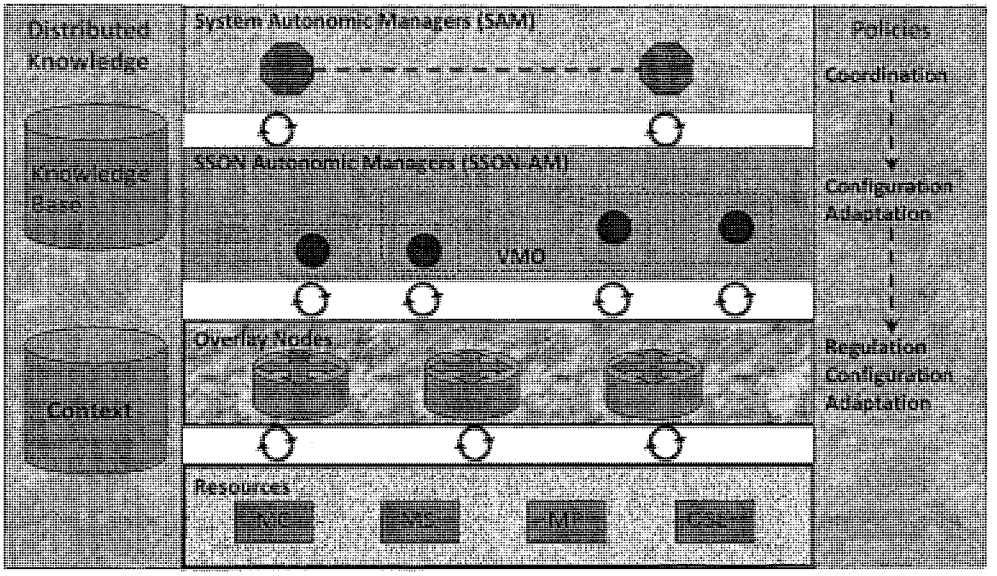

Fig. 6.1 Autonomic overlays architecture

6.2 Autonomic Overlays

To tackle the complexity of overlay management, each SSON is managed by an SSON Autonomic Manager (SSON-AM) that dictates the service performance parameters. This ensures the self.* functions of the service. In addition to this, overlay nodes are made autonomic to self-manage their internal behavior and their interactions with other overlay nodes. In order to ensure system wide performance, System Autonomic Managers (SAM) manages the different SSON managers by providing them with high level directives and goals. The following sections detail the different aspects of our architecture.

6.2.1 Architecture Overview

The set of components that makes up our architecture is shown in Fig. 6.1. The lowest layer contains the system resources that are needed for multimedia delivery sessions. In particular, the Overlay Support Layer (OSL) receives packets from the network, sends them to the network, and forwards them on to the overlay. Overlay nodes implement a sink (MediaClient, or MC), a source (MediaServer, or MS), or a MediaPort (MP) in any combination. MPs are special network side components that provide valuable functions to media sessions; these functions include, but are not limited to, special routing capabilities, caching, and adaptation. These managed resources can be hardware or software and may have their own self-managing attributes.

The next layer contains the overlay nodes. Overlay nodes are physical Ambient Network nodes that have the necessary capabilities to become part of the SSON. They consist of a control plan and a user plan. The control plan is responsible for the creation, routing, adaptation, and termination of SSONs, while the user plan contains a set of managed resources. The self-management functions of overlay nodes are located in the control plan. The Ambient Manageability interfaces are used by the self-managing functions to access and control the managed resources. The rest of the layers automate

the overlays' management in the system using their autonomic managers. SSON-AMs and SAMs may have one or more autonomic managers, e.g. for self-configuring and self-optimizing. Each SSON is managed by an SSON-AM that is responsible for delivering the self-management functions to the SSON. The SAMs are responsible for delivering system wide management functions; thus, they directly manage the SSON-AMs. The management interactions are expressed through policies at different levels. All of these components are backed up with a distributed knowledge. The following sections describe each component in detail.

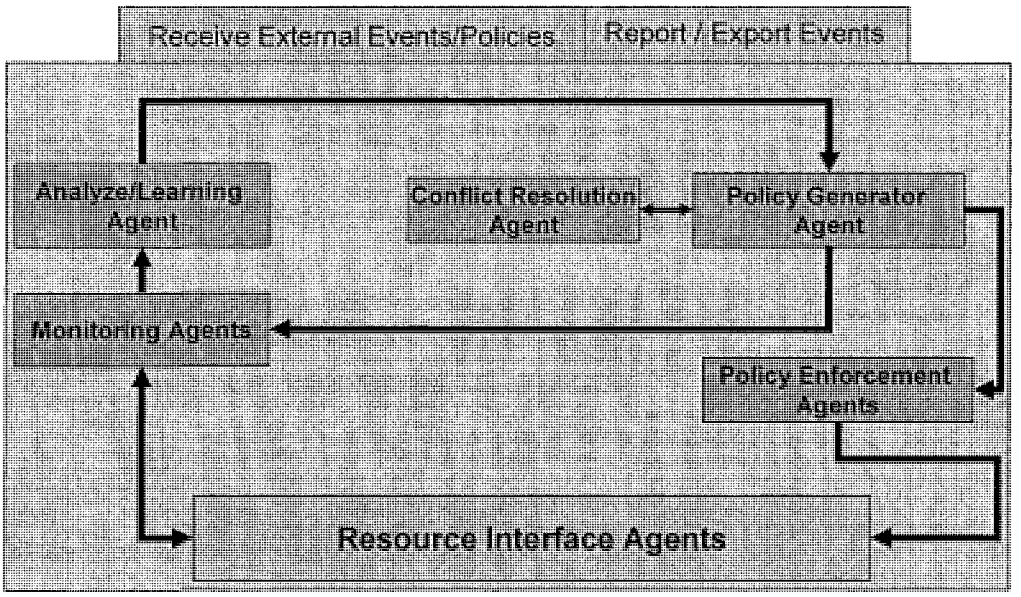

Fig. 6.2 Autonomic control loop

6.2.2 Autonomic Elements

6.2.2.1 Overlay Nodes Autonomic Manager (ONAM)

Each overlay node contains a control loop similar to the IBM control loop [90], as shown in Fig. 6.2. The Autonomic Manager (AM) collects the details it needs from its managed resources, analyzes those details to decide what actions need to change, generates the policies that reflects the required change, and enforces these policies at the correct resources. As shown in the figure, the ONAM consist of the following:

Monitoring Agents (MAs): These agents collect information from the overlay node resources, such as packet loss, delay jitter, and throughput. A MA also correlates the collected data according to the installed policies, and reports any violation to the Analyze/Learning Agent (ALA). For example, an MA for a Caching MP collects information about the MP's available capacity, and whenever the available capacity reaches 10%, it reports to the ALA. Another example is the MA for a routing MP that relays data packets between overlay nodes: Its MA collects information about the throughput and reports to the ALA whenever the throughput reaches a high value. These collected data will be used to decide the correct actions that must be taken to keep the overlay node performance within its defined goals. The MAs interact with the Resource Interface Agents (RIAs) to monitor the overlay node resources availability, and to collect data about the desired metrics. They also receive policies regarding the metrics that they should monitor as well as the frequency in which they report to the ALA.

Analyze/Learning Agent (ALA): This agent observes the data received from the MAs, and checks to see whether a certain policy with which its overlay node is associated is being met. It correlates the observed metrics with respect to the contexts, and performs analysis based on the statistical information. In the case that one of policies is violated, it sends a change request to the Policy Generator (PG). This component is an objective of future work.

Policy Generator (PG): The difference between this control loop and the IBMs' control loop lies in the use of a PG instead of a Plan component. The Plan function – according to IBM [90] – is to select or create a procedure that reflects the desired change based on the received change request from the Analyze Agent. This is not sufficient in our case, where each overlay node receives high level policies and it is up to the overlay node to decide how to enforce these policies based on its available resources. Therefore, we envisioned a PG instead. The PG reacts to the change request in the same way as in the Plan component, although it also generates different types of policies in response to the received high level policies. For example, based on the goal policies received by the overlay node, the policy generator generates the tuning polices and passes them to the

MAs (more about this in Section 3.4). Upon generating new policies, the policy generator consults a Conflict Resolution Agent (CRA) that ensures the consistency of the new generated policies with those that already exist. Generally, we divide conflicts into two types: Static and dynamic. In our model, a static conflict is a conflict that can be detected at the time of generating a new policy, while a dynamic conflict is one that occurs at run time.

Policy Enforcement Agent (PEA): The PG generates suitable policies to correct the situation in response to a change request, and passes these policies to the PEA. The PEA then uses the suitable RIA to enforce them. This includes mapping the actions into executable elements by forwarding them to the suitable RIA responsible for performing the actual adjustments of resources and parameters. The enforced policies are then stored in the Knowledge Base (KB).

Resource Interface Agents (RIAs): These implement the desired interfaces to the overlay node resources. The MAs interacts with them to monitor the availability of overlay node resources and the desired metrics in its surrounding environment. Each resource type has its own RIA that translates the policy actions into an adjustment of configuration parameters that implements the policy action.

External Interfaces: Each overlay node has a set of interfaces to receive and export events and policies to other overlay nodes. These interfaces are essential to enable multiple overlay nodes to cooperate to achieve their goals. In particular, these interfaces are used by the SSON-AM to interact with the overlay nodes that had agreed to participate in the SSON. The SSON-AM sends the system policies to the overlay nodes through these interfaces, through which it also receives reports on their current status.

6.2.2.2 SSON Autonomic Managers (SSON-AM)

SSON-AMs implement the intelligent control loop in much the same way as ONAMs. They automate the task of creating, adapting, configuring, and terminating SSONs. They work directly with the ONAM through their management interfaces. They perform different self-management functions, such as self-configuring, self-optimizing, and self-adapting. Therefore, they have different control loops. Typically, they perform the following tasks:

Self-configuration: SSON-AMs generate configuration policies in response to the received system policies. They use these policies to configure overlay nodes that are participating in a given SSON.

Self-optimization: during SSON construction, SSON-AMs discover the overlay nodes required to set up a routing path for the multimedia session. Therefore, they are responsible for optimizing the service path to meet the required QoS metrics induced from high level policies as well as the context of the service.

Self-Adaptation: SSON-AMs monitor the QoS metrics for the multimedia session and keep adapting the service path to the changing conditions of the network, service, and user preferences. They also monitor the participating overlay nodes and find alternatives in case one of the overlay nodes is not abiding to the required performance metrics.

SSON-AMs receive goal policies from the SAMs to decide the types of actions that should be taken for their managed resources. A SSON-AM can manage one or more overlay nodes directly to achieve its goals. Therefore, the overlay nodes of a given SSON are viewed as its managed resources. In addition, they expose manageability interfaces to other autonomic managers, thus allowing SAMs to interact with them in much the same way that they interact with the ONAMs.

This is illustrated in Fig. 6.3. Where the lower part represents an SSON that consists of a Source (S), a Destination (D), and a MediaPort (MP). The SSON is managed by a SSON-AM. Since the SSON-AM can manage multiple SSONs, it has its own Knowledge Base (KB). It contains also a PG backed up with a CRA. The PG has access to the available context information that assists it in achieving its goals. The upper part represents a SAM and its components. The SAM is able to manage one or more SSON-AMs. Therefore, it has its own KB, and PG. The context information of the user, network, and service is assumed to be available to these autonomic managers as they can acquire it from the Context Functional Area in the Ambient Control Space [46].

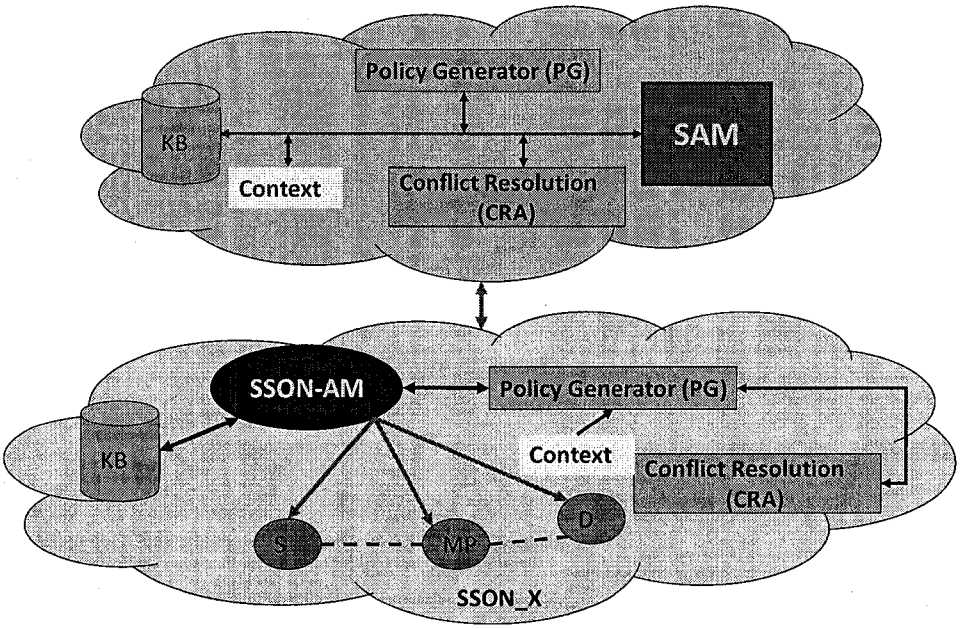

Fig. 6.3 The relation between an SSON, SSON-AM, and SAM

6.2.2.3 System Autonomic Managers (SAM)

A single SSON-AM alone is only able to achieve self-management functions for the SSON that it manages. If a large number of SSONs in a given network with their autonomic managers is considered, it is observable that these SSONs are not really

isolated. On the one hand, each overlay node can be part of many SSONs if it offers more than one service or if it has enough resources to serve more than one session. On the other hand, the SSONs' service paths may overlap, resulting in two or more SSONs sharing the same physical or logical link. For example, consider two SSONs sharing the same routing MP with the same goal to maximize throughput. This will lead to a competition between autonomic managers that are expected to provide the best achievable performance. Therefore, and in order to achieve a system wide autonomic behavior, the SSON-AMs need to coordinate their self-managing functions. Typically this is achieved using SAMs. SAMs can manage one or more SSON-AMs. They pass the system high level policies, such as load balancing policies, to the SSON-AMs. Moreover, whenever they find shared goals between two different SSON-AMs, they inform them to avoid conflicting actions. The involved autonomic managers then contact each other to coordinate their management actions before they are passed to their overlay nodes.

Sharing goals is not the only reason for the coordination step; SSONs sharing common links as well as SSONs that belong to the same policy domain (same service class, ISP, etc.) may also need to coordinate their management actions. Moreover, SSONs that share common nodes/links affect each other's performance, as they compete for the shared resources. This might result in a degraded performance as the competition will cause the control loop to be invoked frequently in an attempt to reach the desired performance goals. Also, all the SSONs in a given domain (ISP) are expected to achieve the domain wide policies together. Coordination allows these policies to be dispatched and adapted to each SSON in a way that achieves the desired goals. Moreover, it also allows the sharing of control and information between different SSONs. A set of SSONs that are co-located in given vicinity (such as an area, domain, AS, etc.) are usually equipped with independent route decisions based on its observations of its environment. Sharing this information will result in a reduced overhead for each overlay to compute this information, and will allow for adapting and generating policies to achieve better performance.

6.3 Distributed Knowledge

Each autonomic manager obtains and generates information. This information is stored in a shared Knowledge Base (KB) (see Fig. 6.3). The shared knowledge contains data such as SSON topology, media type descriptions, the set of policies that are active, and the goal policies received from higher level autonomic managers. The shared knowledge also contains the monitored metrics and their respective values. When coordination is needed, each autonomic manager can obtain two types of information from its peers. The first is related to the coordination actions and the second is related to the common metrics in which each autonomic manager is interested. Therefore, knowledge evolves over time; the autonomic manager's functions add new knowledge as a result of executing their actions, obsolete knowledge is deleted or stored in log files. Also, goal policies are passed from high level autonomic managers to their managed autonomic managers. The context information of the network, users, and services is also used primarily to aid in generating suitable policies at each level of autonomic managers.

