Supporting QoS-aware Service Discovery in Ubiquitous Computing Environments.

Jinshan Liu

N^{O} Ordre de la thèse :

THÈSE

présentée

DEVANT L'UNIVERSITÉ DE VERSAILLES Saint-Quentin-en-Yvelines

pour obtenir

le grade de : ***DOCTEUR DE L'UNIVERSITÉ DE VERSAILLES***

Mention : **Informatique**

PAR
JINSHAN LIU

Équipe d'accueil : **INRIA, Projet ARLES**

TITRE DE LA THÈSE :

Découverte de services sensible à la qualité de service dans les environnements de l'informatique diffuse

SOUTENU LE 11 / 07 / 2006 devant la commission d'Examen

COMPOSITION DU JURY

Cecilia MASCOLO (University College London)	Rapporteur
Karl ABERER (École Polytechnique Fédérale de Lausanne)	Rapporteur
Serge FDIDA (Laboratoire d'Informatique de Paris 6)	Examinateur
Nicole LÉVY (Université de Versailles Saint-Quentin-en-Yvelines)	Examinateur
Valérie ISSARNY (INRIA)	Directrice de Thèse

Abstract

With the advent of portable devices (e.g., smartphones) and the advances in wireless networking technologies (e.g., WLAN, GPRS, UMTS), the vision of ubiquitous computing is becoming a reality. It aims to facilitate user tasks through the seamless utilization of heterogeneous computing and communication capabilities (represented as *services*) available in the environment. Service discovery, which is necessary for achieving the above goal, must be aware of the service's non-functional properties due to the challenges posed by ubiquitous computing, such as device portability and mobility. This thesis proposes an overall solution that supports QoS-aware service discovery in ubiquitous computing environments. Our contribution lies in substantiating QoS awareness in the following three aspects. Firstly, during the process of discovering services, the expiring wireless links resulting from device mobility are identified and avoided since they cause service failures and thus hamper service reliability. Secondly, as multiple services can be discovered, a comprehensive utility function is proposed to evaluate services in terms of their various non-functional properties, meanwhile taking into account the service user's preferences among them, for the purpose of selecting the best one. Thirdly, to avoid untrustworthy services, a distributed reputation mechanism is proposed to facilitate the evaluation of the service host's trustworthiness. The above three proposed solutions are extensively evaluated respectively, based on analysis and simulation. They are further incorporated into a middleware that supports QoS aware Web service discovery in ubiquitous computing environments. A prototype implementing the middleware is deployed and evaluated. The results show that the overhead introduced by QoS awareness seems reasonable.

Dedicated to my parents

Acknowledgments

This thesis represents the end of my journey of obtaining my Ph.D. degree. I wish to express my gratitude to a number of people who have made this journey pleasant and memorable.

First and foremost, I would like to thank my supervisor, Valérie Issarny, who has being continuously offering her advice and encouragement during the course of this thesis. What I have accomplished so far would not be possible without her invaluable guidance and support.

I would like to thank Dr. Cecilia Masocolo and Prof. Karl Aberer for being my *rapporteurs* in spite of their busy schedules. Their insightful comments have helped me improve my thesis. Thanks are also due to Prof. Nicole Lévy, for being the president of my jury and Prof. Serge Fdida for being present in my jury.

I feel fortunate to have met the people in the INRIA-ARLES team that have been so friendly and supportive. I want to especially thank Nikolaos for his friendship and encouragement and for being company in the INRIA shuttle. I also want to thank Sonia, Pierre-Guillaume, Roberto, Ferda, Daniele, David, Damien(s), Manel, Françoise, Oriana, Malika, Khoi and Rafik for the great time and unforgettable memories. Thanks are also extended to Ahmad Abdulwakeel for his help in implementation of the prototype. And I am greatly indebted to Emmanuelle for her generous help.

Last but certainly not least, I thank my parents, my brother Jingshu and my sister Bingmei. Their love and care have made it possible for me to make it this far.

Contents

List of Figures

List of Tables

I

Introduction

In 1996, Mark Weiser summarized the last fifty years of computing history and stated that it had been witnessed two major eras and we were entering the era of ubiquitous computing [Weiser and Brown, 1996]. In the first era of "mainframe era", many people were tied to a single (mainframe) computer which was mostly run by experts behind closed doors. The second era connected individuals to desktops and laptops and computers became personal. In the third era of *ubiquitous computing*, one user has access to many computers (devices), which assume various forms (e.g., mobile phones, sensors) and are interconnected.

These devices along with their software components (e.g., a dictionary service) and contents (e.g., a personal blog) pose as abundant and even explosive resources accessible to a user. These heterogeneous resources are often generalized and abstracted as *services* (e.g, [Czerwinski et al., 1999, Papazoglou and Georgakopoulos, 2003]). Such plentiful services should benefit instead of overwhelm users. It thus requires support of dynamic and efficient service discovery to facilitate the realization of user-centric tasks. As these services can exhibit great diversity in their non-functional properties due to factors such as service providers' computing power, it calls for incorporating QoS awareness in the process of service discovery, which is the focus of this thesis.

The rest of this chapter is organized as follows. We first elaborate the motivation of our work on supporting QoS-aware service discovery in Section I.1. Then we present the structure of this document in Section I.2.

I.1 Motivation

In order to facilitate the utilization of the (digital) services available in the environment, we address in this thesis the support of QoS-aware service discovery in ubiquitous computing environments, which is illustrated with the following scenario.

> On a sunny afternoon, Bob is browsing his schedule on his smartphone while waiting for his friends in a campus cafe. He plans to watch a new movie with his friends and would like to know more information about the movie. With a single click, he gets some reviews on the movie and finds a movie trailer. He browses the reviews, watches the movie trailer and happily finds that the movie seems to be even more exciting than he has expected.

The above scenario can be supported by an intelligent system running on Bob's handheld. At first, although the cafe has installed a base station for Internet access, it is quite slow as many people are using it for surfing the Internet. Therefore, the system chooses to send Bob's request for "more information about a movie" through the ad hoc network of handhelds in the cafe. Within a short time (e.g., 0.5 second), the system has collected several movie reviews and also chosen a movie trailer. The reviews come from personal blogs on other people's handhelds and they are provided for free or 0.01 cybeuro as courtesy (*cybeuro* is a cyber currency used for buying and selling services). There are actually 4 identical trailers available (e.g., copies downloaded from the film website) in Bob's surrounding environment. The first one comes from Alice, who is just passing by. The system detects that her smartphone is very likely to get out of reach soon and thus does not even send her the query. The second comes from Conan, who is identified by the system as a "dishonest entity", since Conan sent a virus during the last encounter with Bob. The other two trailers come from Dennis and Elton. Dennis' smartphone, however, has not much battery power left and therefore its system accordingly raises the price (in cybeuro) of the service providing the trailer. In addition, Elton's smartphone is equipped with a new wireless card and can send the trailer really fast. Considering the price and performance, Bob's system selects Elton's trailer and presents it to Bob. Note that all of the above actions of finding and selecting services are automatically carried out by the system behind the scene and do not need any user intervention (either from Bob or others).

The above scenario presents an example of QoS-aware service discovery in ubiquitous computing environments (e.g., a campus cafe). Bob's hand-

held can carry out the following functions: locating a service based on both functional (e.g., movie review) and QoS properties (e.g., reliability); selecting among service instances based on their QoS and prices (e.g., determined by the service's battery consumption); storing the interaction histories for future reference. Although all the above seems achievable using current technologies, there remain open issues. In particular, how to handle mobility issues when users move around (i.e., how does Bob's system detect that Alice is just passing by)? How to select the best among several services, all of them satisfying the user's requirements (i.e., how to select between Dennis and Elton's services)? And how can two entities interact if they do not know each other (i.e., what can Bob do if he never met Elton before)?