6.4 Policies

The use of policies offers an appropriately flexible, portable, and customizable management solution that allows network entities to be configured on the fly. Usually, network administrators define a set of rules to control the behavior of network entities. These rules can be translated into component-specific policies that are stored in a policy repository and can be retrieved and enforced as needed. Policies represent a suitable and efficient means of managing overlays. However, the proposed architecture leverages the management task to the overlays and their logical elements, thus providing the directives on which an autonomic system can rely to meet its requirements. Policies in our autonomic architecture are generated dynamically, thereby achieving an automation level that requires no human interaction. In the following, we will highlight the different

types of policies specific to autonomic overlays. These policy types are generated at different levels of the system.

Configuration policies: These are policies that can be used to specify the configuration of a component or a set of components. The SSON-AMs generate the configuration polices for the service path that meets the SSON's QoS requirements. The ONAMs generate the specific resource configuration policies that, when enforced, achieve the SSON QoS metrics. The user, service, and network context are used by these autonomic managers to generate configuration policies.

Adaptation policies: Thesre policies that can be used to adapt the SSON to changing conditions. They are generated in response to a trigger fired by a change in the user, service, or network context. SSON-AMs receive these triggers either from the SAMs or from the ONAMs, while the ONAMs receive these triggers either from the SSON-AMs or from their internal resources. Whenever a change that violates the installed policies occurs, an adaptation trigger is fired. The autonomic manager that first detects this change tries to solve the problem by generating the suitable adaptation policies; if it does not succeed, it informs the higher level autonomic manager.

Coordination policies: Are policies that can be used to coordinate the actions of two or more SSON-AMs. They are generated by the SAMs to govern the behavior of SSON managers that have conflicting goals to avoid race conditions.

Regulation policies: These are generated by the overlay nodes themselves to control the MAs' behavior with respect to their goals. For example, a MA that measures throughput has a policy to report throughput < 70%. Another regulation policy can be installed to replace this policy and report throughput < 90%. The second regulation policy can be generated in response to an adaptation policy that requires throughput to be at least 90%. The MAs therefore are made more active to contribute to achieving the required tasks.

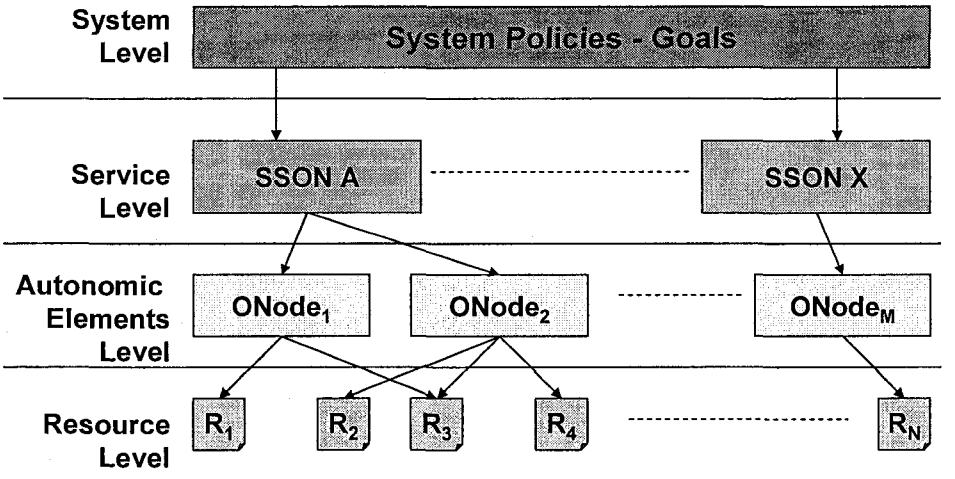

Fig. 6.4 Different Policy Levels

Figure 6.4 shows how these policies are related to our autonomic architecture. At the highest level, the SAMs define the set of system polices. These policies represent the system-wide goals and do not describe either the particular devices that will be used to achieve the system goals, or the specific configurations for these devices. SAMs pass these policies to the SSON-AMs. SSON-AMs refine the system policies and generate service specific policies. They do so by adding further details to the system policies. These details are induced from the system policies as well as from the context information of the users, the network, and the service. At this level, the goals of the SSON under discussion, such as the permitted QoS metrics, are defined. These goals are still device independent policies. The set of service polices is then passed to the ONAMs. These autonomic managers further refine the received policies and generate the overlay node polices and their respective resource specific policies. Overlay node policies represent the goals that this overlay node is expected to achieve, while resource

specific policies represents the actual actions that the resources of the overlay node has to do to achieve the overlay node goals. This separation of policies allows each autonomic element to focus on its goals and how to achieve them using its current resources while contributing at the same time to the overall system performance. By de-coupling the functionality of adapting overlay node resources policies from the task of mapping system objectives and abstract users' requirements, the policy separation offers users and IT professionals the freedom to specify and dynamically change their requirements. The hierarchical policy model is used to facilitate the mapping of higher level system policies into overlay node objectives. Given sets of user, service and network context and constraints, as well as sets of possible actions to be taken, decisions for policy customizations are taken at run time based on values obtained from MAs to best utilize the available overlay node resources.

In addition to generating policies from high level goals, the policy generator located in each autonomic manager serves as a Policy Decision Point (PDP) for the low level autonomic manager. For example, the SSON-AM serves as a PDP for the ONAM. Whenever an ONAM detects that one of the configuration policies has been violated, it tries to solve the problem locally. If it is unable to do so, it consults the SSON-AM to which the overly node is providing a service. The SSON-AM then tries to solve the problem by either relaxing the goals of the services or by finding an alternative overlay node that is able to achieve the SSON's goals. The SSON-AM then informs the ONAM of its decision, and may also consult its designated SAM to acquire decisions on situations that it cannot handle locally. The autonomic manager acting as a PDP decides which policies, if any configuration or adaptation policies have been violated, were most important and what actions to take. It uses information about the installed policies and the current context of the user, network, and service.

6.5 Summary

In this chapter, a novel scheme for SSONs autonomic management has been presented. This work provides a complete integrated architecture for autonomic SSONs

management; it illustrates the benefits of avoiding the complexity of existing service management systems. The road towards fully autonomic system architecture is still long; however, and this chapter presents an autonomic overlay architecture that represents the basic building blocks needed by autonomic overlay systems.

The success of autonomic computing relies on systems' ability to manage themselves, and to react to changing conditions. The proposed layered architecture for autonomic overlay provision enables autonomy and dynamic overlay construction through multi-level policies. The architecture components can self-assemble into an overall autonomic system—flexibility is crucial to the system. Therefore, individual overlay nodes should be able to self-organize to form diverse SSONs. This is possible through the investigation of the different media types and QoS requirements for each media delivery session, which allows for the dynamic self-composition of the fundamental services needed by SSONs. This will lead to the ultimate dynamic self-management, and will require the dynamic assignment of SSON-AMs and SAMs.

Chapter 7

A Self-Organizing Composition towards Autonomic Overlay Networks

As illustrated in the previous chapter, a major challenge for autonomic computing is composing multiple autonomic entities to achieve system-wide goals. In autonomic overlays, the challenge involves composing multiple autonomic overlay nodes to construct SSONs, which achieves the required QoS. This chapter presents a novel self-organizing composition scheme that can compose overlay nodes to realize SSONs using a self-organizing principle. The rest of this chapter is organized as follows: Section 7.1 introduces autonomic composition challenges. Section 7.2 summarizes related work. Section 7.3 introduces design goals, composition model, self-organizing rules, and the composition algorithm. In Section 7.4, we present simulation details and results, and finally, the chapter is concluded with a brief summary.

7.1 Introduction

Recently, considerable research has been exhausted on self-managing systems, including work from IBM's autonomic computing initiative. As illustrated in **Section 3.2.4,** IBM introduced the concept of Autonomic Computing (AC) [93], which allows systems to manage themselves. IBM identified the complexity of current computing systems as a major burden that hinders its growth [90]. AC simplifies and automates many system management tasks that are otherwise traditionally carried out manually. Systems that manage themselves are able to adapt to changes in their environment in accordance with business objectives. The result is a great saving in management costs and in the time of IT professionals. Liberated from manual operations, these professionals can focus on

improving their overall service. A major challenge in their work [90], [92] is to compose multiple autonomic entities to achieve system-wide goals. We faced a similar challenge when we attempted to compose multiple autonomic overlay nodes to construct SSONs.

Service composition has been proposed within service-oriented environments [182]. It allows simple services to be dynamically combined into more complex services. Service composition is usually defined as a directed acyclic graph $G(N, L, W)$, where N is the number of services in G, and L is the set of links in which a link $l(u, v)$ represents a service composition between u and v, with W as its cost. A service path is defined as a path in G that minimizes a cost criterion. In a highly dynamic network such as SMART, MPs' services are dynamic and change over time. The number of services N is therefore not known beforehand. Assuming there is a large number of MPs, the set of links L that represents all the possible service compositions, changes dynamically. It is impractical to rely on a predefined set of services, as we need an accurate view of the network at any time. These problems can be solved using a registrar entity in which all MPs register their services. The service path is then found by searching G for the best path that minimizes the cost criterion (delay, jitter, throughput, etc...). Most service composition schemes use this model, but it has become unsuitable for media delivery because of its poor scalability and reliability. The central entity is a single point of failure; it also consumes bandwidth because each MP has to re-register its services and resources each time a change occurs. This also means that G has to be re-computed, and another search performed for the best service path. Clearly, this solution is not cost efficient, and in a dynamic network where the topology is always changing, a central entity is not reliable as it may unpredictably leave the network.

This chapter builds on the previous one, where we proposed Autonomic Overlays, and developed the service specific autonomic architecture, and focuses on the problem of composing different autonomic elements to achieve system wide goals.

7.2 Related Work

Service Composition is the orchestration of a number of existing services to provide a richer composite service assembled to meet some user requirements. In particular, if no single service can satisfy the functionality required by the user, it should be possible to combine existing services together in order to fulfill the request. This has triggered a considerable number of research efforts on composition. Composition techniques can be classified into static and dynamic. In static composition, available services are combined by adding a central coordinator that is responsible for invoking and combining the single sub-services. This means that the requester should build an abstract process model before the composition starts. The model includes a set of tasks and their data dependency. On the other hand, the dynamic composition composes services on demand, based on requests from users. For instance, by dynamically composing services on demand, services do not need to be configured or deployed in advance. In addition, by composing services based on requests from users, it is possible to customize the services to individual user profiles. The dynamic composition of services requires the placement of services based on their capabilities and the recognition of those services that can be matched to create a composition.

Several dynamic service composition systems have been proposed. In [183], an architecture that obtains intuitively the semantics of the requested service, is proposed. It discovers the components required to compose a service, and composes the requested service based on its semantics and the semantics of the discovered components. Unfortunately, discovery and execution of the service are carried out by a central middleware. SpiderNet [184] is a QoS-aware service composition framework that provides a Bounded Composition Probing (BCP) scheme to achieve QoS-aware service composition. The basic idea of BCP is to examine a small subset of good candidate compositions according to the users' service requirements and current system conditions. The BCP scheme executes a hop-by-hop distributed composition protocol to achieve its goals. However, the user has to be aware of the required services and specifies them before hand.

Work flow systems [185], [186], [187], [188], [189], and [190] require a user to request a service by choosing or creating a service template that describes the structure of the service in a flowchart-like diagram. They compose the requested service through discovering the components necessary to convert the template into an executable workflow. For example, eFlow[185] uses a static workflow generation method. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The graph is created manually but it can be updated dynamically.

Many research efforts tackling service composition problem via AI planning have been reported. Methods in [191], [192], and [193] adapt and extend the Golog language for automatic construction of Web services that are built on top of the situation calculus. The general idea of this method is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition, and inter-operation. Existing systems [194], [195], [196], [197], and [198] require a user to request a service using a logic language. For example, the system described in [198] requires a user to choose a meta-program described in Golog logic programming language. Similarly, in SWORD [194], a service is modeled by its preconditions and post conditions. They are specified in a world model that consists of entities and relationships among entities. A web service is represented in the form of a Horn rule that denotes the post conditions are achieved if the preconditions are true. To create a composite service, the service requester only needs to specify the initial and final states of the composite service, and then the plan generation can be achieved using a rule-based expert system. However, rule-based chaining can sometimes generate "uncertain" results if a precondition cannot uniquely determines a post condition. AI planning systems compose the requested service in a logic language through a form of planning. However, understanding logic programming languages may not be an easy task.

Service composition has been also addressed by systems such as GriPhyN [199], Libra [200], Ninja [201], and CANS [202]. GriPhyN considers compositions as a static graph of services, and assumes prior knowledge of the participating services and their interaction patterns. The Libra framework aims to automate the optimal composition of

services across the wide-area network using service-specific knowledge. Ninja is a path-based approach that allows services to be automatically discovered and composed into a path. CANS uses type-based specification of components and network resources to enable service access paths to be dynamically and automatically constructed. A mechanism that constructs all possible compositions based on their semantic and syntactic descriptions was proposed in [182]; in this approach, all available services are grouped into directories. The approaches in [203], [204], and many others, mandate the service requests to describe the structure of the service. The composition is carried out by discovering the necessary components. Projects [205], [206] compose services if their basic components are present in the network. If one component is missing, an extended discovery stage is required. Such and other research projects (e.g., [207], [208], and [209]) attempt to generate a global system configuration, under specific optimization criteria. Most see composition as a discovery problem, but they either rely on a centralized composition entity—which has scalability limitation—to carry out the discovery, integration and composition of services, or they assume a prior knowledge of a service graph that defines the basis for their composition algorithms. Moreover, previous work only supports linear composition topology and fixed composition order, which greatly limits the applicability and efficiency of service composition. Our work addresses these limitations and proposes the composition of autonomic elements in which each autonomic element is self-managed.

7.3 Self-Organizing Composition

Autonomic elements inherently guarantee self-management functions for their own resources. However, SSONs are made up of many autonomic elements. The need is therefore to develop tools and environments that facilitate the automated composition of elements into more complex services. Our scheme is based on self-organizing principles found in many biological systems [210]. The requirements for SSON composition are listed below.

- *Decentralized*: A mechanism is needed to dynamically and automatically select different overlay nodes first to construct an SSON and then be manageable by SSON managers. This achieves automatic operation and avoids a single point of failure.
- *Efficient*: The overlay nodes should be selected with minimal disruption to existing services. The use of extensive real-time communications should be limited, while maintaining the QoS requirements.
- *Robust*: Automatic and self-organizing selection and reselection of overlay nodes is needed in order to avoid both the failure of nodes (due to the network's high mobility and heterogeneity) and a single point of failure.
- *Dynamic*: No set of autonomic managers in the network should be permanently responsible for a particular management task. Tasks should be automatically transferable so that loads are balanced and scalable.
- *Distributed and self-organized*: The selection of the overlay nodes required to construct an SSON should be distributed in order to minimize overhead on the node responsible for management. The selection of the set of nodes with which SSONs are constructed should use local knowledge only.

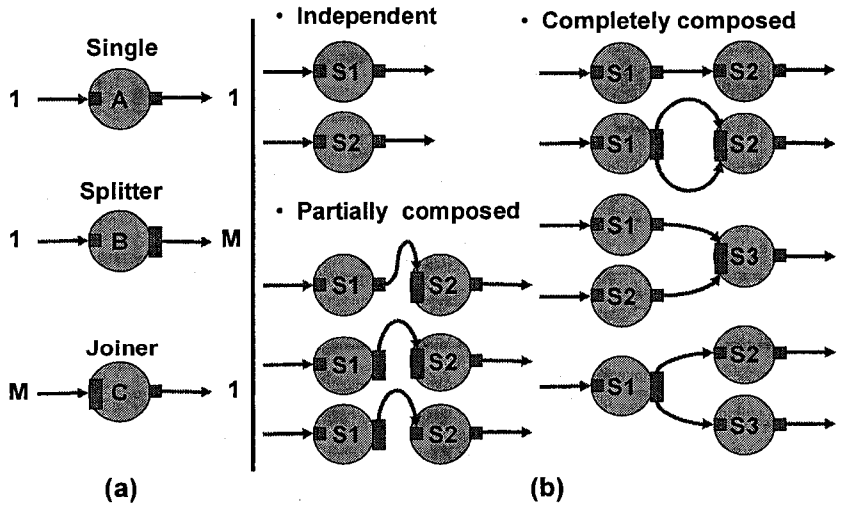

Fig. 7.1 Types of MPs Services Composition

7.3.1 Composition Model

Our work does not assume a specific service description; services can be described using standard Web Service Description Language (WSDL) [211], for example, and extended with semantic metadata. For clarity, we use the same MPs modeling presented in Section 5.3. To summarize, a service S can be described using service identification ID, an input I, an output O, and the function f that the service provides. Using this simple representation, a service S always receives I and produces O as a result of applying f on I. Each service used incurs a cost and each MP provides one or more services. Given an input media I and a requested output media O, the problem is to find a service path that transforms I into O and minimizes or maximizes a cost criterion.