I.2 Contribution

This thesis addresses the above issues by supporting QoS-aware service discovery in ubiquitous computing environments. The support is substantiated in three sub-steps of service discovery: service location (i.e., how to find required services), service selection (i.e., how to choose among the qualified services) and trustworthiness evaluation of services (i.e., how to judge the honesty of a service provider).

In chapter II, we study ubiquitous computing system architectures and state the motivation for QoS support during service discovery in the following three aspects: mobility awareness to improve service reliability; QoS-aware service selection to choose the best service instance depending on users' preferences among service's QoS properties and price; a robust and incentive compatible reputation mechanism to evaluate trustworthiness of a service provider.

Chapter III surveys the state of the art of service discovery in ubiquitous computing environments, in particular, the handling of the aspects as stated above, i.e., service location, selection and reputation-based trust management. The existing work does not handle these aspects sufficiently, which leads us to devise supporting solutions in the following chapters.

Chapter IV presents Signal Strength based Service Location (S3L) that improves robustness of the discovered services against device mobility for higher service reliability. Using signal strength, S3L identifies the expiring links that are likely to break soon because of device mobility. By avoiding those unstable links, S3L shows considerable improvement in the reliability of discovered services.

As the service location process may find multiple qualified service instances, we describe how to select the one among them that best matches the user's requirements and preferences in terms of service QoS and price in Chapter V. We first present a comprehensive utility function to evaluate a service with respect to various QoS properties, price and user's preferences among them. To tackle the selfishness of autonomous devices belonging to different persons or organizations, we use Vickrey auction as a pricing model to motivate entities to reveal the truthful price to ensure the selection of the most suitable service provider.

As devices can be selfish due to resource limitation and misbehavior is possible given the openness of the network, a mechanism for evaluating trustworthiness needs to be in place to avoid dishonest service providers. Chapter VI presents a fully distributed reputation mechanism that combines direct and indirect experiences (i.e., recommendations from others) for evaluating an entity's trustworthiness. It not only shows robustness against rumors (i.e., untruthful recommendation), but also motivates entities to help each other by providing truthful recommendations.

The three above proposals are integrated into a middleware that supports QoS-aware Web service discovery in ubiquitous computing environments, presented in Chapter VII. A prototype implementing the middleware is deployed and evaluated, especially, in terms of the overhead of introducing QoS-awareness.

Chapter VIII summarizes this thesis and our contributions. It later explores some future research directions, continuing and beyond this thesis.

II

System Architecture for Ubiquitous Computing

II.1 Ubiquitous Computing Vision

The vision of *ubiquitous computing* [Weiser, 1991] refers to the creation of environments saturated with a spectrum of heterogeneous computing and communication capabilities, which seamlessly integrate with the physical world [Satyanarayanan, 2001]. It aims to facilitate daily tasks and enhance user productivity through the utilization of those capabilities in an unobtrusive fashion, such that they completely blend in the physical environment and become "invisible". Such capabilities cover the spectrum ranging from traditional devices (e.g., speakers), wireless mobile devices (e.g., cellphones) to smart devices (e.g., badges, sensors, intelligent appliances). For example, an intelligent alarm clock can wake a user up fifteen minutes earlier than it is normally set, if it detects via network connection that there is a traffic jam on the user's way to office. Through any device whether mobile or not, users can interact with the environments *anytime*, *anywhere* [Miller and Pascoe, 2000]. Meanwhile, the environment can detect the presence of users and devices and integrate them as a part of it [Lyytinen and Yoo, 2002].

One closely related field of ubiquitous computing is mobile computing, which essentially enables a user equipped with her mobile device to access computing capabilities *anytime, anywhere*. The differences between the two computing paradigms lie in (1) mobile computing assumes a *reactive* approach to access computing capacities while ubiquitous computing assumes a *proactive* "all the time anywhere" approach [Saha and Mukherjee, 2003]. The difference results from different goals: in mobile computing, users require con-

tinuous access to computing capabilities with their mobile device; while in ubiquitous computing, the (intelligent) environments intend to proactively satisfy users' needs with available capacities; (2) mobile devices in mobile computing are mainly used to access capacities and mostly assume the role of *client*, while in ubiquitous computing, these devices can also act as *thin servers* [Weiser and Brown, 1996] that offer resources/services for others; (3) besides mobility support, ubiquitous computing also needs to support other properties such as smartness and invisibility to ensure that users have seamless access to computing capabilities.

II.1.1 Enabling Elements

The vision of ubiquitous computing is becoming a reality, thanks to the following twofold facts [Saha and Mukherjee, 2003]: computing devices are becoming increasingly powerful, smaller and affordable [Want and Pering, 2005], leading to populous deployment of them in living and working spaces; the wireless networking technology is rapidly progressing, making it possible to connect various devices with multiple networking paradigms.

The first paradigm is through network infrastructure, which is assumed to be always accessible for nomadic mobile devices. Interactions with the (intelligent) environment are carried out through the infrastructure. An example is a home wireless LAN that interconnects all home devices and provides access to capabilities in the "smart" home environments. Such an infrastructure manages and offers rich facilities that are ready for use, but requires deployment and maintenance and cannot assume to be always available.

Therefore, in order to achieve "all the time everywhere" access to resources in ubiquitous computing environments [Saha and Mukherjee, 2003], it necessitates a more flexible alternative for networking. Mobile Ad hoc NETworks (MANET) pose as a good choice: mobile devices establish connections on the fly with peer devices when needed [MANET, 2005]. The devices (nodes) are free to move around and the network can be reorganized arbitrarily. Nodes communicate with each other using ad hoc routing protocols [Perkins, 2001], which dynamically find routes in spite of changing network topology.

Compared to infrastructured networks, MANETs have the advantage of better availability: they are always accessible, in contrast with the former which can become unavailable when the access point is overloaded. Also, MANETs are deployment-free, while networking infrastructures require efforts of setup and maintenance. Moreover, MANETs can be used to extend the coverage area of the network infrastructures by having the nodes with

access to infrastructure relay traffic for those that are out of coverage. Finally, MANETs realize spontaneous networking of devices and support impromptu interaction between entities, which is a desirable feature for ubiquitous computing [Kindberg and Fox, 2002].

In addition to the above advantages, MANETs can help alleviating the problem of "uneven conditioning", as mentioned in [Satyanarayanan, 2001]. Uneven conditioning of ubiquitous computing environments arises from the different degree of penetration of ubiquitous computing technology into various physical environments. For example, a well-equipped conference room generally offers more facilities than a street. Using mobile ad hoc networks empowers users with awareness of not only the resources that are embedded in the environment, but also those from other mobile devices. Therefore, a user equipped with her device can dynamically establish a "smart space" around her, even when she is moving, to exploit the resources available in the surrounding environments which are not necessarily "smart". For example, instead of referring to base stations which are overloaded or unavailable or costly, a tourist can refer to the dictionary service provided by a nearby PDA to translate the sign she has just seen.

In summary, MANET poses as a flexible and suitable networking paradigm for interconnecting devices in ubiquitous computing environments. Hence, our work on service discovery in ubiquitous computing environments assumes ad hoc networking of devices. The terms of *device*, *node* and *entity* are used interchangeably in the rest of this thesis. However, MANETs complement rather than completely replacing infrastructured networks [1]. Although interactions among entities in ubiquitous computing environments are preferably carried out on MANETs due to the above reasons, it is reasonable to assume that a user has access to network infrastructure from time to time, as home/office environments are accessed on a daily base.

II.1.2 Characteristics and Challenges

As ubiquitous computing supports mobility, it inherits the characteristics and challenges of mobile computing [Forman and Zahorjan, 1994, Satyanarayanan, 1996] (such as device mobility and portability). Together with the introduction of new elements of ubiquitous computing (such as "thin servers"), the devices who provide and utilize resources in ubiquitous computing environments have the following characteristics: the devices need to be portable and thus are resource constrained; the devices exhibit great mobility; the devices

[1] unless in certain special geographical environments such as deserts.

exhibit great heterogeneity in capability and resource availability; the device mobility also increases the network's openness and makes it very likely for an entity to encounter others which it has no or very little knowledge of.

Device Portability. Although handheld devices (e.g., PDA, smartphones) are getting more and more powerful, they always need to be compact and portable. The size limit and portability requirement put constraint on the number of functionalities that can be integrated into a portable device.

First, mobile devices have limited computing capability. Although fueled by Moore's law, processors will continue to shrink while increasing in capability and capacity, new applications will demand ever-greater processing capabilities [Want et al., 2002]. For example, it is still costly to implement public key encryption on current PDAs. Second, devices are normally powered by battery, which is not only limited in capacity, but also progressing very slowly. From year 1990 to 2003, the battery's energy density has improved by a factor of only three [Paradiso and Starner, 2005].

Issues resulting from resource limitation of portable devices include the devices' short running time and their incapability of carrying out expensive computation. It also increases the system's dynamics: nodes can turn off when the battery runs out or just for saving energy.

Device Mobility. Mobile devices require wireless network access [Forman and Zahorjan, 1994]. Although technical advances in wireless networking technology empower higher throughput, wireless communication is much more difficult than wired counterpart since the former is more susceptible to interference, which leads to varied bandwidth and higher error rate [Rappaport, 2002]. This is complicated by the fact that, due to the movement of nodes, the network bandwidth fluctuates with time and wireless connection can be degraded or even lost.

Mobility has been divided into three categories [Roman et al., 2000]: (i) device mobility; (ii) personal mobility, which refers to the mobility of users that do not necessarily have a device; (iii) computational mobility that relates to migration of code over physical nodes. The first two are inherent in ubiquitous computing environments, while the third is essentially a software technology related to *mobile agent*. In ubiquitous computing environments, as it is commonplace for a user to bear one or multiple computing devices (e.g., cellphone, active badge), personal mobility and device mobility converge – a user's device moves along with her, performs her tasks and interacts with the environment on her behalf.

Device Heterogeneity. Devices exhibit high heterogeneity in inherent capabilities (e.g., computing power) and changing characteristics such as resource richness (e.g., battery level) and mobility (e.g., different moving speed). Such diversity has a great impact on service provisioning, e.g., a PDA generally takes longer latency than a laptop to provide the same service; while for the same PDA, the service latency can vary depending on its load.

Openness. Device mobility makes nodes' joining and leaving of a network much more frequent than in traditional wired networks. It makes it very commonplace for a node to encounter entities that it never met before.

An arising issue from the network's openness is trustworthiness. Entities belonging to different organizations or persons need to trust each other to make interactions possible. Lacking of security infrastructure requires an entity to make fully autonomous security decisions [Cahill et al., 2003]. This is aggravated by the fact that a device tends to be selfish because providing services consumes limited resources. It therefore requires the enforcement of cooperation among autonomous devices, especially for MANETs [Obreiter et al., 2003, Buttyan and Hubaux, 2003, Zhong et al., 2003, Marti et al., 2000], where the networking operations require the nodes to forward packets for each other. In order to save battery and thus extend lifetime, a node can refuse to forward packets for others while seeking such favors from others. One solution to enforce cooperation is to introduce service charge (e.g., [Buttyan and Hubaux, 2003] and [Zhong et al., 2003]). Every node owns an amount of currency kept with a counter. Sending a packet requires a node to reward the intermediate nodes that forward it. With such a mechanism, the nodes are forced to earn currency by forwarding packets for others. The counter can be kept on a trusted and tamper resistant hardware module [Buttyan and Hubaux, 2003] or a bank node running credit clearance service (CCS) [Zhong et al., 2003]. Both are viable in ubiquitous computing environments: tamper resistant hardware can be built within small devices (e.g., smartcards [Kommerling and Kuhn, 1999]), while the banker node does not have to be accessible during the interaction. Therefore, we assume the use of virtual currency and existence of charging/rewarding mechanisms for service usage/provision. In contrast with packet forwarding, different services can have different prices.

Summary. Table II.1 summarizes the device characteristics and the resulting challenges as elaborated above. These challenges make it very difficult and error-prone to build applications in ubiquitous computing environments, soliciting the needs of middleware to facilitate such process.

Device Characteristics	Challenges
Device portability	Limited battery power
	Limited processing power
Device mobility	Disconnection
	Bandwidth variability
Device heterogeneity	Different capacity and resource
Network openness	Encounter without knowledge

Table II.1: Faced challenges in ubiquitous computing environments

II.2 Ubiquitous Computing Middleware

Ubiquitous computing subsumes mobile computing, which essentially empowers the user with the capability of physically moving computing services with her. This has been realized by reducing the size of the computing devices and providing lightweight devices with access to computing capacity over wireless networks [Satyanarayanan, 1996]. Mobile devices are resource-constrained compared to static ones, in particular, they have limited battery lifetime; wireless connectivity available to mobile devices is highly variable. Therefore, middleware systems have been proposed to handle such constraints. Their main objective is to assist the development of services on resource-constrained devices in presence of mobility [Issarny et al., 2004]. The examples include content management on mobile nodes, such as data sharing over mobile ad hoc networks [Boulkenafed and Issarny, 2003, Mascolo et al., 2001] and handling disconnected operations [Ekenstam et al., 2001]; adaptation to varying available resources [Noble and Satyanarayanan, 1999].

Middleware architectures have also been proposed for ubiquitous computing environments (e.g., Gaia [Roman et al., 2002] and Centaurus [Kagal et al., 2002]). They focus on creating and maintaining a middleware infrastructure, which is responsible for managing the devices and resources in the environment, such that a user (her device) can integrate on-the-fly into the intelligent environment when entering it. The infrastructure offers various facilities (e.g., resource discovery) for the devices, which, meanwhile, become part of the infrastructure (e.g., by registering their offered services).

Besides the middleware infrastructure, other middleware have also been proposed to facilitate carrying out user tasks proactively, i.e., with very little or no intervention of the user. Since user-carried devices often have limited resources, they need the help of remote resources in order to realize user

tasks. Therefore, middleware has been proposed to implement resource discovery (e.g, [Chakraborty et al., 2006, Liu and Issarny, 2005]) to benefit from the available resources in ubiquitous computing environments. Resource discovery middleware dynamically configures and updates the resource availability in spite of the evolution of environments (e.g., resources appear and disappear).