As shown in Fig. 7.1a, MPs can be described according to their input and output ports: Single, splitters, or joiners [212]. Single MPs have only one input port and one output port. Splitters have one input and several outputs. Joiners have several inputs that they merge into one output. Services can therefore be independent, or partially or completely composed. As shown in Fig. 7.1b, independent MPs can perform a service without help from other MPs. Partially composed MPs are those that need other MPs to provide a complete service. Completely composed MPs are those that provide a complete service.

7.3.2 Definition of the Problem

In a dynamic network, each of a set of MPs may offer one or more services. Each MP has knowledge of only its services and of those offered in its vicinity. A Media Client (MC) requests media from a Media Server (MS). Media are characterized by the input (I) that represents the type of media that the MS has, the output (O) that the client can accept, and the required QoS. The composition problem is to determine the media flow that transforms I into O.

Knowledge is defined by the vicinity of each MP. The vicinity can be as small as the direct neighbors of a MP or as large as the whole network. The smaller the vicinity, the more local the knowledge and the lower the cost needed to acquire it. Operating at either end of this range is impractical: Knowledge of the whole network poses similar problems to using a central entity, and the direct neighbors of a given MP might not be MPs themselves. We therefore define the vicinity as the set of MPs in a sub-area of the network. Each sub-area must be large enough to contain multiple media ports and small enough to minimize the cost of acquiring the local knowledge. Generally, a service path should meet the required QoS, that is, it should minimize or maximize a cost criterion. We do not impose strict parameters on these criteria because they are application and user-dependent. For example, one user might be interested in maximizing the throughput, while another user might wish to minimize delay. Moreover, only the input and the output of each requested media flow are known beforehand, while the service path composition and the order of service are determined dynamically.

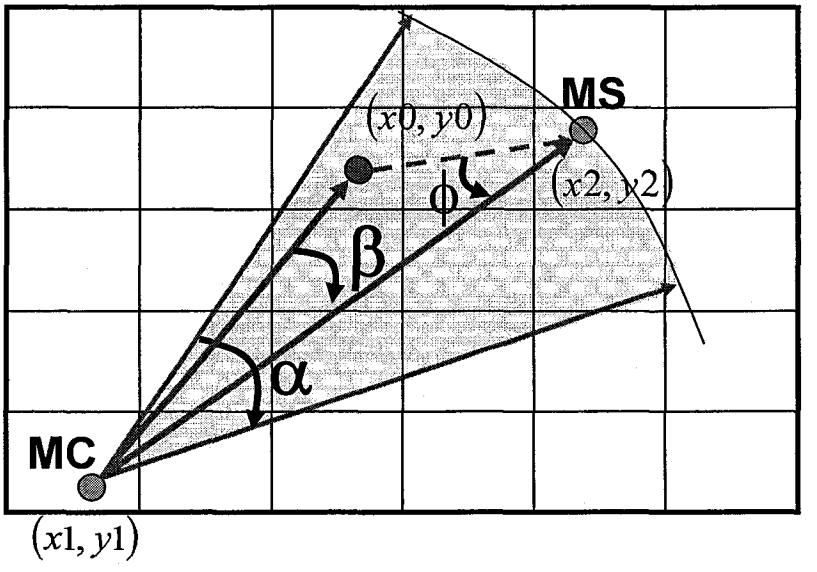

Fig. 7.2 Network Geographical Area and Search Scope Angle

7.3.3 Self-composing Assumptions and Rules

We assume that each node knows its geographical location, and that the network geographical area is two-dimensional. This can be obtained by mapping IP to geographical locations [213]. We also assume that the area is further divided into sub-areas of equal size as shown in Fig. 7.2. It is the same as Fig. 5.5. We repeat it here for clarity. Note that although the computation is similar, it is being used differently in this chapter. In this chapter we don't maintain any form or structure between nodes. Therefore, the only needed knowledge is the geographical location. While in Chapter 5, the resource discovery, we form an optimal chordal ring between nodes and the computed nodes inside a given search angle are the nodes on that ring. Moreover, routing is based on the connections of the optimal chordal ring. In contrast, routing here is based on flooding and the search angle limits the propagation of the flooded request to those nodes inside the angle scope. The sub-areas are fixed for sufficient time to allow each node to become aware of them. This knowledge can be broadcast once to all nodes in the network; new nodes acquire the knowledge from neighboring nodes. As stated previously, each MP knows its own services and those offered by MPs in its sub-area. This knowledge is also obtained by broadcasting it in each sub-area. Although this step is a clear overhead, it is required only once by each MP. Initially, the service request is forwarded based on local knowledge only, that is, only to nodes that can provide a value for the service request and for its output service. We also forward the service request only to MPs in the direct path between the MC and the MS (the shaded area shown in Fig. 2). This is because service composition is useful only when it can bring the media flow closer to its goal. Looping of a service path is undesirable because of its clear performance problem, and because closer nodes can be expected to require fewer hops than those further away. Finally, each MP has a distance function that is used to produce the list of required adaptations for a media flow based on its input and output. Using these assumptions, a MediaPort A has a list of other MPs that provide services (called the ActiveList). A may be able to compose with all of these services, with some of them, or with none at all. The composition may be partial or complete. Whenever A receives a composition task during its lifetime, if it is unable to execute the task

independently, it forwards the composition task to the highest ranked MP that it can compose with in its ActiveList. The rank between MPs A and B is $R(A,B)$ and is calculated using the following rules:

- $R(A,B) = 4$, If A knows that B can compose with A to provide a complete service
- $R(A,B) = 3$, *If A* knows *that B can compose with A to provide a partial service*
- $R(A,B) = 2$, *If A knows that B can provide a complete service by itself without composing with A*
- $R(A,B) = 1$, *If A knows that B can provide a partial service by itself without composing with A*
- $R(A,B) = 0$, *If A knows that B can't provide a service at all*

These ranks can be viewed as virtual links with a strength value. A rank of 0 means that the link has no chance to be selected in the composition process as it provides no useful service. However, the link is maintained, as it might be needed to forward the request if the current node is not aware of any other node that it can compose with. The higher the strength, the more chances the MP has to be selected. The ranks represent the initial view of MP A to its vicinity. The ActiveList becomes dynamic, as the ranks based on the actual selection of nodes are modified and as links are added or removed. Since the network is dynamic, MPs may leave or join it. Whenever a new MP joins the network, it broadcasts its availability to its vicinity allowing other nodes to update their ActiveLists and whenever a MP discovers that one of its ActiveList members is unavailable, it removes it from the list. By the use of learning rules, new members from neighboring sub-areas can be added to the ActiveList as described later.

1. If *A* is within the search scope, it Computes the list of adaptations A[i] that are needed to transform I into O,
2. Extracts the elements in the composition history and adds to the list of adaptations A[i] all the available partial adaptations.
3. For each adaptation in A[i]

   3.a If *A* can provide a complete service it will add to the composition history its ID, the service cost, and the available information that is relevant to the QoS metrics (ex. The delay between itself and the node that it composes with)

   3.b If *A* can provide a partial service then it will add to the composition history the above information in addition to the output partial service.

   3.c If *A* can provide both complete and partial adaptation it will add both to the composition history.

   3.d If *A* doesn't provide any type of adaptation, it adds nothing.
4. Forwarding the service request: *A* checks its ActiveList to decide where to send the service request. Generally, *A* prefers MPs with highest ranks. To further limit the forwarding and to reduce the cost of sending messages it do the following:

   4.a If *A* knows a MP that it can provide a complete service it forward the message to it.

   4.b If *A* knows a MP that it can compose with, it forward the message to it.

   4.c If the activeList has all ranks zero, then *A* has no clue about where it should forward the message. Therefore, *A* sends it to all of its ActiveList Members.
5. Any node that receives a forwarded message deals with it in the same way as *A* did. The only exception is for the receiver node to be the MS, in which case, the MS waits for a period of time T, and retrieves the composition history from each received message. It then selects a path that meets the cost metric for the media flow, and sends a ConstructPath message through the path to the MC in a reverse order. If no path is retrieved, the MS sends a failure message to the MC that will resubmitted composition request after increasing α.

Fig. 7.3 Composition Algorithm

7.3.4 Self-organizing Composing Algorithm

A media flow is constructed when a MC requests a service composition task. The MC broadcasts a composition request to its vicinity containing the request ID (RID), the MediaClient Input (I), the MediaServer Output (O), the QoS requirements, and the angle α [0,180] that determines the search scope between the MC and the MS (see Fig. 7.2). The larger α is, the more sub-areas are included in the search. The message contains the task information as well as the composition history accumulated as the message is sent through the network. To prevent the uncontrolled forwarding of a service request, each node keeps a record of received requests and compares any new request with this record. If the request has already been dealt with, it is discarded. MP A processes a request as shown in Fig. 7.3.

To know that it is within the search scope, A computes the angles β and ϕ using the following formula:

$$\beta = \sin^{-1}\left(\frac{|(x2-x1)(y1-y0)-(x1-x0)(y2-y1)|}{\sqrt{(x2-x1)^2+(y2-y1)^2}*\sqrt{(x1-x0)^2+(y1-y0)^2}}\right) \tag{7.1}$$

Where $(x1, y1)$ is the MC location, $(x2, y2)$ is the MS location, $(x0, y0)$ is the location of A.

$$A \text{ is within the scope area if } \beta < \alpha/2 \wedge \phi < 85^{O}$$

If A is not within the search scope, it discards the message.

This composition algorithm relies on the message being broadcasted from the MC to MPs in its vicinity and from any MP with an ActiveList of zero to its vicinity. But the clear overhead of broadcasting renders this undesirable. We therefore combine the algorithm with learning rules induced from biological systems [210]. These rules are:

1. *Learning from interaction*: Since the service request is sent to all sub-areas in the search angle α, an MP in one sub-area may learn about MPs that it can compose with in other sub-areas. Step 3.b of the algorithm shows that this knowledge can be

acquired with no extra cost. Because each MP adds its own adaptations to the adaptation list A[i], any MP can decide which of the MPs that it can compose with are not listed in its ActiveList. Those MPs can be added to the ActiveList for future use and the same rules can be used to rank them. In the first few requests, this rule does not reduce the message cost, and the algorithm still uses the broadcasting option. As the number of requests increases, so does the number of useful links added. The message overhead is thereby reduced. Typically, each MP has limited space in which to store information about its neighboring MPs. Adding more MPs fills the available space quickly. We therefore use a replacement strategy whereby any new MP is added to the ActiveList if the ActiveList is not full. If it is full, the MP with the lowest rank is replaced by the new one.

2. *Positive and negative feedback*: in step 4 of the algorithm, a MP forwards the service request to nodes in its ActiveList. Although our filtering rules reduce the number of nodes to which the message is forwarded, the number of candidate nodes could still be high and the resulting overhead is undesirable. Nor are all candidate nodes really needed, as some may not be willing to participate in the new media flow (either because they do not have enough resources or because they have their own policies that do not allow them to participate). In fact, a dynamic network implies that not all nodes are available all the time. We therefore extend our algorithm to include *positive feedback*: We increase the rank of links to nodes that are known to be cooperative and have actually participated in a media flow. This is simple knowledge to acquire, and comes with no cost. In step 5 of the algorithm, each node receiving a ConstructPath message executes an UpdateRanks function that increases the rank of nodes that a MP is composing with. Conversely, if a MP is known to fail frequently or not to have participated in a media flow for some time, its rank is decreased. This positive and negative feedback reduces the number of nodes that receive composition requests. It also becomes much more probable that messages are received by a MP that has more opportunity to participate usefully and cooperatively in the media flow.

3. *Orientation-based modulation:* from Fig. 7.2, it can be seen that the composition request moves to its destination in a specific direction. A node therefore selects nodes from its ActiveList that reflect that direction. This ensures that each receiving node is closer to the destination than the transmitting node and prevents the message from going into loops.

7.3.5 Discussion

After waiting for a period of time T, the MS may receive several media flow paths, both valid and incomplete. Using an optimization criterion, the MS computes the cost of each complete flow, and selects the one that meets the required cost. The media flow paths, either complete or partial, are of great importance for autonomic systems. Since the network is dynamic, an established path may not be available for the duration of the session. Participating nodes may run out of resources; they may also fail or leave the network. It is therefore essential for autonomic systems to be able to recover from these and similar situations. The MS keeps a record of all the possible media flow paths returned when the algorithm is executed. Once it receives a leave notification from a current path member, it looks for an alternative node as a replacement or for an alternative path from those already available.

In the previous chapter, an SSON autonomic manager is responsible for self-configuring and self-optimizing the SSON overlay path or media flow. Our composition algorithm assumes that the MS plays this role. However, if a different node claims the SSON autonomic manager, it receives the media flow paths. To obtain the best performance, the SSON autonomic manager should be close to the media flow path. And since the flow path is not known beforehand, the SSON autonomic manager should be located in one of the sub-areas between the MC and MS.

Although alternative media flow paths are important, accounting for all possible alternatives and partial solutions increases the size and overhead cost of the request message. The following rule can therefore be added to the algorithm at steps 3.a and 3.b

IF A can provide a complete OR partial service (S)

IF S ~∃ in the composition history OR (S ∃ and the number of similar services < δ)

Add the service to the composition history

This rule limits the number of similar services in a path history to a predefined threshold value (δ). The value of δ depends on how dynamic the network is and how much bandwidth is available. In a highly dynamic network, nodes leave and join frequently. We need more alternatives in order to avoid service breakdown. We therefore set δ to a greater value. Setting δ to a lower value reduces the number of alternatives, but does not eliminate them completely. For example, if δ=0, there are no alternative solutions in the current message. But the algorithm allows a number of messages with the same service request to be forwarded along different paths.

In the proposed composition algorithm, QoS has been generically addressed through using a cost metric, where its value decides if a node will be selected in a final media flow path or not. While this allows for a wider use of QoS parameters, it doesn't explicitly address the intelligibility of the media flows after being processed by the composed MPs [214] and leaves this topic as a future work.

7.4 Experimental Evaluation

We used a discrete event simulator to evaluate the performance and efficiency of the algorithm. A large-scale network was used to test measurements such as network load, composition time, stretch, and success rate. We first compared the self-organizing algorithm (Self-Org) with limited-flooding (LF) and Graph Based (GB) approaches. In a LF protocol, a composition request is broadcast to all direct neighbors. Close neighbors send it on to their neighbors with the propagation controlled by a TTL value. In a GB approach [182], all services register with a central directory. The service advertisement contains a graph that represents the service to be registered, and the directory maintains a global graph of all registered services. Composition requests are then sent to the

directory. In the interests of a more realistic comparison with the GB approach, we considered a service model that transforms one alphabet into another [182] (for example, a service that accepts a as input and transforms it to b, $a \rightarrow c$,..., $b \rightarrow a$,...*etc.*). This results in a total of 625 different services. We then examined the effect of learning rules on the same measurements by simulating the self-organizing algorithm enhanced with the rules we developed (Self-Org+).