Proactivity requires context-awareness, which is considered as a prerequisite of having a minimally intrusive ubiquitous computing system [Satyanarayanan, 2001, Saha and Mukherjee, 2003]. Thus middleware has also been proposed to manage context and incorporate context awareness to help delivering tailored services to users (e.g., [Hightower et al., 2002, Ranganathan and Campbell, 2003]). *Context* is defined as any information that can be used to characterize the situation of an entity (e.g., a person) that is relevant to the interaction between the user and the application [Dey, 2001]. Context can be very rich. For example, a user's context can include attributes such as physical location, physiological state (e.g., body temperature), personal schedules, etc. It is thus necessary for the middleware to collect and aggregate context information from potentially heterogeneous technologies and present it to the applications in a generic manner, regardless of how it is obtained [Hightower et al., 2002]. The middleware also provides abstracting and reasoning of the context information (e.g., [Ranganathan and Campbell, 2003]), which can be based on predefined rules or dynamically learned if the contexts are difficult to capture (e.g., user mood).

Usage of various resources available in the environment needs to be supervised by a security mechanism, e.g., regarding what entities are entitled to access a certain resource. Traditional security measures such as authentication and access control fall short of supporting ubiquitous computing environments featuring openness due to the large number of devices/resources that need be configured. In addition, portable devices have limited computing power and thus cannot afford expensive computations (e.g., asymmetric encryptions) [Creese et al., 2004]. Moreover, in delivering user-tailored services, ubiquitous computing systems have access to the various information regarding users' preferences, movement, habits, etc. This poses as a severe privacy threat for users [Campbell et al., 2002] as uncontrolled usage of such information can lead to consequences from targeted spam to blackmail. Therefore, middleware has also been devised to handle security and privacy (e.g., [Campbell et al., 2002]).

The ubiquitous computing middleware such as those presented above are not orthogonal. They are often interleaved and combined towards more powerful functionalities and more advanced features. For example, resource dis-

covery is often integrated with context information to find more precise and personalized resources that meet user needs (e.g., [Raverdy et al., 2006, Capra et al., 2005]). Some cellphone services (e.g., AT&T M-mode service) provide a feature to allow the users to make location-aware queries such as finding the nearest cinema, based on the location data on the current cell tower in use. In the meantime, context information such as location information should be protected from being leaked as it is generally considered private. Secure resource discovery adds protection mechanism (e.g., access control) for resource information [Zhu et al., 2004] and the communication during the discovery process can be encrypted and authenticated [Hodes et al., 2002].

II.3 Service Discovery in Ubiquitous Computing Environments

The heterogeneous resources available in the ubiquitous computing environment can be generalized as *services*, leading to Service oriented Computing (SoC), which is a computing paradigm that utilizes services as fundamental elements for developing applications [Papazoglou and Georgakopoulos, 2003]. SoC evolves from distributed object-oriented and component-based computing [Baker and Dobson, 2005]. The latter (e.g., with CORBA) realizes great level of flexibility through transparent service localization (e.g., by defining service interface with interface definition language (IDL)) and dynamic binding (e.g., through Dynamic Invocation Interface (DII)). However, object-oriented and component-based computing do not handle well heterogeneity (e.g., CORBA and non-CORBA middleware) and autonomy (e.g., objects belonging to different organization) [Baker and Dobson, 2005, Huhns and Singh, 2005]. SoC raises the level of abstraction while preserving the advantages of object orientation such as modularity and encapsulation [Sen et al., 2005]. It seeks to establish a standard way of making resources and capabilities available for use by others in the form of services over wide range of computing devices (such as PDAs) and software platforms (e.g., UNIX or Windows) [Papazoglou, 2003]. SoC thus reduces the complexity and increases efficiency of software development by allowing reuse of functionality provided by aggregated objects [Dokovski et al., 2004].

A service is a set of functionalities provided by one entity for the use of others [OASIS, 2005]. It is characterized by its functional and non-functional attributes and is accessible by other services. Generally, SoC involves the following three entities [Huhns and Singh, 2005]: *service client* is an entity in need of services; *service provider* is an entity that offers services; *service directory* is

an entity that stores service information and handles service lookup requests.

Applying SoC in ubiquitous computing environments leads to *Ubiquitous Service-Oriented COmputing* (*USoCo*). An entity in ubiquitous computing environments, whether mobile or not, can assume the role of service client, service provider or both. Besides the advantages in terms of its handling of heterogeneity and abstraction, SoC fits ubiquitous computing thanks to its *minimalist philosophy* [Sen et al., 2005], i.e., an entity only needs to carry a small amount of codes locally and discover and exploit other services to realize its tasks. In the following, *USoCo environments* refer to the ubiquitous computing environments where all resources are presented as services.

According to the definition of Wikipedia[2], service discovery refers to the functionality of automatic detection of services offered by the devices on a computer network. Service discovery empowers devices to properly discover services and exploit them. It is considered the keystone for a service oriented computing framework [Sen et al., 2005] and an essential enabler for ubiquitous computing vision [Zhu et al., 2005] to realize service access *anytime*, *anywhere* [Miller and Pascoe, 2000].

A service is characterized by not only its inherent functionalities, but also the manner in which the functionalities are provided, i.e., the Quality of Service (QoS). Based on the definition of *quality* given by International Organization for Standardization (ISO)[3], it is straightforward to derive the definition of *QoS* as "the totality of features and characteristics of a service that bear on its ability to satisfy stated or implied needs" [ISO, 2002]. In another word, QoS essentially relates to a service's characteristics that affect the service's capability to deliver its functionality and satisfy the client's needs. *QoS-aware* service discovery thus refers to the consideration of these service characteristics during the process of service discovery. As the devices, which host and consume the services, pose various challenges for service provision/consuming in ubiquitous computing environments as presented in Section II.1.2, it calls for QoS-awareness during service discovery in *USoCo* environments, which is substantiated in the following.

II.3.1 Service Location

The main functionality of a service discovery protocol is to *find* the services in the network that satisfy the client's requirements, namely *service location*.

[2] `http://en.wikipedia.org/wiki/Service_discovery`

[3] *Quality* is defined as "the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs" [ISO, 2002]

It deals with publishing or acquisition of service information, including the information necessary for service matching (of functional and non-functional properties) and usage (e.g., access information), while remaining independent of how the service is described or used. It can be *push-based*, where service providers announce their service information in the network. When a service client needs a service, it has all the service information at disposal already. Push-based service location deals with the propagation means of service information (e.g, the propagation range, the destined receivers). Service location can also be *pull-based*, when a client actively disseminates its requests in the network. It concentrates on how to acquire service information (e.g., where to propagate service requests).

Due to the featured distribution of the targeting environments, service location needs to be decentralized and independent of any fixed entity. Besides, service location needs to be QoS-aware not only in that the located services should meet the client's QoS requirements, but also in its awareness of device mobility, because inherent mobility of nodes can break the network path between the client and the server. Such breakage can fail the services when being delivered and thus degrade *service reliability*, i.e., *the probability of a service performing its purpose adequately for a period of time intended under the conditions encountered* [Reibman and Veeraraghavan, 1991].

It is therefore necessary to incorporate "mobility awareness" into service location. In infrastructured networks where nodes roam around but remain connected to the infrastructure, Mobile IP [Perkins et al., 1998] can be utilized to manage device mobility and enforce service continuation [Bhagwat et al., 1994]. Such a mechanism relies on the infrastructure and thus can not be assumed to be available in *USoCo* environments. In ad hoc networks, one alternative approach to handle mobility is to rely on routing protocols, which are in charge of maintaining and updating routing tables in face of dynamically changing network topology. However, most routing protocols, whether table-driven or on-demand, handle mobility in a reactive manner: only a link breakage can trigger the effort to find another route and update the routing table. Although there exist some efforts on finding reliable links that are unlikely to break soon (e.g., observing whether the signal strength is strong), they are either based on an oversimplified propagation model or relying on some utilities (e.g., GPS). Therefore, service location can not rely on the routing protocols to shield mobility. It needs a mobility detection mechanism, which is largely missing or weakly supported (e.g., using service advertisement expiration time) in current service discovery protocols.

We propose a mobility-aware method of locating services using signal strength and its tendency (Chapter IV). Our solution steers the service in-

formation only along paths that are not going to break soon. By doing this, it improves service reliability and decreases significantly the number of service delivery failures.

II.3.2 Service Selection

As service location possibly finds multiple service instances that satisfy the user's requirements, it calls for selecting the best one among them. For example, in Figure II.1, a service client requests a service that costs less than 20 cybeuros (abbreviated as *cb*) and takes less than 5 seconds. It finds 3 instances, all meeting its requirements and thus needs to select the best one.

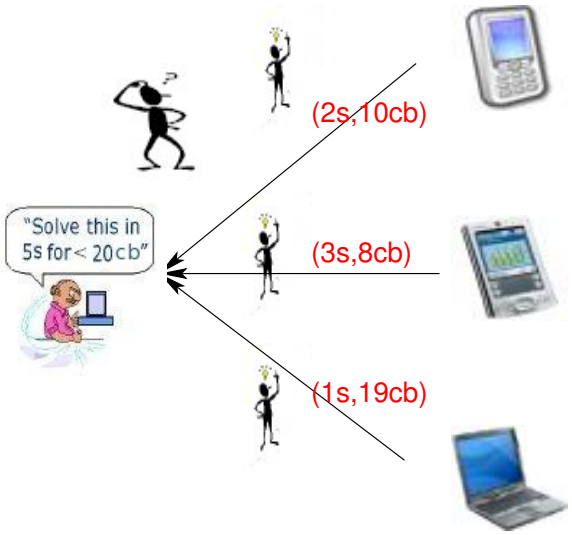


Figure II.1: Problem definition of service selection

Service selection is largely missing in current service discovery protocols [Zhu et al., 2005], which forces the client to make a choice for every selection. The selection process can be tedious and error-prone as the clients have to go through and compare all different QoS properties. Such burden on users conflicts with the unobtrusiveness of the ubiquitous computing vision [Weiser, 1991]. It thus calls for the functionality of automatic service selection on the client's behalf.

For a service client, its ultimate goal is to select the service that best satisfies its needs, which are reflected in two aspects [Hung and Li, 2003]: (1) the QoS offered by the service; (2) the price the client has to pay. While for a service provider, its goal is to maximize its revenue (i.e., the price paid by the client) at minimal overhead. Given the same level of QoS, it is desirable

to choose the one that requires the lowest overhead, i.e., consuming the least resources. This is because it achieves the system-wide goal of realizing a certain level of QoS with the least resources, which are limited and precious for portable devices. However, the client's selection is determined by QoS and service price, the latter of which is given by service providers and are subject to their strategies. Therefore, there exists a gap between system wide goal and individual interests.

We address the above issues in two steps (Chapter V): (1) proposing a utility function that evaluates services from the point of view of the client, which comprehensively integrates the factors including user's preferences, service QoS and price; (2) using Vickrey auction [Vickrey, 1961] to resolve payment to motivate truthful price revealing and thus realize pairing a service request with the most suitable service provider (i.e., the one consuming the least resources).

II.3.3 Reputation Mechanism

During service selection, trustworthiness of service providers cannot be taken for granted given the openness of the ubiquitous computing environments [Kindberg and Fox, 2002]. Therefore, a client needs to evaluate the trustworthiness of service providers, because a dishonest service provider can cheat (e.g., by exaggerating its offered QoS) for more revenues. Different from the other two factors (i.e., QoS and price) that affect service selection, trustworthiness is concerned with a service provider instead of a single service, i.e., whether a service provider delivers the QoS as it claims. Moreover, trustworthy evaluation needs to be carried out even when service selection is not necessary, i.e., there is only one located service. Because of the difference, we address service provider's trustworthiness as an individual issue, although it can be a factor, along with other QoS properties and price, affecting service selection.

Traditionally, security mechanisms such as authentication and access control (e.g., Pretty Good Privacy [Zimmermann, 1995] and X.509 [Adams and Farrell, 1999]) are used to fight against malicious parties. However, they rely on security infrastructure such as Certificate Authority, which is not suitable for ubiquitous computing environments [Cahill et al., 2003]. More importantly, since it is commonplace to interact with strangers in *USoCo* environments, even with authentication and authorization services at disposal, authenticating an unknown entity does not provide any access control information. Trust and reputation, on the other hand, can provide protection against

such threats.

Trust deals with the estimation of a node's future behavior. For example, a client trusts a service provider in that the latter *will* actually offer the QoS as claimed. Therefore, Gambetta defines trust as "*a particular level of the subjective probability with which a node assesses that another node or group of nodes will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action*" [Gambetta, 1990]. Trust is generally difficult to establish between strangers [Resnick et al., 2000], because they do not have any previous experiences and are not subject to a network of informed entities about their behaviors. Reputation, which is "perception that a node creates through past actions about its intentions and norms" [Mui et al., 2002], is important for fostering trust [Resnick et al., 2000], because it dissuades entities to misbehave resulting from no fear for future revenge. It has been proved to be a useful model and widely deployed in various scenarios [Miller et al., 2002, Grandison and Sloman, 2000, Zacharia and Maes, 2000] such as electronic market places (e.g., eBay[4]) and online communities (e.g., Slashdot[5]). The reputation assessment of a trusted node, named *trustee*, by a trusting node, named *trustor* are dependent on [Yu and Singh, 2002]: (i) the trustor's own direct experiences with the trustee; (ii) the trustor's indirect experiences, i.e., *recommendations* (also named *ratings*) from other entities. The entities who give recommendations are called *recommenders*. To prevent loops, recommendations are based solely on recommenders' own direct experiences.

Given the openness of *USoCo* environments, it is very likely that before interacting with an entity, the accumulated direct experiences are too few or too old to derive a trust decision. Therefore, recommendations are indispensable for alleviating the problem of insufficient direct experiences. Besides, recommendation facilitate interactions because they make it possible that an entity's conducts, whether good or bad, are remembered by not only the directly involved entities, but also others. Therefore, recommendations create expectation of reciprocity or retaliation due to the current behavior (so-called *shadow of the future* in [Resnick et al., 2000]). In addition, recommendations help speeding up the recognition of the trustworthiness of another entity, in contrast to having to accumulate enough direct experiences.

However, recommendations can be difficult to elicit, i.e., entities are reluctant to recommend. This is because [Miller et al., 2002]: entities may withhold positive evaluations if a seller's capacity is limited, e.g., wise parents are reluctant to reveal the names of their favorite baby-sitters; entities may be reluc-

[4]http://www.ebay.com
[5]http://www.slashdot.com

tant to give positive recommendations because they lift the reputation of the trustees, which are potential competitors; entities may be afraid of retaliation for negative feedbacks; last but not least, the (truthful) recommendations only benefit others. Moreover, recommendations are also subject to manipulation and can be false, e.g., colluders give high recommendations for each other. A false recommendation is called a *rumor*. Since truthful recommendations are critical for a reputation mechanism to operate effectively [Resnick et al., 2000], the above two issues pose obstacles for designing a reputation mechanism that is capable of recognizing the real trustworthiness of an entity.