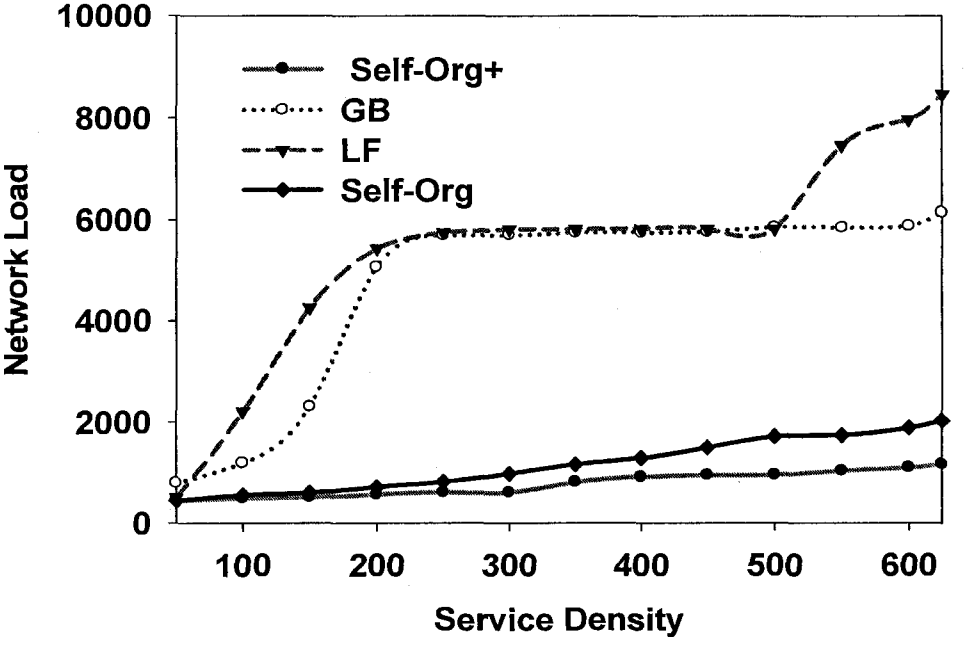

Fig. 7.4 Network Load

7.4.1 Simulation Setup

The simulation topology was constructed using the BRITE [169] topology generator. The topology had 2000 nodes in a 1000×1000 node two-dimensional overlay space; bandwidth assigned to each node was randomly selected between 128 and 512 kbits/s; links propagation delay was fixed at 1 ms; each node had a random geographical location. To simulate a flash crowd, all nodes issued their composition requests at a random point during the first 15 seconds, with the simulation lasting for another 10000

seconds. We ran the simulation 13 times with varying service densities, and varying values for the search angle α , and for *search scope*. (This value is similar to TTL except that it measures how far the composition request travels in the network in terms of network distance. This is a relatively stable characteristic.) For each run, a random number of compositions (between 1800 and 2000) was requested. The results were collected and averaged after each run. In the GB approach, updates are triggered every minute. In Self-Org and Self-Org+, δ is set to 2, the ActiveList size is 15, and each sub-area is 40×40.

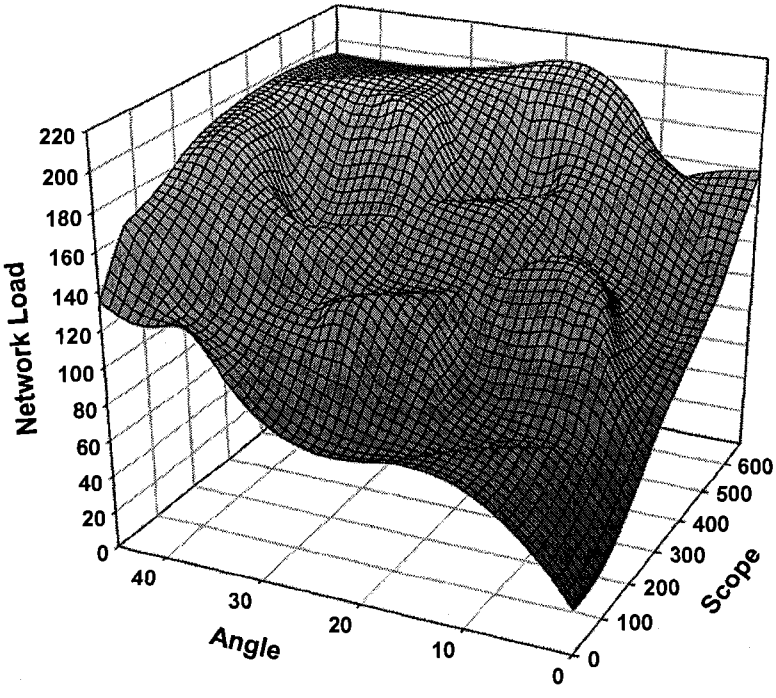

Fig. 7.5 Self-Org+ Network Load as a Function of Scope and Search Angle

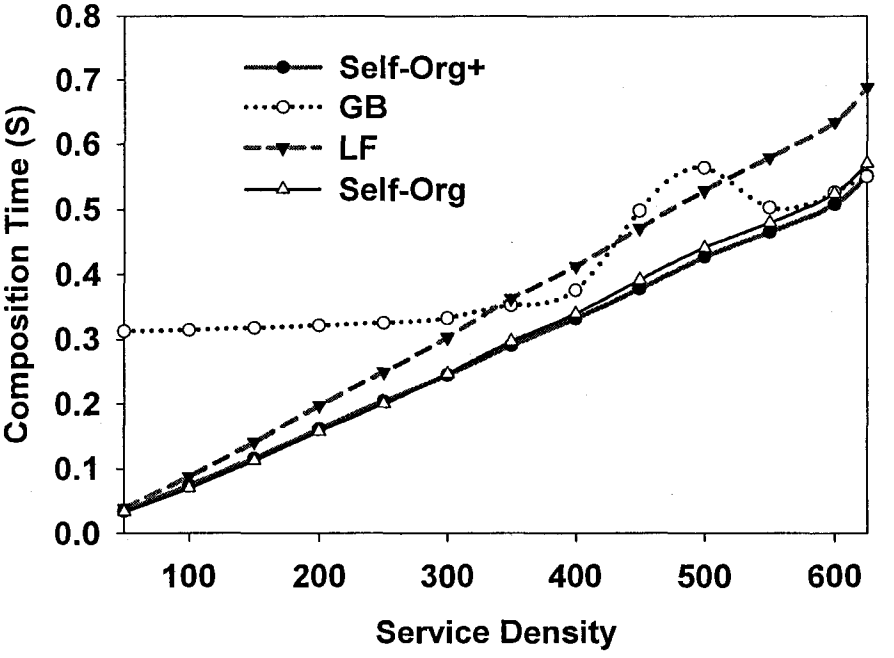

Fig. 7.6 Composition Time

7.4.2 Network Load

Network Load quantifies the cost of using the composition scheme. It represents the total number of generated messages (the total number of hops taken by all composition requests divided by the number of requests).

Fig. 7.4 shows that LF has the worst performance, as it produces a greater number of messages; the GB approach performs better than LF with small service densities. This is because the network load in GB is determined by the number of services because each service produces many service advertisements, while in LF, the network load is determined by the TTL value. The figure shows that the Self-Org approach has a lower load than LF and GB, and the Self-Org+ algorithm has the lowest network load. This is because the Self-Org+ load is determined not only by the number of services but also by the search scope angle α. Fig. 7.5 shows a 3D mesh for Self-Org+ with α and scope varying simultaneously. The figure shows that increasing α and the scope increases the

network load. However, after $\alpha = 40$ and scope = 550, the increase is only slight. This means that the network is stabilizing due to the learning rules and that all composition requests are being served with a bounded number of messages.

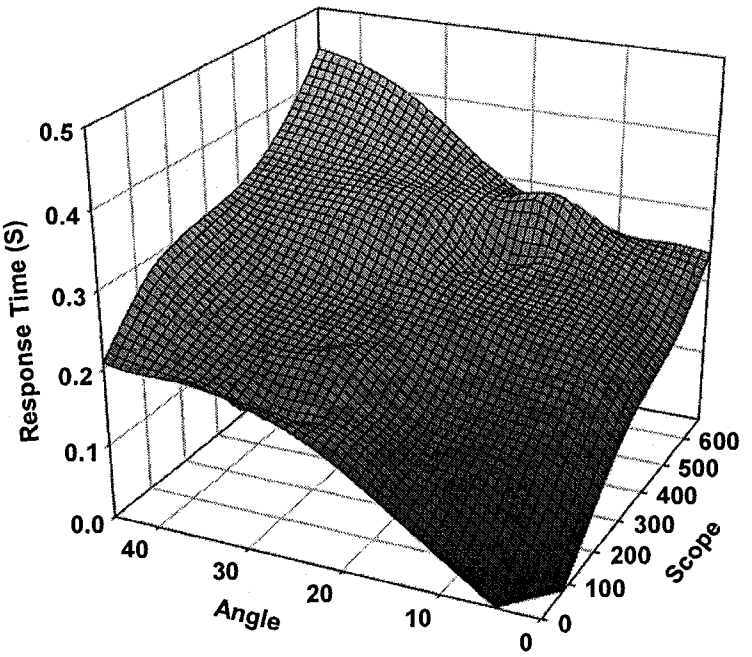

Fig. 7.7 Self-Org+ Composition Time as a Function of Search Angle and the Scope

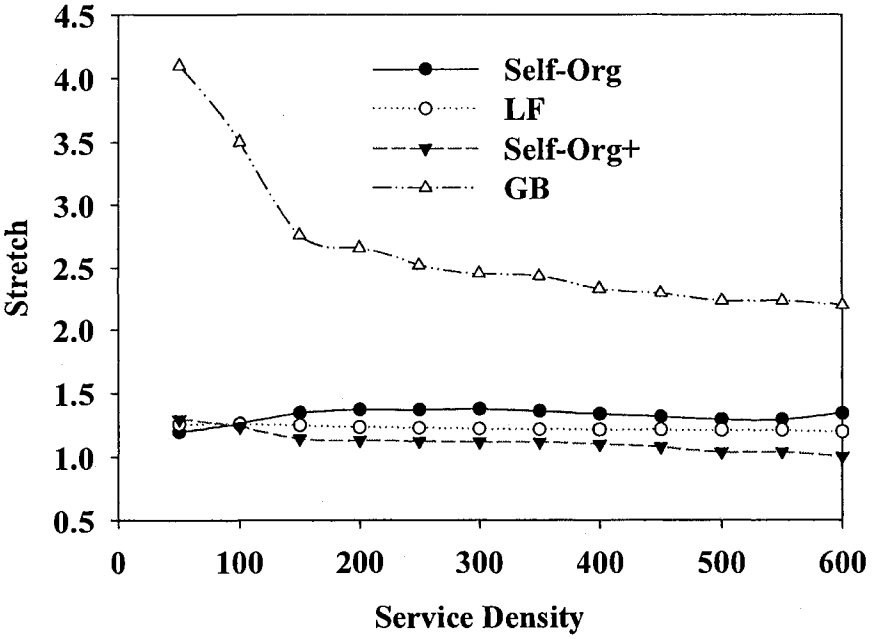

Fig. 7.8 SSON Overlay Path Stretch

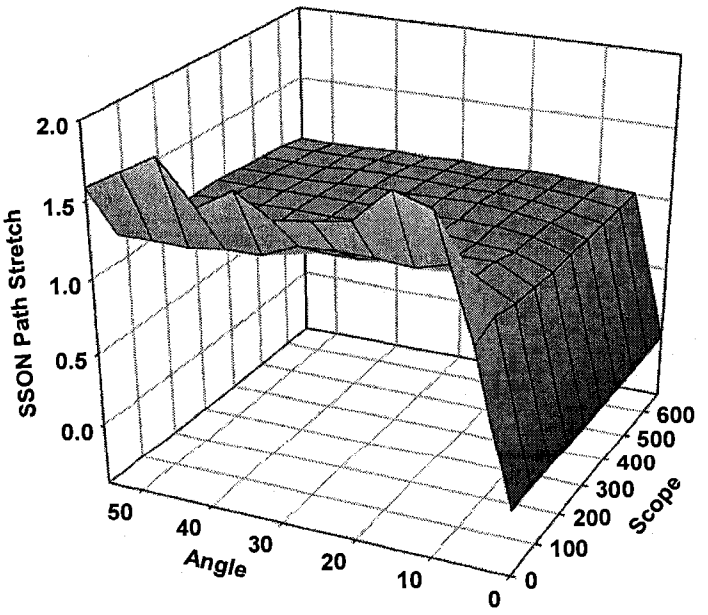

Fig. 7.9 Self-Org+ Overlay Path Stretch as a Function of the Search Angle and the Scope

7.4.3 Average Composition Time

The composition time is the difference between the starting time of the composition request and the arrival of the complete results. Fig. 7.6 shows that the average composition time of the GB approach is at least three times higher than of the LF approach when the number of services is small. This is because composition time in GB greatly depends on the number of requests. The average composition time observed in Self-Org+ is slightly lower than Self-Org for a large number of services and slightly higher for a small number of services. We believe that this increase is primarily due to the reduced amount of learning. Fig. 7.7 shows the 3D mesh for Self-Org+ when both the scope and the angle α are varied.

7.4.4 Packet stretch

Stretch is defined as the number of hops taken by an overlay packet divided by the number of hops the packet takes when using an IP-layer path between the same source and destination. A high stretch value indicates an inefficient SSON topology as longer routes delay the packets. Fig. 7.8 shows the simulation results for the average stretch, and Fig. 7.9 shows the 3D mesh for Self-Org+ when both the scope and the angle α are varied. GB displays the worst stretch, especially for a low number of services. When the number of services increases, the stretch decreases. The results show that the stretch for the Self-Org+ approach ranges from 1.01–1.2 for large search scope values, and from 1.01-1.8 for smaller values. The results also show that the angle α has little effect on the stretch. This is because the path that best minimizes the stretch lies directly between the source and the destination; increasing α will therefore not affect the stretch. Generally, the stretch in Self-Org+ is not significant, considering the gains in other measurements.

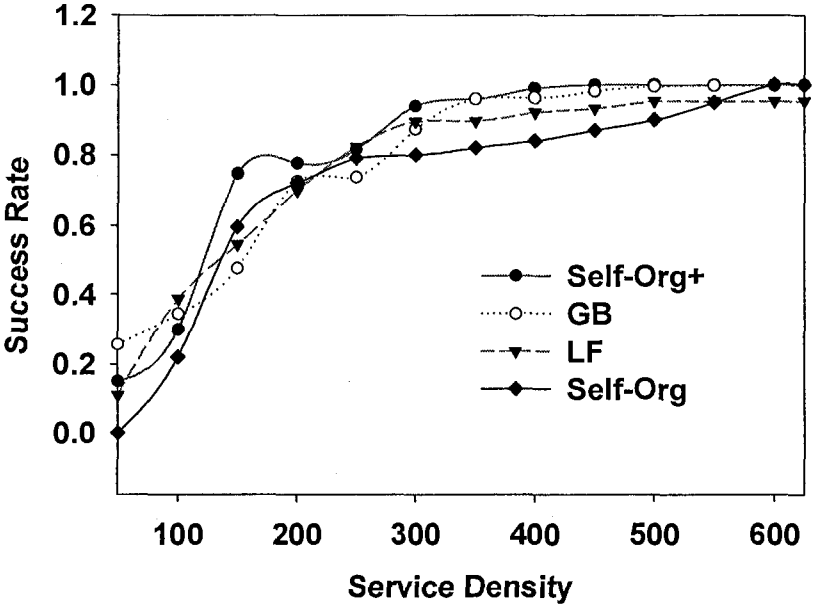

Fig. 7.10 Service Composition Success Rate

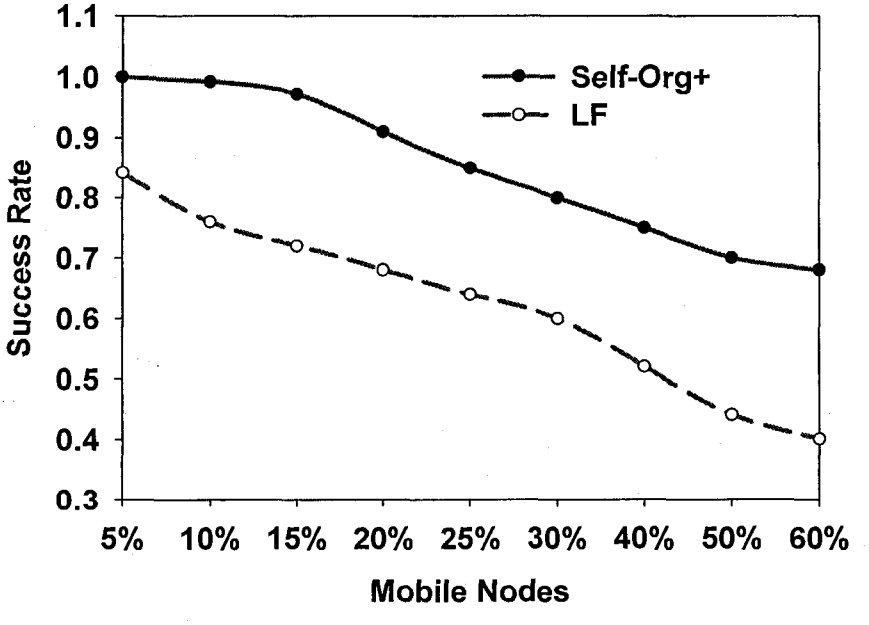

Fig. 7.11 Service Composition Success Rate with Mobility

7.4.5 Success Rate

Success rate is defined as the number of requests that receive positive responses, divided by the total number of queries. Fig. 7.10 shows that Self-Org+ results in a higher success rate, except when the number of services is relatively small. In that case, LF is more effective, though the success rate is still less than 100%. GB is also more effective than Self-Org+ for a small number of services and attains a 100% success rate after a certain number of services. However, Self-Org+ reaches the 100% success rate earlier.