Existing reputation mechanisms (e.g., [Buchegger and Boudec, 2002, Michiardi and Molva, 2002, Miller et al., 2002, Jurca and Faltings, 2003]) do not solve the two aforementioned problems altogether. Therefore, we propose a distributed reputation mechanism that incentivizes entities to recommend truthfully and actively (Chapter VI). Our mechanism is robust against rumors, and distinguishes (1) between trustworthy and untrustworthy service providers and (2) between honest and dishonest recommenders.

II.4 Concluding Remarks

In summary, while service discovery is essential for the realization of ubiquitous computing vision, it is also faced by the related challenges. It is thus necessary to consider QoS properties during service discovery, especially in the following three aspects: (1) during service location, it needs to have enhanced awareness of device mobility to improve service reliability; (2) during service selection, service instances need to be evaluated taking into account their QoS properties and the client's preferences; (3) reputation-based trustworthiness evaluation of an entity requires enforcement of honest and active recommending. Our work focuses on the above three aspects, as current work on service discovery does not provide adequate support regarding them.

QoS-aware service discovery lays a basis for other middleware functionalities in *USoCo* environments. For example, QoS-aware service composition (e.g., [Mokhtar et al., 2005]) assembles elementary services towards implementing more powerful and complex services that meet certain QoS requirements. It needs to discover appropriate service instances with certain QoS properties such that they meet the overall QoS requirements after composition.

Before we present our work of supporting QoS-aware service discovery, we survey the existing work on service discovery in the next chapter. Especially, we study the related work addressing the three aspects as discussed

above.

III

Service Discovery in Ubiquitous Computing Environments: State of the Art

In the previous chapter, we have identified three aspects of QoS-aware service discovery in ubiquitous computing environments that need to be addressed: (i) mobility awareness during service location, (ii) QoS awareness during service selection and (iii) reputation of a service provider. In this chapter, we first survey the existing service discovery protocols in Section III.1. Then we investigate the existing efforts addressing the above issues and analyze their suitability for *USoCo* environments. They include related work on QoS-aware service location (Section III.2), service selection (Section III.3) and reputation mechanism (Section III.4).

III.1 Service Discovery Protocols

Over the past few years, many organizations have designed and developed service discovery protocols. They include proposals from software vendors or industry standard communities, such as Service Location Protocol [Guttman et al., 1999], UPnP Simple Service Discovery Protocol (SSDP) [Goland et al., 1999] and Bluetooth SDP [Bluetooth SDP, 2004]. Proposals from academic research include Intentional Naming System (INS) [Adjie-Winoto et al., 1999], Ninja Service Discovery Service (SDS) [Hodes et al., 2002], DEAPspace [Nidd, 2001], Konark [Helal, 2002] and Group based Service Discovery (GSD) [Chakraborty et al., 2006]. Each of the above addresses a different mix of issues, but most of them do not address (well) the three issues we have identified.

Among the existing SDPs, service location protocol (SLP) [Guttman et al., 1999] is an Internet Engineering Task Force (IETF) standard for decentralized, lightweight and extensible service discovery. In SLP, there exist three kinds of entities: *User Agents* (UA) perform service discovery on behalf of the service client; *Service Agents* (SA) advertise the location and properties of services on behalf of the service provider; *Directory Agents* (DA) collect advertisements from SAs and respond to service requests from UAs. SLP can work with and without DAs. With DAs, services are registered at DA and discovery requests are answered by DAs. Otherwise, UAs send their service requests to the SLP multicast address. All SAs listening to that multicast address can unicast responses to the UA. Furthermore, SAs periodically multicast their services such that UAs can learn about their existence. Although SAs can advertise the QoS attributes of their services using a service template, SLP does not support mobility awareness and does not handle service selection or trustworthiness.

Universal Plug and Play (UPnP) is an architecture to facilitate network connectivity of intelligent appliances, wireless devices and PCs of all form factors. In UPnP, a device can dynamically join a network, obtain an IP address, respond to service discovery requests and learn about the capabilities of other devices in the surrounding environments. UPnP uses Simple Service Discovery Protocol (SSDP) for service discovery. A joining device (service provider) sends out an advertisement multicast message to advertise its services to potential clients (called *control points*). Similarly, a newly joined control point can search for devices of interest on the network. The service descriptions in UPnP can be retrieved given the URL embedded in the discovery message. Before a device and its services are removed from the network, the device multicasts a "goodbye" message to indicate so. Therefore, the mobility issue does not exist. However, neither service selection or trustworthiness is addressed.

Service discovery protocols have also been proposed for MANETs (e.g., Konark [Helal et al., 2003], GSD [Chakraborty et al., 2006], Allia [Ratsimor et al., 2002]). Helal *et al.* propose a service discovery protocol targeting ad hoc networks of mobile light-weight devices [Helal et al., 2003]. Their protocol assumes a completely distributed setting, i.e., every device acts as a service directory and stores the information about their local services and the service discovered or cached via advertisements. Service discovery in Konark can assume a *push* or *pull*-based model. A service advertisement or a service response (to service discovery request) only contains URL of the service description, where non-functional properties are stored. A client thus has to further retrieve the service description before knowing whether a service satisfies its QoS requirements. Konark does not consider service selection among

multiple qualified instances or trustworthiness issue.

Group based Service Discovery (GSD) proposed in [Chakraborty et al., 2006] also targets MANETs, where services are advertised in the vicinity (bounded by the propagation range) and cached by the recipients. GSD uses semantic service description based on Web Ontology Language (OWL) [W3C, 2004a], where every service, depending on its functionality, belongs to a service group as defined in the service ontology (e.g., DReggie Ontology [Chakraborty et al., 2001]). Thus the service advertisement of a node not only includes its services, but also the groups of the services it has "heard" (i.e., cached advertisements from its neighbors). Therefore, when a client needs a service that is not provided by its one-hop neighbors or itself, it selectively sends the request to the neighbors who have "heard" the services from the same group as the requested service, in order to improve the probability of discovering the requested service. GSD focuses on service functionality, on which the grouping and service request forwarding are. Service QoS is not considered, neither is service selection or trustworthiness.

III.2 QoS-aware Service Location

QoS-aware service location supports various QoS properties of a service when service information is being acquired (for pull based models) or published (for push based models). These properties are described in the QoS description of a service. Moreover, other factors affecting QoS need to be taken into account, particularly device mobility, which can lead to low service reliability.

III.2.1 QoS Description Awareness

Current efforts on QoS-aware service location mostly focus on Web service (e.g., [Ran, 2003, Al-Ali et al., 2003, Cardoso et al., 2004, Maximilien and Singh, 2004, Chen et al., 2003]), addressing the lack of support of non-functional properties in Universal Description, Discovery and Integration (UDDI). In general, the above solutions provide facilities (e.g., an additional broker or an enhanced UDDI registry) for registering services along with its QoS properties. Meanwhile, clients are empowered with the capability of specifying their QoS requirements in the service discovery requests. For example, in [Ran, 2003], Ran suggests extending an UDDI registration record with a data type of *qualityInformation*, used to store a service's QoS description. By enhancing UDDI registry with capability of service lookup satisfying both functional and QoS requirements, a client can carry out QoS-aware discovery of

Web services. In order to provide QoS description of a service, QoS attributes that need to be taken into account are defined (e.g., [Ran, 2003]), which cover properties ranging from runtime related ones (e.g., performance) to transaction support. The QoS attributes, however, do not consider the characteristics of thin devices regarding their limited capacity.

Besides Web service, Xu *et al.* propose a service/resource discovery in the context of global computational grids [Xu et al., 2001]. The service location is carried out along a hierarchy of discovery servers. By caching the clients' feedbacks on experienced QoS on the intermediate discovery servers, the QoS requirements from a client are taken into account during the process of service location. The feedbacks are only propagated in the *QoS-similar domains*, where a majority of clients in the domains tend to observe similar QoS from the same service provider. The hierarchical directories, however, are not available in *USoCo* environments.

In [Liu et al., 2003], Liu *et al.* propose a distributed resource discovery in MANETs using dynamically generated directories. Each directory stores resource information (e.g., QoS properties) offered by the nodes under its coverage. Directories are assumed to be synchronized such that they can estimate the path delay between each other. This delay is also used to estimate the delay between two non-directory nodes. A client c starts by sending a resource request to its home directory H, which in turn calculates the resource's hash index that maps to a list of peer directories who store the location information of the requested resource (i.e., its home directory). H first asks the peer directory with the lowest delay for the home directories of the requested resource. It then requests for QoS information from the home directories of all resource candidates. Thus QoS awareness is not substantiated until after H collects all QoS information about the resources, which can actually be carried out when H is asking for QoS information from the resource candidates.

III.2.2 Mobility awareness

Besides the service QoS, the service path also needs to be taken into account during service location. In particular, due to device mobility, the service path between service client and provider can be prone to break, leading to low service reliability. In general, routing protocols for ad hoc networks handle broken links and are supposed to shield mobility from upper layers (e.g., middleware).

III.2.2.1 Ad Hoc Routing

Many routing protocols in ad hoc networks handle mobility in a reactive manner: the rediscovering and updating of routes are triggered only after a link breaks, which is reported by the MAC protocol (e.g., the IEEE 802.11 MAC protocol [IEEE, 1999]). If a SDP simply follows the routes provided by the routing protocols, it can suffer low service reliability because of the existence of routes that tend to break soon. For more clear explanation, we give an estimation of the probability of service delivery failure. As shown in Figure III.1, assume node *A* has a communication range of R and node *B* falls in that range. Thus *A* can locate a service at *B* using the path A $\rightarrow$ B given by the routing table. Assume *B* is moving away from *A* at a speed of 1.5 m/s (human walking speed) and the time between when *B* sends back a service discovery response and the service finishes is T (including message's round trip time and service latency), the probability of service delivery failure thus equals the probability of *B* getting out of *A*'s communication range after T. Assume that *B* falls into any position in the communication range of *A* with equal probability, the service delivery failure probability is equal to the probability of *B* falling into the shaded area as shown in the figure, i.e., $(\pi \times R^2 - \pi \times (R - T \times 1.5)^2)/\pi R^2$. Given T of 15 seconds and communication range of 100 meters, the probability of service failure amounts to 40%. Note that this is just the probability for the cases when there is only one hop between service client and provider. The probability is even higher for multihop paths. Although routing protocols can try to find another path to the service provider after the current path breaks, it is possible that there is no alternative path, i.e., the service provider gets disconnected from the client. In addition, rediscovering a path requires additional time and the rediscovered path can break again before the service finishes.

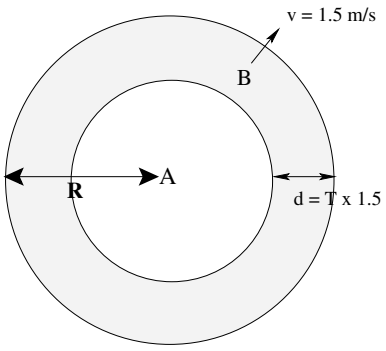


Figure III.1: Link stability with two ends moving

To address the above issue, efforts have been put on finding *reliable routes*,

i.e., routes that are unlikely to break soon. As a route is composed of one or multiple links, reliable routes require every composing link to be *stable*, i.e., with high possibility of continuing to be valid for an intended period of time. A stable link is different from a link with long lifetime in that the former has long *residual link lifetime*, which is the length of the time that a link will continue to be valid. As follows, we survey the existing work on finding stable links.

One way to estimate the residual time of a link is by measuring the distance between two ends of a link and the speeds of two nodes (e.g., [Jiang et al., 2001] and [Su et al., 2001]). Assuming that both nodes continue with their speeds and directions, it can easily estimate the residual link time, i.e., the time before the distance between two ends becomes larger than the transmission range. Both the distance and speed can be obtained via GPS, which provides geographical positions at continuing points of time and therefore can estimate a node's speed. GPS, however, suffers from poor performance in indoor environments. An approach to obtain indoor geographical positions is by radiolocation [Caffery and Stuber, 1998], with the coordination of multiple (normally three) fixed position known nodes, namely triangulation. The basic idea is that a node's position can be calculated with the distances to three given positions, named *landmarks*, (e.g., RADAR [Bahl and Padmanabhan, 2000] and APS [Niculeascu and Nath, 2001]) or with the three angles to three given positions (e.g., Angle of Arrival [Niculeascu and Nath, 2003]). For example, in Figure III.2, node *x*'s position can be fixed given its distances of *d1*, *d2* and *d3* from three fixed base stations (*B1*, *B2* and *B3*). The shortcoming of these solutions is their dependence on the coordination of multiple landmarks, which are assumed to be fixed and pre-deployed.

Alternatively, *Associativity Based Routing* [Toh, 1997] considers a link stable if its life time exceeds $A_{thresh} = 2 \times r_{tx}/v$, where r_{tx} is the transmission range and v represents the relative speed between two ends of the link. It is based on the assumption that when a link reaches a certain age, an implicit grouping of nodes can be deduced, i.e., nodes are likely to move with similar speeds and directions and will possibly stay together for a relatively long period of time. The weakness of ABR results from the difficulty of setting relative speed v.

Besides distance and speed, signal strength (SS) has also been used as a means for deriving link stability [Chin et al., 2002, Dube et al., 1997, Agarwal et al., 2000, Qin and Kunz, 2002, Goff et al., 2001, Klemm et al., 2005]. The received signal strength (SS) of a packet is generally stronger with shorter distance between the sender and the receiver, but it fluctuates greatly over a short moving distance [Rappaport, 2002]. In [Dube et al., 1997] and [Chin et al., 2002], links are considered "good" if they have strong signal strength.

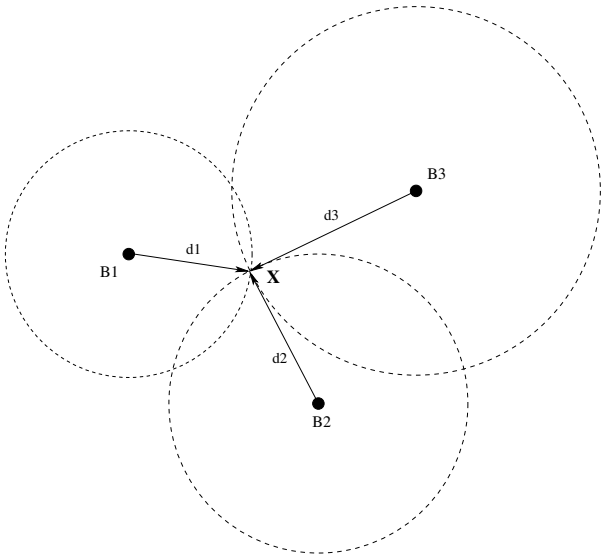


Figure III.2: Triangulation with three fixed base stations

However, such links can be unstable since it is possible that the two ends just start moving away from each other. In [Agarwal et al., 2000], Route-Lifetime based Routing Protocol (RABR) identifies stable links by estimating the residual lifetime of a link with $(SS_{thresh} - SS_{current})/\Delta SS_{ave}$, where SS_{thresh} and $SS_{current}$ are the threshold and current signal strength respectively and ΔSS_{ave} is the average changing rate of SS. Its shortcoming lies in the difficulty of measuring ΔSS_{ave}, since SS fluctuates greatly over a short distance. In addition, RABR assumes linear decreasing of signal strength with time, which hardly reflects the reality. This is because even if the signal transmission assumes the *free space* propagation model (the simplest one), the signal strength is proportional to the square of distance. If the speed is fixed, the signal strength is proportional to $1/(d - v\Delta t)^2$, the variation of which is non-linear to time t.