Mobility is an important challenge in a dynamic network. The MC (or user) might move to another location and the MP, which is providing the service, might be mobile or become unavailable due to power limitation. Therefore, we measured the success for Self-Org+ compared to LM under mobility situations. The topology parameters in this experiment are the same as in section 7.4.1. The only exception is that each node is equipped with a wireless interface. The MAC layer is using the IEEE 801.11 protocol and the mobility model for each node is a Random Waypoint. Each mobile node moves with at a speed of 6 meters/second. Service density and the search angle α are fixed at 500 and 35 respectively.

Mobility of nodes affects the multimedia sessions in progress as well as those sessions that are being composed. Therefore, when a mobile node moves, an alternative node must replace it immediately in order to reduce service disruption. Fortunately, increasing the value of δ in Self-Org+ algorithm can be of a great help in this situation. To this end we set δ to be proportional to the number of mobile nodes. We ran the simulation multiple times with varying the number of mobile nodes. Fig. 7.11 shows the success rate of the mobility experiment. We observed that Self-Org+ outperforms LF and attains the 100% success rate with mobility less than 10%. Increasing the number of mobile nodes decreases the success rate in both Self-Org+ and LF. For 50% mobility (that is

1000 nodes in our simulated topology), Self-Org+ attains 70% success rate while LF attains only 44%.

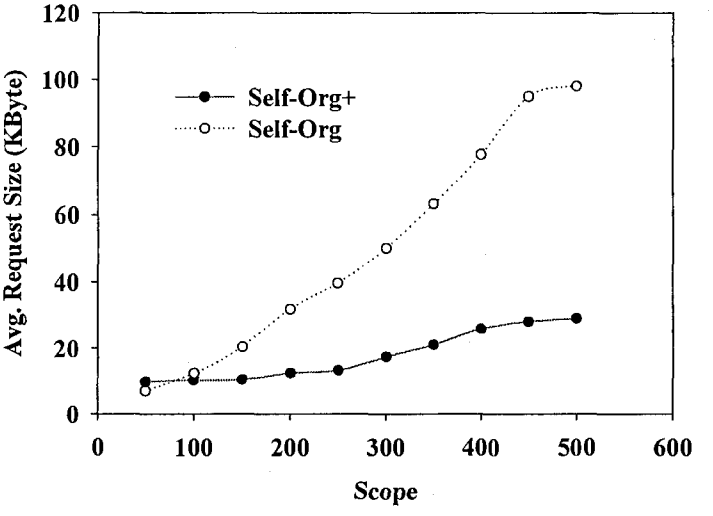

Fig. 7.12 Average Request Size

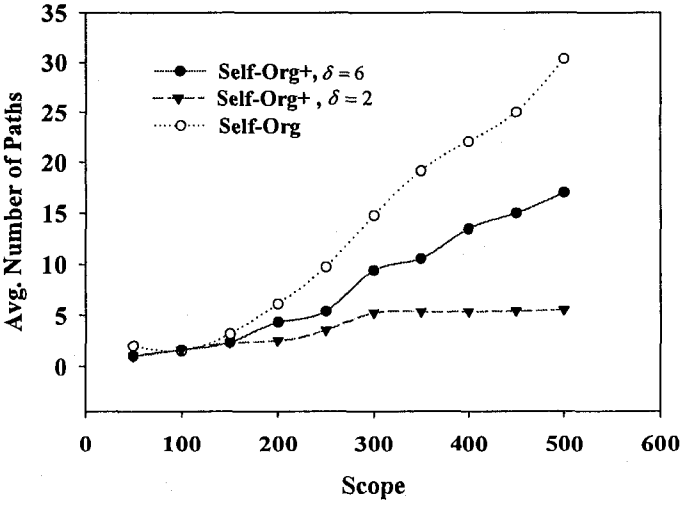

Fig. 7.13 Average Number of Paths Returned at the MS

7.4.6 Additional results

As illustrated, the composition request accumulates the possible compositions while the request flows from the source to the destination. This indicates that both composition request size and the number of returned paths are vital to the success of the algorithm. First, a very large request size is not desirable because it consumes bandwidth. Second, a large number of paths is not desirable also because it generates so many messages. Fig 7.12 shows the measured average request size and Fig. 7.13 shows the average number of paths. For these figures, the search angle was fixed at 35. We observed that the average request size is greatly reduced by Self-Org+. It increases with the search scope, but varies between 10 and 30 Kbytes, over 50% less than Self-Org. We also observed that the average number of paths is much lower when we decrease δ from 6 to 2. It varies from 2 to 5, which results in a great saving for the bandwidth.

7.5 Summary

In this chapter, a novel scheme for SSONs self-organizing composition has been presented, in which, autonomic elements can organize themselves into SSONs using a self-organizing algorithm. The algorithm is powered by learning rules derived from biological systems, and composition requests are forwarded based on the knowledge acquired from previous requests. The scheme also accounts for alternative media flow paths, as well as for partial media flow paths, and provides rules to control the growth of possible solutions to an acceptable level. It was shown that the proposed scheme is efficient in composition cost, accuracy, and composition time.

Chapter 8

Conclusions and Future Work

This chapter identifies contributed research work and discusses planned and future directions; it is organized into two sections: Section 8.1 gives a summary of research contributions in the area of autonomous SSONs management. Section 8.2 sheds light on future research directions.

8.1 Dissertation Contributions

The focus of the conducted research has been the development of an autonomous management system for SSONs. The first step towards achieving that goal demanded a literature study of the autonomous management problem and management difficulties. More precisely, we aimed to address two questions: 1) What are the requirements of an autonomous management system? 2) Why is it difficult to satisfy these requirements with the current approaches? Answering these questions materialized into a state-of-the-art survey of major research directions and efforts in the areas of autonomous overlay networks management and resource discovery schemes. Based on the identified limitations of current research work, a novel framework for an automated SSONs management system has been designed. The framework has been presented as a multi-layered model that utilizes context of users, applications, and the underlying network to perform autonomous management functionalities. The main contributions of the current research work can be summarized as follows:

1. A state-of-the-art survey of management approaches.
2. A complete design and functional specification of an autonomous SSON management framework. The framework makes use of the available context

information such as user, network, and service provider context information to automate the creation, adaptation, and termination of SSONs. The performance of the proposed scheme has been evaluated through simulation.

3. A novel scheme for a semantic MPs resource discovery has been presented, and is based on a widely studied family of chordal rings called the optimal chordal ring. The semantics of MPs, as well as their geographical locations, were used to achieve the highest possible performance. In contrast to existing approaches, the proposed approach requires the lowest number of states maintained at each node, and produces an acceptable message overhead. Simulation results have merited the efficiency of the proposed scheme.

4. Due to the increased management complexity, a novel, autonomic overlays architecture for SSONs management has been presented; SSONs and their constituent overlay nodes are made autonomic, and thus become able to self-manage. Autonomic entities are driven by policies that are generated dynamically from the context information of the user, network, and service providers. This ensures that the creation, optimization, adaptation, and termination of overlays are controlled by policies, and thus the behaviors of the overlays are tailored to their specific needs.

5. To tackle a major challenge in autonomic computing, a Self-organized composition for autonomic entities has been presented. Overlay nodes are composed of SSONs using a self-organizing algorithm to achieve system-wide goals. The algorithm is powered by learning rules induced from biological systems, and endowed with filtering rules to achieve the highest possible performance. The performance of the proposed composition scheme has been evaluated by simulation.

8.2 Future Research Work

The main focus of our future research work can be divided into two key directions as follows: Semantic QoS Composition and Case-Based & Reinforcement Learning Adaptive Management.

8.2.1 Semantic QoS Composition

A subject for future work is the intelligibility of media flows after being processed by the MPs along the composed path. By incorporating the semantics of offered services and the QoS requirements into MPs composition, one can further enhance the quality and performance of an autonomous management system. One way to achieve that is through the utilization of the technical quality and semantics of the media content. Consider, for example, a media content that has been converted from DivX into RM and finally into MPEG. The quality of MPEG media can be very low compared to the original media encoding quality. Therefore, it is essential to consider not only the required conversions for the content but also the quality of the complete chains of concatenated conversions. A model is thus needed to evaluate the assumed outcome of such media conversions, and to automatically propose the most suitable way of conversion and delivery, which might require the adaptation of methods from the area of automatic decision making, and also algorithms for selecting and configuring the composition path.

8.2.2 Case-Based & Reinforcement Learning Adaptive Management

By incorporating the experience gained from applying different management strategies, one can further enhance the performance of autonomous management systems by (among other methods) the utilization of Case-Based Reasoning (CBR), as well as Reinforcement Learning (RF) concepts. CBR is a problem solving and learning paradigm that has received considerable attention over the last few years [215]. Reinforcement learning (RL) is a promising new approach for automatically developing

effective policies for self-* management Systems [216]. RL has the potential to achieve superior performance to traditional methods while demanding less built-in domain knowledge.

On one hand, an agent in RF learns effective decision-making policies through an online trial-and-error process, which works by observing the environment's current state, performing some legal action, and receiving a reward (a numerical value that the user would like to maximize) followed by an observed transition to a new state. RL might need to observe a huge number of (state, action) pairs and state transitions to converge to optimal policies. This prohibits an online training approach (due to initial poor policies), and is not suitable for highly dynamic environment. On the other hand, CBR suffers from scalability of Case Memory because it requires huge number of cases. We plan to investigate the feasibility of representing policies as cases, and to learn new policies using RF. We also plan to address the difficult challenge of coordinating decisions between different, and possibly conflicting, Autonomic Managers. To resolve this challenge, we will need to better characterize their actions. We will also need to develop a technique to coordinate the various evaluation metrics, and to determine a coordination policy to ensure coherent action among them.

8.3 Scalability of Proposed Autonomous Management Framework

Scalability in our proposed autonomous management framework can be viewed from different perspectives. On one hand scalability might refer to the number of messages generated by our resource discovery protocol, the number of policy objects exchanged between the OPEP and the OPDP, and the number of messages generated by our self-organization composition scheme. On the other hand scalability might refer to the maximum number of concurrent SSONs that could coexist in the same network while fixing its resources.

Although we don't provide a mathematical model that proves the scalability of our resource discovery protocol we showed in Section 5.9.4 that the proposed protocol is

scalable because it generates less number of messages compared to two well known scalable protocols. These are Chord and CAN. This is evident in Fig. 5.19 where we compared the total message overhead of our resource discovery protocol to that of Chord and CAN. The total overhead is computed as the number of messages needed for the lookup phase and the maintenance messages generated by each protocol to preserve its structure integrity.

The number of policy objects generated by our autonomous management framework is another factor that affects its scalability. Usually these policy objects are generated due to a change in the network environment. For example, when the resources being monitored by the OPEP are fallen below a threshold, the OPEP constructs a policy object that reflects this change and sends it to the OPDP. The OPDP in turn construct a decision stating how to react to the noticed change and sent back to the OPEP to be enforced. Since SSONs usually consists of a limited and small number of overlay nodes, the number of such policy objects is expected to be very small. Fig. 4.10 shows the average management overhead of using our proposed policy architecture. It shows that in average 0.278 seconds are needed to react to any single change in a given SSON. While Fig 4.11 shows that 32% of this time is being used for message exchange. Considering the low number of overlay nodes and thus the low number of needed adaptations, these figures implies that our system is scalable in terms of the number of policy objects needed to adapt an SSON.

The number of messages generated by our self organization composition algorithm is another factor that affects the scalability of our proposed autonomous management framework. Once a composition request is sent from the MC, it will be kept forwarded inside the search scope angle until it is received by the MS. Although the basic forwarding mechanism is based on broadcasting and learning, many factors support the scalability of our composition technique. First, the composition request forwarding is limited to those nodes that lie in inside the search scope angle (see Fig 7.2). Second, the composition request is being forwarded selectively to those nodes that are expected to provide a service for the request. And finally, learning rules were used to prevent the

composition request from going into loops in the network. This is evident in Fig. 7.4 were our proposed composition algorithm produced a very low number of messages compared to most popular techniques.

Viewing scalability as the maximum number of concurrent SSONs that could coexist in the same network while fixing its resources results in a different way to analyze and proof our autonomous management framework scalability. Although it is really hard to come up with a measure that quantifies the number of concurrent SSONs that can coexist while providing the best requested QoS, a deep look at how SSONs are being constructed can give us a strong hint on whether the system is scalable or not. As a matter of fact, the most efficient SSON is the one that uses the least possible packet latency, and any system will not be scalable if the SSONs latencies are way above the minimum possible latency. Fortunately, the minimum possible latency for an SSON can be measured by the shortest path latency between the MC and the MS. Since no system could ever produce latency smaller than the shortest path latency, dividing the SSON latency by the shortest path latency will give us a measure, called the stretch, of how efficient and scalable our framework in constructing SSONs. Fig. 4.8 shows that the average overlay path stretch is 1.76 when the resource discovery is centralized and Fig. 7.8 shows that the average overlay path stretch for our composition technique is 1.1. It is worth noting that when the stretch was 1.76, the resource discovery were carried out separately from the construction phase of the SSONs, while it was integrated in the construction phase when we reached the 1.1 stretch which is relatively low proves that our framework is constructing SSONs with latencies very close to optimal shortest path latency.

8.4 Research Work Limitations

As we explained earlier, the focus of the conducted research has been the development of an autonomous management system for SSONs. We identified the requirements for an autonomous management system and proposed a framework for the automation of SSONs management system. The framework has been presented as a multi-layered

model that utilizes context of users, applications, and the underlying network to perform autonomous management functionalities. The main limitations of the current research work can be summarized as follows:

1. The focus throughout this dissertation was on automating the management functions of SSONs. We have considered all the possible phases that the SSON go through during its life time. More specifically, we considered the creation, optimization, adaptation, and termination phases. Although SSONs represents a special type of overlay networks, they cannot be treated in the same fashion. One limitation is that, SSONs usually consists of limited number of nodes, these includes the MS, MC, and a set of MP that are needed to transform the requested media- located at the MS- from its current state to a state acceptable by the MC. Our study and experiment showed that the length of an SSON in terms of the number of participating nodes varies from 2 to 6 nodes. However, SSONs are customized and tailored to the specific demands of the users. In contrast overlay networks such as P2P networks consists of thousands up to millions of users and nodes. But they are generic and don't represent or satisfies the users specific requirements.

2. Although MPs provides value added functions to SSONs such as caching, synchronization and routing, they can be considered as a limitation for overlays in general. This is due to the fact that these MPs are located inside the network, i.e. not at the network edges, and their ownership is usually belongs to a certain service provider. Mandating that the multimedia session has to go through one or multiple media ports implies that the SSON is no longer controlled by the users but rather by the service providers that own, install, and control MPs. The rapid deployment of MPs thus might results in decreasing the number of new services that evolves over time. Although this might be desirable from the service providers' perspective, it limits the growth of the technologies.

3. The semantic resource discovery technique presented in Chapter 5 is being designed for the specific needs of SSONs. Although it provides comparable

results to the most popular protocols such as Chord and CAN, it might not be a ready solution for just any resource discovery problem. The reason is that, the design and implementation of our resource discovery technique considers and exploits the specific properties of SSONs. For example, the chaining of MPs to realize SSONs has to be in a specific order and this order starts from the MS and ends at the MC. Therefore we focus our search for resources in those areas located between the MS and MC. This might not be the case for many other applications.

List of Publications

Journal Publications

1. I. Al-Oqily, A. Karmouch, "Towards Automating Overlay Networks Management", Journal of Network and Computer Applications (Elsevier), to appear in *Special Issue on Service Oriented Computing: A New Horizon for Internet Appl Applications.* Accepted.
2. I. Al-Oqily, A. Karmouch, "SORD: a Fault-Resilient Semantic Overlay for MediaPorts Resource Discovery" in IEEE Transaction on Parallel and Distributed Systems, Revised and Submitted June,1, 2008.
3. I. Al-Oqily, A. Karmouch, "QoS Composition of Autonomic Entities", Journal of Network and Systems Management, Springer. Under preparation.

Conference Publications

1. I. Al-Oqily, A. Karmouch, "Automating Overlay Networks Management," *aina*, pp. 386-393, *21st International Conference on Advanced Networking and Applications* (AINA '07), May, 2007.
2. I. Al-Oqily, A. Karmouch, "A Lightweight Semantic Overlay Resource Discovery", t*he 14th IEEE International Conference On Telecommunications (ICT-MICC'07)*, May, 2007.
3. I. Al-Oqily, A. Karmouch, "Policy-Based Context-Aware Overlay Networks", *IEEE Global Information infrastructure symposia (GIIS'07).* July, 2007.
4. I. Al-Oqily, A. Karmouch, "Towards an Autonomic Management for Service Specific Overlay Networks", *5th IEEE Latin American Network Operations and Management Symposium (LANOMS'07), Sep. 2007.*

5. I. Al-Oqily, A. Karmouch, "A Self-Organization Composition for Autonomic Entities", *20th IEEE/IFIP Network Operations & Management Symposium. NOMS'08.*

6. I. Al-Oqily, A. Karmouch, "An Autonomic Service Architecture for Service Specific Overlay Networks," *2008 IFIP Conference on Wireless Sensors and Actor Networks (WSAN 08).*

Bibliography

[1] Niebert, N.; Schieder, A.; Abramowicz, H.; Malmgren, G.; Sachs, J.; Horn, U.; Prehofer, C.; Karl, H., "Ambient networks: an architecture for communication networks beyond 3G," *Wireless Communications, IEEE* , vol.11, no.2, pp. 14-22, Apr 2004.

[2] F.Hartung, S.Herborn, M.Kampmann, and S.Schmid, "Smart Multimedia Routing and Adaptation using Service Specific Overlay Networks in the Ambient Networks Framework," WWRF #12 ,Toronto. Nov 4-5, 2004.

[3] K.Sripanidkulchai, B.Maggs, and H.Zhang, "An Analysis of Live Streaming Workloads on the Internet," In ACM IMC, Oct. 2004. pp. 41-54.

[4] K.Yang, A.Galis, T.Mota, and S.Gouveris, "Automated Management of IP Networks through Policy and Mobile agents," Proc. of 4th Int. Workshop on Mobile Agents for Telecommunication Applications. LNCS-2521, Springer, Spain, Oct. 2002. Pp. 249-258.

[5] N.Damianou, N.Dulay, E.Lupu, and M.Sloman, "The Ponder Specification Language," Lecture Notes in Computer Science, Springer, Vol. 1995/2001. pp. 18-38.

[6] I. Al-Oqily, A. Karmouch, "Automating Overlay Networks Management," aina, , 21st International Conference on Advanced Networking and Applications (AINA '07), pp. 386-393,May, 2007.

[7] Al-Oqily, I.; Karmouch, A., "Policy-Based Context-Aware Overlay Networks," *Global Information Infrastructure Symposium, 2007. GIIS 2007. First International* , pp.85-92, 2-6 July 2007.

[8] I. Al-Oqily, A. Karmouch, "Towards Automating Overlay Networks Management", Journal of Network and Computer Applications (Elsevier), Special Issue on Service Oriented Computing: A New Horizon for Internet Applications. Accepted.

[9] Al-Oqily, I.; Karmouch, A., "A lightweight semantic overlay resource discovery," *Telecommunications and Malaysia International Conference on Communications,*

2007. ICT-MICC 2007. IEEE International Conference on, pp.702-707, 14-17 May 2007.

[10] I. Al-Oqily, A. Karmouch, "SORD: a Fault-Resilient Service Overlay for MediaPorts Resource Discovery" in IEEE Transaction on Parallel and Distributed Systems, Revised and Submitted June, 1, 2008.

[11] I. Al-Oqily, A. Karmouch, "Towards an Autonomic Management for Service Specific Overlay Networks", 5th IEEE Latin American Network Operations and Management Symposium (LANOMS'07), Sep. 2007.

[12] I. Al-Oqily, A. Karmouch, "An Autonomic Service Architecture for Service Specific Overlay Networks," 2008 IFIP Conference on Wireless Sensors and Actor Networks (WSAN 08).

[13] I. Al-Oqily, A. Karmouch, "A Self-Organization Composition for Autonomic Entities", 20th IEEE/IFIP Network Operations & Management Symposium. NOMS'08.

[14] Akyildiz, I.F.; McNair, J.; Ho, J.S.M., Uzunalioglu, H.; Wenye Wang, "Mobility management in next-generation wireless systems," *Proceedings of the IEEE* , vol.87, no.8, pp.1347-1384, Aug 1999.

[15] I.F.Akyildiz, J.Xie, S.Mohanty "A survey of mobility management in next-generation all-IP-based wireless systems," IEEE Wireless Communications, August 2004, pp. 16- 28.

[16] Saha, D.; Mukherjee, A.; Misra, I.S.; Chakraborty, M.; Subhash, N., "Mobility support in IP: a survey of related protocols," *Network, IEEE* , vol.18, no.6, pp. 34-40, Nov.-Dec. 2004.

[17] H. Schulzrinne, "Personal mobility for multimedia services in the Internet," in European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS), (Berlin, Germany), Mar.4-6, 1996. pp. 143-161.

[18] C. Perkins, "IP Mobility Support for IPv4," IETF RFC 3344, August 2002; http://www.ietf.org/rfc/rfc3344.txt

[19] C. Perkins, "Mobile Networking Through Mobile IP," In IEEE Internet Computing, Jan. – Feb. 1998. pp. 58-69.

[20] C. Perkins , D. B. Johnson,"Route Optimization in Mobile IP". IETF Internet draft, version 11, September 2001; http://www.ietf.org/proceedings/02mar/I-D/draft-ietf-mobileip-optim-11.txt

[21] G. Montenegro, "Reverse Tunneling for Mobile IP, revised," IETF RFC 3024, January 2001; http://www.ietf.org/rfc/rfc3024.txt

[22] G. Montenegro, "Sun's SKIP Firewall Traversal for Mobile IP," IETF RFC 2356, June 1998; http://www.ietf.org/rfc/rfc2356.txt

[23] K. D. Wong, H. Wei, A. Dutta, K. Young, "Performance of IP Micro-Mobility Management Schemes using Host Based Routing," Proc. 4th Int'l Symp. Wireless Personal Multimedia Communications (WPMC'01), 2001.

[24] E. Gustafsson, A. Jonsson, C. Perkins "Mobile IP Regional Registration", Internet Draft, IETF, December 2004; http://www.ietf.org/internet-drafts/draft-ietf-mipshop-hmipv6-04.txt

[25] Misra, A.; Das, S.; Dutta, A.; McAuley, A.; Das, S.K., "IDMP-based fast handoffs and paging in IP-based 4G mobile networks ," *Communications Magazine, IEEE* , vol.40, no.3, pp.138-145, Mar 2002.

[26] Das, S.; Mcauley, A.; Dutta, A.; Misra, A.; Chakraborty, K.; Das, S.K., "IDMP: an intradomain mobility management protocol for next-generation wireless networks," *Wireless Communications, IEEE* , vol.9, no.3, pp. 38-, June 2002.

[27] A. Valko, Design and Analysis of Cellular Mobile Data Networks, Ph. D. Dissertation, Technical University of Budapest, 1999.

[28] A.T.Campbell, J.Gomez, S.Kim, A.G.Valkó, "Design, Implementation, and Evaluation of Cellular IP," IEEE Personal Communications, August 2000, pp. 42-49.

[29] Ramjee, R.; Varadhan, K.; Salgarelli, L.; Thuel, S.R.; Shie-Yuan Wang; La Porta, T., "HAWAII: a domain-based approach for supporting mobility in wide-area wireless networks," *Networking, IEEE/ACM Transactions on* , vol.10, no.3, pp.396-410, Jun 2002.

[30] Dutta, A.; Wong, K.D.; Burns, J.; Jain, R.; McAuley, A.; Young, K.; Schulzrinne, H., "Realization of integrated mobility management protocol for ad-hoc networks," *MILCOM 2002. Proceedings* , vol.1, pp. 448-454 vol.1, 7-10 Oct. 2002.

[31] J. Rosenberg et. al. "SIP: Session Initiation Protocol", IETF RFC 3261, June 2002; http://www.ietf.org/rfc/rfc3261.txt

[32] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF RFC 3550, July 2003; http://www.ietf.org/rfc/rfc3550.txt

[33] D.R. Wisely, "SIP and conversational Internet applications," BT Technol J, Vol 19. No 2. April 2001, pp. 107-118.

[34] H.G. Schulzrinne and J.D. Rosenberg, "The Session Initiation Protocol: Providing Advanced Telephony Services Across the Internet," Bell Labs Tech. J., vol. 3, no. 4, October-December 1998, pp. 144-160.

[35] Wong, K.D.; Dutta, A.; Burns, J.; Jain, R.; Young, K.; Schulzrinne, H., "A multilayered mobility management scheme for auto-configured wireless IP networks," *Wireless Communications, IEEE* , vol.10, no.5, pp. 62-69, Oct 2003.

[36] F. Vakil, A. Dutta, J. C. Chen, S. Baba and Y. Shobatake, H. Schulzrinne, "Supporting Mobility for TCP with SIP," IETF internet draft, June 2001, work in progress; draft-itsumo-sipping-mobility-tcp-00.txt

[37] Politis, C.; Chew, K.A.; Tafazolli, R., "Multilayer mobility management for all-IP networks: pure SIP vs. hybrid SIP/mobile IP," *Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual* , vol.4, no., pp. 2500-2504 vol.4, 22-25 April 2003.

[38] K. Chew, C. Politis, and R. Tafazolli, "Performance Evaluation of Micromobility Protocols for All-IP Based Infrastructures," Wireless World Research Forum (WWRF), 7th meeting, Eindhoven, The Netherlands, 3-4 December 2002.

[39] H. Schulzrinne and E. Wedlund, "Application Layer Mobility using SIP," ACM Mobile Computing and Communications Review, Vol. 4, No. 3, July 2000, pp. 47-57.

[40] TS 23.228, "IP Multimedia Subsystem (IMS)," 3GPP, Release 6.

[41] P. Kim and W. Boehm, "Support for Real-Time Applications in Future Mobile Networks: the IMS Approach," Proceedings of WPMC'03, Oct. 2003.

[42] T.Renier, L.KimLynggarg, G.Castro, H.P.Schwefel, "Mid-Session Macro-

Mobility in IMS-Based Networks," in IEEE Vehicular Technology Magazine, vol 2, Iss 1, pp. 20-27, March 2007.

[43] K.L.Johnson, J.F.Carr, M.S.Day, M.F.Kaashoek, "The Measured Performance of Content Distribution Networks," Computer Comm., vol. 24, nos. 1-2, 2001, p. 202; www.cs.bu.edu/pub/wcw01/206.

[44] D. Kaye, *Strategies for Web Hosting and Managed Services*, John Wiley & Sons, 2002.

[45] I. Lazar and W. Terill, "Exploring Content Delivery Network," IEEE IT Professional, vol. 3, no. 4, 2001, pp. 47–49.

[46] Niebert, N.; Schieder, A.; Abramowicz, H.; Malmgren, G.; Sachs, J.; Horn, U.; Prehofer, C.; Karl, H., "Ambient networks: an architecture for communication networks beyond 3G," *Wireless Communications, IEEE* , vol.11, no.2, pp. 14-22, Apr 2004.

[47] W.T.Ooi, R.V.Renesse, and B.Smith, "The design and implementation of programmable media gateways", In Proc.NOSSDAV'00, Chapel Hill, NC, June 2000.

[48] D.Andersen, H.Balakrishnan, F.Kaashoek, and R.Morris , "Resilient Overlay Networks," Proc. 18th ACM Symp. on Operating Systems Principles (SOSP), Banff, Canada. pp. 131-145, Oct. 2001.

[49] J.Jannotti, D.Gifford, K.Johnson, M.Kaashoek, and J.O'Toole, "Overcast: reliable multicasting with an overlay network" Proc. USENIX OSDI, pp.14-14, Oct. 2000.

[50] L.Subramanian, I.Stoica, H.Balakrishnan, and R.Katz, "OverQoS: An overlay based architecture for enhancing Internet QoS," Proc. NSDI, pp. 6-6, California 2004.

[51] Yang-hua Chu; Rao, S.G.; Seshan, S.; Hui Zhang, "A case for end system multicast," *Selected Areas in Communications, IEEE Journal on* , vol.20, no.8, pp. 1456-1471, Oct 2002.

[52] Beichuan Zhang; Jamin, S.; Lixia Zhang, "Host multicast: a framework for delivering multicast to end users," *INFOCOM 2002. Twenty-First Annual Joint*

Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , vol.3, no., pp. 1366-1375, 23-27 June 2002.

[53] Akamai Corporation, http://www.akamai.com. August. 2008.

[54] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in ACM SIGCOMM 2001, pp. 149 - 160 , August 2001.

[55] Agarwal, S.; Chen-Nee Chuah; Katz, R.H., "OPCA: robust interdomain policy routing and traffic control," *Open Architectures and Network Programming, 2003 IEEE Conference on* , vol., no., pp. 55-64, 4-5 April 2003.

[56] A.Nakao, L.Peterson, and A.Bavier, "A routing underlay for overlay networks," in Proc. of ACM SIGCOMM, pp. 11-18, Aug. 2003.

[57] K. Shen, "Saxons: Structure management for scalable overlay service construction," In USENIX NSDI '04, pages 281–294, 2004.

[58] P. Francis, "Yoid: Extending the internet multicast architecture." http://www.aciri.org/yoid/docs/index.htm.

[59] Planetary Network Testbed, "http://www.planet-lab.org."

[60] Braynard, R.; Kostic, D.; Rodriguez, A.; Chase, J.; Vahdat, A., "Opus: an overlay peer utility service," *Open Architectures and Network Programming Proceedings, 2002 IEEE* , vol., no., pp. 167-178, 2002.

[61] Xbone, "http://www.isi.edu/xbone."

[62] L. Subramanian, I. Stoica, H. Balakrishnan, and R.H.Katz, "Overqos: Offering internet qos using overlays," *SIGCOMM Comput. Commun. Rev.* 33, iss.1, Jan. 2003, pp. 11-16.

[63] Z. Duan, Z. Zhang, and Y. T. Hou, "Bandwidth provisioning for service overlay networks," in SPIE ITCOM Scalability and Traffic Control in IP Networks (II), vol. 4868, pp. 139-150 , 2002.

[64] J. Shin, J. W. Kim, and C. J. Kuo, "Quality-of-service mapping mechanism for packet video in differentiated services network," IEEE Trans. Multimedia, vol. 3, no. 2, pp. 217–230, Jun. 2001.

[65] M. Sloman, Policy Driven Management for Distributed Systems, Plenum Press

Journal of Network and Systems Management, vol 2, no. 4, Dec. 1994, pp. 333-360

[66] N. Damianou, N. Dulay, E. Lupu, M Sloman: Ponder: A Language for Specifying Security and Management Policies for Distributed Systems, Imperial College Research Report DoC 2001, Oct. 2000

[67] E. Lupu and M. Sloman Conflict Analysis for Management Policies, Fifth IFIP/IEEE International Symposium on Integrated Network Management IM'97, San-Diego, May 1997, Chapman & Hall Publishers, pp 430-443

[68] M. Sloman. & J.D. Moffett, Domain Management for Distributed Systems, In Proc of the IFIP Symposium on Integrated Network Management, May 1989 pp 505-516.

[69] J. Moffett, M. Sloman, "Policy Hierarchies for Distributed Systems Management," IEEE Journal on Selected Areas in Communications, Vol. 11 No. 9, Dec. 1993, pp. 1404-1414

[70] R. Wies. Policies in Network and Systems Management - Formal Definition and Architecture. Journal of Networks and Systems Management, Vol. 2, No. 1, March 1994, pp. 63-83

[71] Edwards, W. K. Policies and Roles in Collaborative Applications. In Proc. ACM CSCW'96, Nov.16-20, Boston, MA, USA, pp. 11-20.

[72] Y. R., R. Guerin, and D. Pendarakis, "A Framework for Policy-based Admission Control," in IETF RFC 2753, Informational, Jan. 2000.

[73] A.Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, M. Huynh, A. Carlson, J. Perry, and S. Waldbusser, "Terminology for Policy Based Management," in IETF RFC 3198, Nov. 2001.

[74] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, "The COPS(Common Open Policy Service) protocol," in ETF RFC 2748, Jan. 2000.

[75] A.Ferdinando, P.McKee and A.Amoroso, "A Policy Based Approach for Automated Topology Management of Peer To Peer Networks and a Prototype Implementation," Proc of the IEEE 4th Inter Work. on Policies for Distributed Systems and Networks (POLICY 2003). Italy. pp: 235-238, 2003.

[76] M.Massimi and U.Wolz, "Peer-to-Peer Policy Management System for Wearable

Mobile Devices," Proc of the 17th IEEE Int. Symp. on Wearable Computers (ISWC'03), pp: 246-247, 2003.

[77] K.L. Calvert, An architectural framework for active networks, dARPA active nets document, 2001. Available from: <http://protocols.netlab.uky.edu/~calvert>.

[78] M. Solarski, E. Moeller, Challenges in active service deployment, in: ANTA'2002 (The First International Workshop on Active Network Technologies and Applications),Tokyo, Japan, March 2002

[79] A. Galis, S. Denazis, C. Brou, C. Klein (Eds.), Programmable Networks for IP Service Deployment, Artech House Books, 2004, ISBN 1-58053-745-6, p. 450. Available from: <http://www.artechhouse.com>.

[80] M.Solarski, M.Bossardt, and T.Becker, "Deployment and management of component-based services in active networks," The International Journal of Computer and Telecommunications Networking, Vol 50, Iss. 14, pp. 1389-1286, October, 2006.

[81] C.Dhillon, M.Bond, J.Griffioen, and K.L.Calvert, "Building layered active services," The International Journal of Computer and Telecommunications Networking, Vol. 50 , Iss. 14,pp. 2475 – 2487, October 2006.

[82] G. Cortese, R. Fiutem, P. Cremonese, S. D'antonio, M. Esposito, S. P. Romano, and A. Diaconescu, "Cadenus: creation and deployment of end-user services in premium IP networks," Communications Magazine, IEEE, vol. 41, pp. 54 – 60, Jan. 2003.

[83] S. Khaldoon, M. Damien, and L.Pascal, " A Scalable Middleware for Creating and Managing Autonomous Overlays," 2nd International Conference on Communication Systems Software and Middleware (COMSWARE 2007), pp. 1-8, 7-12 Jan. 2007

[84] K.Ragab, N.Y.Horikoshi, H.Kuriyama, and K.Mori, "Autonomous Decentralized Community Communication for Information Dissemination", IEEE CS Internet Computing magazine, Vol.8, Iss.3 , pp.29-36 , May-June 2004.

[85] K.Ragab, Y.Horikoshi, H.Kuriyama, and K.Mori, "Multi-layer autonomous community overlay network for enhancing communication delay," Proceedings of

the Ninth International Symposium on Computers and Communications (ISCC 2004), Vol.2, pp. 987-992, 28 June-1 July 2004.

[86] S.Ratnasamy, M.Handley, R.Karp, and S.Shenker, "Topologically-aware overlay construction and server selection," Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002). Vol. 3, pp. 1190-1199, 2002

[87] C.J.Lin, Y.T.Chang, S.C.Tsai, and C.F.Chou, "Distributed Social-based Overlay Adaptation for Unstructured P2P Networks," IEEE Global Internet Symposium, pp. 1-6, 11 May 2007.

[88] J. A. Pouwelse, P. Garbacki, J. W. A. Bakker, J. Yang, A. Iosup, D. Epema, M.Reinders, M. R. van Steen, and H. J. Sips, "Tribler: A social-based based peer to peer system," in 5th Int'l Workshop on Peer-to-Peer Systems (IPTPS), February 2006.

[89] P. Androutsos, D. Androutsos, and A. Venetsanopoulos, "Small world distributed access of multimedia data: an indexing system that mimics social acquaintance networks," Signal Processing Magazine, IEEE , vol.23, no.2pp, pp. 142– 153, Mar, 2006.

[90] IBM Corporation, "An architectural blueprint for autonomic computing," White Paper, Jun. 2006.

[91] IBM Corporation,"Autonomic computing - a manifesto," http://www.research.ibm.com/autonomic /manifesto/, Oct. 2001.

[92] J.Kephart , "Research Challenges of Autonomic Computing," Proc. of the 27th int. conf. on Software Engineering (ICSE'05), St. Louis, Missouri, P. 15 – 22, May 15–21, 2005.

[93] J.Kephart, and D.Chess, "The vision of autonomic computing," IEEE Computer Mag.. Vol. 36, No.1, PP.41–50. Jan. 2003. Forum, "Autonomic communication." http:// www.autonomic-communication.org.

[94] R.Farha and A.Leon-Garcia, "Blueprint for an Autonomic Service Architecture" 2006

[95] J.Nichols, H.Demirkan, and M.Goul, "Autonomic Workflow Execution in the

Grid," IEEE Tran. on Systems, Man, and Cybernetics—Part C: Applications And Review, VOL. 36, NO. 3, MAY 2006.

[96] E.Kasten and P.McKinley, "MESO: Supporting Online Decision Making in Autonomic Computing Systems," IEEE Trans. on Knowledge And Data Engineering, VOL. 19, NO. 4, April 2007.

[97] Bahati, R.M.; Bauer, M.A.; Vieira, E.M.; Baek, O.K.; Chang-Won Ahn, "Using policies to drive autonomic management," *World of Wireless, Mobile and Multimedia Networks, 2006. WoWMoM 2006. International Symposium on a* , vol., no., pp. 5 pp.-, 26-29 June 2006.

[98] Pena, J.; Hinchey, M.G.; Sterritt, R.; Ruiz-Cortes, A.; Resinas, M., "A Model-Driven Architecture Approach for Modeling, Specifying and Deploying Policies in Autonomous and Autonomic Systems," *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on* , vol., no., pp.19-30, Sept. 2006.

[99] F.Zhang, J.Gao, B.Liao, "Policy-Driven Model for Autonomic Management of Web Services Using MAS," Proc. of the 5th Int. Conf. on Machine Learning and Cybernetics, Dalian, 13-16 Aug. 2006. pp. 34-39.

[100] Balasubramaniam, S.; Barrett, K.; Donnelly, W.; van der Meer, S.; Strassner, J., "Bio-inspired Policy Based Management (bioPBM) for Autonomic Bio-inspired Policy Based Management (bioPBM) for Autonomic," *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on* , pp.3-12.

[101] P.McKinley, F.Samimi, J.Shapiro, and C.Tang, "Service Clouds: A Distributed Infrastructure for Constructing Autonomic Communication Services," Proc. of the 2nd IEEE Int. Symposium on Dependable, Autonomic and Secure Computing (DASC'06), 2006. pp. 341-348

[102] D. Xiangdong, S.Hariri, L.Xue, H.Chen, M.Zhang, S.Pavuluri, and S.Rao, "Autonomia: an autonomic computing environment," in Proc. of the IEEE Int. Performance, Computing, and Communications Conf., pp. 61–68, Apr. 2003.

[103] P.Grace, G.Coulson, G.Blair, L.Mathy, W.Yeung, W.Cai, D. Duce, and C. Cooper, "GRIDKIT: pluggable overlay networks for grid computing," in Proc. of the

distributed objects and applications conf. (DOA'04), Cyprus, p1463-81, October 2004.

[104] M.Parashar, H.Liu, Z.Li, V.Matossian, C.Schmidt, G.Zhang, and S.Hariri, "AutoMate: enabling autonomic applications on the grid," Cluster Computing, Vol. 9, No. 6, PP.161–174, 2006.

[105] Chess, D.M.; Segal, A.; Whalley, I.; White, S.R., "Unity: experiences with a prototype autonomic computing system," *Autonomic Computing, 2004. Proceedings. International Conference on* , pp. 140-147, 17-18 May 2004.

[106] S.Dobson, S.Denazis, A.Fernández, D.Gaïti, E.Gelenbe, F.Massacci, P.Nixon, F.Saffre, N.Schmidt, F.Zambonelli and A.Fernández, "A survey of autonomic communications," ACM Transactions on. Autonomous and Adaptive Systems, Vol. 1, No. 2, P. 223-259, Dec.2006.

[107] Napster: http://www.napster.com/

[108] M. Ripeanu, M.Bowman, J.S.Chase, I. Foster, and M. Milenkovic, "Globus and PlanetLab Resource Management Solutions Compared." Proc. of The 13th IEEE International Symposium on High Performance Distributed Computing, pp. 246–255, Jun. 2004.

[109] R. V.Renesse, "Scalable and Secure Resource Location", in Proceedings of IEEE Hawaii International Conference on System Sciences, January 2000. pp.4-7.

[110] E. Simonton, K.C. Byung, and S.Seidel, "Using Gossip for Dynamic Resource Discovery," Proc. of the 2006 Int. Con. on Parallel Processing (ICPP'06), pp: 319-328, 14-18 Aug, 2006.

[111] Dimakopoulos, V.V.; Pitoura, E., "Performance analysis of distributed search in open agent systems," *Parallel and Distributed Processing Symposium, 2003. Proceedings. International* , pp. 22-26 April 2003.

[112] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, "A Review of Routing Protocols for Mobile Ad Hoc Networks," Ad Hoc Networks, vol. 2, no. 1, pp. 1-22, Jan. 2004.

[113] Gnutella RFC, http://rfc-gnutella.sourceforge.net, 2003.

[114] V. Dimakopoulos, and E. Pitoura, "On the Performance of Flooding-Based Resource Discovery," IEEE Trans on Parallel and Distributed Systems, Vol. 17, No.

11, Nov. 2006.

[115] S. Herborn, Y. Lopez, and A. Seneviratne, "A Distributed Scheme for Autonomous Service Composition," MSC'05, ACM, Nov. 11, 2005. pp. 21-30.

[116] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", in Proceedings of ACM SIG-COMM 01, vol.31, iss. 4, pp. 161 - 172, Sep. 2001.

[117] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in Proceedings of the 2001 IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), ser. Lecture Notes in Computer Science, vol. 2218. Springer-Verlag, 2001, pp. 329–350.

[118] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Univ. California, Berkeley, CA, Tech. Rep. UCB/CSD-01-1141, 2001.

[119] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," In IPTPS, Cambridge, vol. 2429, pp.53-65, Mar 2002.

[120] D.Karger, E.Lehman, T.Leighton, M.Levine, D.Lewin, R.Panigrahy, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," Proc. 29th Annual ACM Symp. Theory of Comp., May 1997, pp. 654–63.

[121] National Institute of Standards and Technology (NIST), "Secure hash standard," U.S. Department of Commerce, National Technical Information Service FIPS 180-1, Apr. 1995.

[122] C. Plaxton, R. Rajaraman, and A. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," Proc. 9th Annual ACM Symp. Parallel Algorithms and Architectures, 1997. Pp. 311 – 320.

[123] P. Trunfio, D.Talia, P. Fragopoulou, C. Papadakis, M. Mordacchini,M. Pennanen, K. Popov, V. Vlassov, and S. Haridi , "Peer-to-Peer Models for Resource Discovery on Grids," CoreGRID Technical Report Number TR-0028, URL:

http://www.coregrid.net, Mar. 17, 2006.

[124] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. Technical Report UCB//CSD-03-1299, University of California, Berkeley, June 2004.

[125] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," in Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference (Middleware 2003), ser. Lecture Notes in Computer Science, vol. 2672. Springer-Verlag, 2003, pp. 21–40.

[126] Balazinska, M., Balakrishnan, H., Karger, D.: INS/Twine: a scalable peer-to-peer architecture for intentional resource discovery. In: Pervasive 2002 – 1st International conference on Pervasive computing, pp. 26–28. Zurich, Switzerland, 2002.

[127] O. D. Gnawali, "A keyword-set search system for peer-to-peer networks," Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States, June 2002.

[128] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, and J. Kubiatowics, "Approximate object location and spam filtering on peer-to-peer systems," in Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference (Middleware 2003), ser. Lecture Notes in Computer Science, vol. 2672. Springer-Verlag, 2003, pp. 1–20.

[129] C. Tang and S. Dwarkadas, "Hybrid global-local indexing for efficient peer-to-peer information retrieval." in Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI 2004). USENIX, 2004, pp. 211–224.

[130] S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, and M. Chen, "Making peerto- peer keyword searching feasible using multi-level partitioning," in Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004)., ser. Lecture Notes in Computer Science, vol. 3279. Springer-Verlag, 2005, pp. 151–161.

[131] P. Ganesan, Q. Sun, and H. Garcia-Molina, "Adlib: A self-tuning index for dynamic peer-to-peer systems," in Proceedings of the 21st International Conference on Data Engineering (ICDE'05). IEEE Computer Society, 2005, pp. 256–257.

[132] S.G.Doudane, and N.Agoulmine, "Enhanced DHT-based P2P Architecture for

Effective Resource Discovery and Management," Jour. of Network and Systems Management, Springer, Vol.15, Iss.3, pp. 335-354 , Sept., 2007.

[133] Y.Joung, Li.Yang, and C.Fang, "Keyword search in DHT-based peer-to-peer networks," IEEE Journal on Selected Areas in Comm., Vol.25, Iss.1, pp. 46-61, Jan. 2007.

[134] Lintao Liu, Lintao Liu, Kang-Won Lee, "Keyword fusion to support efficient keyword-based search in peer-to-peer file sharing," *ccgrid*, Fourth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04), pp. 269-276, 2004.

[135] Li, J., Loo, B.T.: On the feasibility of peer-to-peer web indexing and search. In: Proceedings of the 2nd IPTPS, pp. 20–21. Berkeley, CA, USA (2003)

[136] Ganesan, P.; Gummadi, K.; Garcia-Molina, H.; "Canon in G Major: Designing DHTs with Hierarchical Structure", *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on* , pp. 263-272, 2004.

[137] Alan Mislove and Peter Druschel. "Providing Administrative Control and Autonomy in Structured Peer-to-Peer Overlays". Proc. IPTPS04, San Diego, CA, vol. 3279 , pp.162-172, February 2004.

[138] Zhiyong Xu; Rui Min; Yiming Hu, "HIERAS: a DHT based hierarchical P2P routing algorithm," *Parallel Processing, 2003. Proceedings. 2003 International Conference on* , pp.187-194, 9-9 Oct. 2003.

[139] Artigas, M.S.; Lopez, P.G.; Ahullo, J.P.; Skarmeta, A.F.G., "Cyclone: a novel design schema for hierarchical DHTs," *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on* , pp. 49-56, 31 Aug.-2 Sept. 20.

[140] Z.Haiyang, and M.Huadong, " An Efficient Hierarchical Dht-Based Complex Query For Mul-timedia Information," IEEE Int. Conf. on Multimedia and Expo, pp.568-571, 2-5 July 2007.

[141] A. Crespo and H. Garcia-Molina, "Semantic Overlay Networks for P2P Systems," Stanford University, Tech. Rep., 2002.

[142] K. Aberer and P. Cudr´e-Mauroux. Semantic overlay networks. In VLDB Tutorial, page 1367, Aug. 2005.

[143] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," in Communications of the ACM, July 1970, vol. 13(7), pp. 422–426.

[144] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in Proc. INFOCOM, vol. 3, New York, NY, June 2002, pp. 1248–1257.

[145] I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Beyond term indexing: A P2P framework for web information retrieval. Informatica,30(2),pp.153–161, June 2006.

[146] K. Aberer, P. Cudr´e-Mauroux, M. Hauswirth, and T. V. Pelt, "Gridvine: Building Internet-Scale Semantic Overlay Networks," in Proceedings of International Conference on Semantic Web (ISWC'2004), 2004.

[147] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. DL meets P2P - distributed document retrieval based on classification and content. In ECDL, pp. 379–390, Sep. 2005.

[148] C. Tempich, S. Staab, and A. Wranik, "REMINDIN': Semantic Query Routing in Peer-to-Peer Networks based on Social Meta-phors," in Proceedings of WWW'2004, pp. 640 – 649, 2004.

[149] S. Yinglin, S.liang, X.Haung, and Y.Lin, "Resource discovery in locality-aware group-based semantic overlay of peer-to-peer networks," 1st Int. Conf. on Scalable information systems, Hong Kong, ACM Vol. 152, 2006.

[150] X.Tong, D. Zhang, and Z. Yang, "Efficient Content Location Based On Interest-Cluster in Peer-to-Peer System," Proc. of the 2005 IEEE International Conference on e-Business Engineering (ICEBE'05) 0-7695-2430-3/05, 2005.

[151] M. Klein and B. König-Ries. "Multi-layer clusters in ad-hoc networks - an approach to service discovery," In Proc.of 1st Intl Work.on P2P Computing (Co-Located with Networking 2002), Pisa, Italy, pp. 187–201, 2002.

[152] M. Ruta, T.D.Noia, E.D.Sciascio, and F.M Donini, "Semantic enabled resource discovery, Composition and substitution in pervasive environments," IEEE Conf. on Electrotechnical, (MELECON'06), pp. 754- 760, 16-19 May 2006.

[153] M.Klein, B.Konig-Ries, and P.Obreiter, "Service rings – a semantical overlay for service discovery in ad hoc networks". In: The Sixth International Workshop on

Network-Based Information Systems (NBIS2003), Workshop at DEXA 2003, Prague, Czech Republic,pp.180, 2003.

[154] M. Cardei, I. Cardei, and D.Z.Du, "Resource Management in Wireless Networking," Book Chapter "Efficient Resource Discovery in Wire-less AdHoc Networks: Contacts Do Help", in Springer; 1st edition, Jan 2005.

[155] M. Hauswirth, and R. Schmidt, "An overlay network for resource discovery in Grids," Proceedings of Sixteenth Workshop on Database and Expert Systems Applications, pp. 343-348, 22-26 Aug., 2005.

[156] Juan Li; Son Vuong, "A semantics-based routing scheme for grid resource discovery," *e-Science and Grid Computing, 2005. First International Conference on* , pp. 8 pp.-, 5-8 Dec. 2005.

[157] M.Klein, B.König-Ries, and P.Obreiter, "Lanes – a lightweight overlay for service discovery in mobile ad hoc networks," Technical Report 2003/6, Universität Karlsruhe, Faculty of Informatics, 2003.

[158] J.Tchakarov, and N.Vaidya, "Efficient Content Location in Wireless Ad Hoc Networks," Proceedings of IEEE International Conference on Mobile Data Management, pp: 74-85, Aug. 2004.

[159] B. Karp and H. T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in Proc. ACM Mobicom 2000, pp. 243-254.

[160] M. Mauve, J. Widmer, and H. Hartenstein, "A survey on position-based routing in mobile ad hoc networks," IEEE Network Mag., vol. 15, pp. 30-39, Nov./Dec. 2001.

[161] K.Bałos, T.Szydło, R.Szymacha, and K.Zieliński, "Context Dissemination and Aggregation for Ambient Networks," 1st European Conference on Smart Sensing and Context, Netherlands, vol. 4272, pp. 54-66,Oct. 26-27, 2006.

[162] J.Salo, A.Tarlano, and A.Galis, "Context Sources and their presentation in the WWI System Architecture," Wireless World Research Forum- WWRF18, Helsinki, Finland, June 13-15, 2007.

[163] N.Samaan, and A.Karmouch, "An automated policy-based management framework for differentiated communication systems," IEEE Jour. On Selected

Areas in Comm., Vol.23, Iss.12, PP.2236 – 2247, Dec. 2005.

[164] Z.Li and P.Mohapatra, " QRON: QoS-Aware Routing in Overlay Networks," IEEE Jour. On Selected Areas In Comm., VOL. 22, NO. 1, Jan. 2004.

[165] B.Vleeschauwer, F.Turck, B.Dhoedt, and P.Demeester, "Dynamic algorithms to provide a robust and scalable overlay routing service", Proc Int. Conf. on Information Networking (ICOIN 2006), Sendai, Japan, vol. 3961, pp. 945-954, 16-19 Jan, 2006.

[166] A.Lakhina, J.W.Byers, M.Crovella, and I.Matta,"On the geographic location of Internet resources," IEEE Journal on Selected Areas in Communications, Vol. 21, Iss. 6, PP.934 – 948, Aug. 2003.

[167] T.Melodia, D.Pompili, and I.F.Akyildiz, "On the interdependence of distributed topology control and geographical routing in ad hoc and sensor networks" IEEE Journal on Selected Areas in Communications, Vol. 23, Iss. 3, PP.520 – 532. March 2005.

[168] Chakraborty, D., Perich, F., Joshi, A., Finin, T. W., and Yesha, Y. 2002. A Reactive Service Composition Architecture for Pervasive Computing Environments. In *Proceedings of the IFIP Tc6/Wg6.8 Working Conference on Personal Wireless Communications* (October 23 - 25, 2002). C. G. Omidyar, Ed. IFIP Conference Proceedings, vol. 234. Kluwer B.V., Deventer, The Netherlands, 53-62.

[169] A.Medina, A.Lakhina, I.Matta, and J.Byers, "BRITE: Universal topology generation from a user's perspective," Boston University, Tech. Rep. 2001-003, 1 2001.

[170] H.Y.Tyan and C.J.Hou, JavaSim On-Line Manuals and Tuto-rials. Available on-line at <http://j-sim.cs.uiuc.edu/>.

[171] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," http://www.w3.org/ TR/wsdl, Mar., 2001.

[172] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadukal, D. Xu, "An XMLbased quality of service enabling language for the web", Tech. Rep UIUCDCS-R-2001-2212, Uni. Illinois at Urbana-Champaign, 2001.

[173] Narayanan, L., Opatrny, J., and Sotteau, D. 1999. All-to-all optical routing in

optimal chordal rings of degree four. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Baltimore, Maryland, United States, January 17 - 19, 1999). Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, 695-703.

[174] B. Parhami, "Chordal Rings Based on Symmetric Odd-Radix Number Systems," Proc. International Conf. on Communications in Computing, Las Vegas, NV, pp. 196-199, June 27-30, 2005.

[175] R. Beivide, C.Martınez, C.Izu, J.Gutierrez, J.Gregorio, and J.Miguel-Alonso, "Chordal Topologies for Interconnection Networks," Jour. Lect. notes comput. Sci, pp. 20-22, Oct. 2003.

[176] Y. Wang, D. DeWitt, J. Cai, "X-Diff: An effective change detection tool for XML documents", In Proc. ICDE, 2003.

[177] C. Dipanjan, A.Joshi, and Y. Yesha, "Toward Distributed service discovery in pervasive computing environments", IEEE Tran. On Mobile Computing, Vol. 5, No. 2, Feb. 2006.

[178] D.Xu, K.Nahrstedt, and D.Wichadakul,, "MeGaDiP: A wide-area media gateway discovery protocol", Proc. of IEEE Int. Conf. on Performance, Computing, and Communications (IPCCC). PP.257 - 263, 20-22 Feb 2000.

[179] E. Asmare, S.Schmid, and M.Brunner, "Setup and Maintenance of Overlay Networks for Multimedia Services in Mobile Environments." In Proc. of MMNS 2005, Barcelona, Spain, vol. 3754, pp. 82-95, October 2005.

[180] S.Ratnasamy, " A Scalable Content-Addressable Network," PhD thesis, University of California, Berkeley, October 2002.

[181] S.Tabbane, "An Alternative Strategy for Location Tracking," IEEE Jour. On Selected Areas in Comm., vol. 13, no. 5, pp. 880-892, June 1995.

[182] S.Kalasapur, M.Kumar, and B.A. Shirazi, "Dynamic Service Composition in Pervasive Computing," IEEE Transactions On Parallel And Distributed Systems, VOL. 18, NO. 7, JULY 2007.

[183] k. Fujii and T. Suda, "Semantics-Based Dynamic Service Composition," IEEE Jour. On Selected Areas In Comm., Vol. 23, No. 12, December 2005

[184] G. Xiaohui and K. Nahrstedt, "Distributed Multimedia Service Composition With Statistical QoS Assurances," IEEE Trans. On Multimedia, vol. 8, no. 1, February 2006.

[185] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan, "Adaptive and dynamic service composition in eFlow," in Proc. Int.Conf Advanced Inf. Syst. Eng., Stockholm, Sweden, vol. 1789, pp. 13-31, Jan. 2000.

[186] D. Mennie and B. Pagurek, "An architecture to support dynamic composition of service components," in Proc. 5th Int. Workshop Component-Oriented Program., Sophia Antipolis, France, 2000.

[187] M. Minami, H. Morikawa, and T. Aoyama, "The design and evaluation of an interface-based naming system for supporting service synthesis in ubiquitous computing environment," Trans. Inst. Electron., Inf. Commun. Eng., vol. J86-B, no. 5, pp. 777–789, May 2003.

[188] Sheng, Q. Z., Benatallah, B., Dumas, M., and Mak, E. O. 2002. SELF-SERV: a platform for rapid composition of web services in a peer-to-peer environment. In *Proceedings of the 28th international Conference on Very Large Data Bases* (Hong Kong, China, August 20 - 23, 2002). Very Large Data Bases. VLDB Endowment, 1051-1054.

[189] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, "Dynamic workflow composition using Markov decision processes," in Proc. 2nd Int. Conf. Web Serv., San Diego, CA, Jul. 6–9, 2004, pp. 576–582.

[190] P. Traverso and M. Pistore, "Automated composition of semantic web services into executable processes," in Proc. 3rd Int. Semantic Web Conf., Hiroshima, Japan, , vol. 3298, pp. 380-394, Nov. 7–11 2004.

[191] S. McIlraith and T. C. Son, "Adapting Golog for composition of Semantic Web services," In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning(KR2002), Toulouse, France, pp. 482-496, April 2002.

[192] Stollberg, M.; Haller, A., "Semantic Web services tutorial," *Services Computing, 2005 IEEE International Conference on* , vol.2, no., pp. xv vol.2-, 11-15 July 2005.

[193] S. Narayanan and S. McIlraith, "Simulation, verification and automated composition of Web service," In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA, pp. 77-88, May 2002. ACM. presentation available at http://www2002.org/presentations/narayanan.pdf.

[194] S. R. Ponnekanti and A. Fox, "SWORD: A developer toolkit for web service composition," in Proc. 11th World Wide Web Conf. (Web Eng. Track), Honolulu, Hawaii, May 7–11, 2002.

[195] E. Sirin and B. Parsia, "Planning for semantic web services," in Proc. Semantic Web Services Workshop 3rd Int. Semantic Web Conf., 2004.

[196] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau, "Automating DAML-S web services composition using SHOP2," in Proc. 2nd Int. Semantic Web Conf., Sanibel Island, FL, vol. 2870, pp.195-210, Oct. 2003.

[197] B. Limthanmaphon and Y. Zhang, "Web service composition with case-based reasoning," in Proc. 14th Australasian Database Conf., K.-D. Schewe and X. Zhou, Eds., Adelaide, Australia, 2003, pp. 201–208.

[198] S. McIlraith and T. Son, "Adapting Golog for composition of semantic web services," in Proc. 8th Int. Conf. Knowl. Representation Reasoning, Apr. 2002, pp. 482–493.

[199] M.Wilde, I.T.Foster, J.Vckler and Y.Zhao. "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," In SSDBM, pp. 37–46. 2002.

[200] Huang, An.-C.; Steenkiste, P., "Building self-configuring services using service-specific knowledge," *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on* , pp. 45-54, 4-6 June 2004.

[201] S.D.Gribble, M.Welsh, R.von Behren, E.A.Brewer, D.Culler, N. Borisov, S.Czerwinski, R.Gummadi, J.Hill, A.Joseph, R.Katz, Z.Mao, S. Ross, and B.Zhao, "The Ninja Architecture for Robust Internet-Scale Systems and Services," IEEE

Computer Networks, Special Issue on Pervasive Computing, vol.35, iss.4, Mar. 2001.

[202] F.Xiaodong, and K.Vijay, "Automatic creation and reconfiguration of network-aware service access paths," Elsevier, Computer Communications Vol.28 , pp. 591–608, 2005.

[203] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, "Dynamic workflow composition using Markov decision processes," in Proc. 2nd Int. Conf.Web Serv., San Diego, CA, pp. 576–582, Jul. 2004.

[204] P. Traverso and M. Pistore, "Automated composition of semantic web services into executable processes," in Proc. 3rd Int. Semantic Web Conf., Hiroshima, Japan, vol. 3298, pp. 380-39, Nov. 7–11, 2004.

[205] X. Gu, and K. Nahrstedt, "Distributed Multimedia Service Composition With Statistical QoS Assurances," IEEE Trans. on Multimedia, VOL. 8, NO. 1, Feb. 2006.

[206] J. Robinson, I. Wakeman, and T. Owen, "Scooby: Middleware for Service Composition in Pervasive Computing," Proc. Second Workshop Middleware for Pervasive and Ad Hoc Computing, pp. 161-166, 2004.

[207] A. P. Black, J. Huang, J.Walpole, and C. Pu, "Infopipes: An abstraction for multimedia streaming," Multimedia Syst. (Special Issue on Multimedia Middleware), vol. 8, no. 5, pp. 406–419, 2002.

[208] B. Raman and R. H. Katz, "An architecture for highly available wide area service composition," Comput. Commun. J. (Special Issue on Recent Advances in Communication Networking), Vol. 26, Iss. 15, pp. 1727-1740 , May 2003.

[209] D. Xu and K. Nahrstedt, "Finding service paths in a media service proxy network," in Proc. SPIE/ACM Multimedia Computing and Networking Conf. (MMCN'02), San Jose, CA, Jan. 2002.

[210] S.Camazine, J.Deneubourg, N.R.Franks, J.Sneyd, G.Theraulaz, and E.Bonabeau, Self-Organization in Biological Systems, Princeton University Press, Princeton, UK, ISBN:0-691-11624-5, 2003.

[211] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," http://www.w3.org/ TR/wsdl, Mar., 2001.

[212] S.Herborn, Y.Loez, and A.Seneviratne, "A Distributed Scheme for Autonomous Service Composition", Proc. of the 1st ACM Int. Work. on Multimedia Service Composition, MSC'05,Singapore, pp. 21-30, Nov.11, 2005.

[213] A.Lakhina, J.Byers, M.Crovella, and I.Matta, "On the geographic location of Internet resources," IEEE Jour. On Selec. Areas In Comm., Vol. 21, No. 6, Aug. 2003.

[214] T.Pfeifer, Automatic conversion of communication media, Ph.D. Thesis, TU-Berlin, GMD 2000.

[215] Kolodner, *Case-based reasoning*. Morgan Kaufmann Publishers Inc., 1993.

[216] G.Tesauro, "Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies," *Internet Computing, IEEE* , vol.11, no.1, pp.22-30, Jan.-Feb. 2007.