Preemptive routing as proposed in [Goff et al., 2001] finds other paths before a path fails, triggered by low signal strength of received packets. The threshold value is set according to a pessimistic estimation, which enables a communication task to be completed even if two nodes move away with the maximum speed. The weakness of their approach lies in the assumption that SS behaves strictly according to the Two-Ray Ground Model, similar to [Qin and Kunz, 2002]. This model is also used by [Klemm et al., 2005] for predicting distances in order to determine whether packet loss is due to node mobility or congestion for the purpose of improving TCP performance over wireless networks. However, Two-Ray Ground model works well only for predicting distances of several kilometers for cellular telephony systems and is not suit-

able for MANETs [Kotz et al., 2003].

III.2.2.2 Mobility Aware Service Location

Current service discovery protocols put very few efforts on dealing with mobility. One possible way to handle the mobility issue is to voluntarily announce a device's departure when it leaves the network (e.g., UPnP SSDP [Goland et al., 1999]). Such an approach is suitable for the scenarios where a user intends to remove a device from a network. For a mobile device that unconsciously disconnects from the network, it is complicated to arrange such an announcement because it is difficult to estimate when a device actually gets disconnected from the network.

Some SDPs maintain the status of service availability as a *soft state* [Raman and McCanne, 1999], such as UPnP SSDP [Goland et al., 1999], Konark [Helal et al., 2003], Ninja [Hodes et al., 2002], DEAPspace [Nidd, 2001]. The idea is that a service provider periodically transmits to potential clients "refresh messages", which include *time to live* (TTL), the time a service is expected to remain available. The recipient clients keep a service advertisement until its TTL expires. Lack of refresh message after expiration of TTL implies unavailability of the service. However, being available cannot guarantee successful service delivery since services can fail when being delivered. Therefore, usage of TTL remains a coarse-grained approach to handle dynamics of service availability. It is more suitable for relatively stable environments where mobility of service providers is less considerable during service delivery.

III.3 Service Selection

Service selection evaluates the located service instances on behalf of a client, according to the latter's preferences among different properties. For example, a client can prefer services with the smallest latency or the lowest price. Although both service location and selection are QoS-aware, they are very much different in that the former finds services that satisfy the QoS requirements posed by a client, while the latter selects among those instances returned by the former, taking into account the service's QoS properties. For example, besides the requirements in terms of latency and price, client A prefers to have a service latency as small as possible while client B may go for low prices. Thus a service with small latency and high price is of greater value to A than B, even if it meets the requirements of both A and B. Note that it can be argued

that a client can simply choose the first arrived service reply and invoke it immediately. It is simple and does not require any waiting time for other replies to arrive. However, this approach does not necessarily save time for the client since the first reply can have long service latency (at the service provider). More importantly, it excludes the possibility of finding instances that better suit the client's needs and match its preferences than the first reply.

III.3.1 Service Evaluation

As all the located service instances provide the functionality required by the client, we focus on QoS-based service evaluation. Existing work generally evaluates a service from three different aspects: (1) evaluation based on QoS description of services; (2) evaluation based on QoS properties of service paths; (3) evaluation based on resource availability of service providers. They are detailed respectively as follows.

III.3.1.1 Evaluation based on QoS Description

Similar to a client's QoS-based service evaluation to select the best one, a resource management system evaluates services in order to make decisions on shared resource allocation. A service can have different QoS and thus different utility (e.g., degree of user's satisfaction) depending on the amount of available resources (e.g., bandwidth). The resource management system then takes into account each service's utility and cost (in terms of resource consumption) for the purpose of, for example, achieving maximum overall utility [Liao and Campbell, 2001, Geihs, 2002]. Similarly, adaptation decisions on behalf of applications can be automatically made (e.g., Odyssey [Noble and Satyanarayanan, 1999]), depending on the utilities of different QoS levels.

In [Venkatasubramanian and Nahrstedt, 1997], the authors propose a metric to measure the efficiency of video transmission (Equation III.1), which is essentially the ratio of user satisfaction (US) and resource consumption (RC). User satisfaction is determined by various QoS parameters such as end to end delay, synchronization skew; resource consumption refers to the maximum consumption (in terms of ratio) of various resources. This metric evaluates the cost-efficiency of a video transmission and can be useful for resource management in multimedia systems.

$$M \propto \frac{US}{RC} \tag{III.1}$$

The above evaluation bears great similarities to a client's service evaluation in that they both take into account the overall QoS brought to the user (i.e., user satisfaction) and the introduced overhead. However, the above solution is more suitable for cooperative environments where a service selection decision is made to the interests of both the service client (the overall QoS) and the provider (the resource consumption). In *USoCo* environments, a client is concerned with the service price to be paid, instead of the resource consumption on the service provider. The difference in concerns leads to different evaluation results and thus different selection decisions.

In [Lee et al., 2004], the authors propose an approach to select among Internet service providers in ubiquitous computing environments with pervasive network connectivity. They present a "personal router" that dynamically chooses among available network connectivity alternatives and reevaluates with the user moving and network services dynamically appearing, disappearing and changing. Each service instance is evaluated with its QoS profile (e.g., bucket profile for describing the network service in short and long term characteristics) and service cost containing two price attributes, price per minute and per kilobyte. Factors of QoS, price and user preference between quality and cost are integrated in a utility function to assess a service. A user interface is further provided to collect user feedback regarding the current service. For example, if a user is dissatisfied with the current service's cost and requesting a lower cost service, the weight of price is raised in the utility function. Their approach to select services is specific for network service and mainly aims to facilitate handoff between wireless networks, reflected in their simple attributes in the QoS profile and lack of concrete means for measuring the overall QoS.

Liu *et al.* present a QoS-aware Web service framework where a centralized registry evaluates service instances in terms of their QoS, price and reputation [Liu et al., 2004]. The evaluation incorporates different dimensions and covers user preference, such as being price-sensitive or QoS-sensitive. Normalization skills are employed to compute the overall service QoS for the purpose of comparison.

Besides QoS description, services can also be evaluated based on service path, which has impact on the experienced QoS of a client, as explained as follows.

III.3.1.2 Evaluation based on Service Path

A common attribute of a service path is *number of hops*, which is widely used by routing protocols as a criteria for finding the best route. Service path plays an important role during server selection in Internet cache architecture, where a client needs to select among multiple replication servers to achieve best performance or for the purpose of load balancing. For example, in [Dykes et al., 2000], different strategies such as bandwidth-based, latency-based, hybrid of both, are explored and compared. The metrics such as bandwidth or latency can be obtained based on either historical data (e.g., the previous experienced latency) or probing. The authors further conclude that probing generally outperforms statistical estimation based on historical data.

In [Varshavsky et al., 2005], service provider is chosen based on lowest hop count in order to localize communication. The selection is done by allocating services to nearby providers such that the traffic of service provision is not spread all over the network. However, the hop count alone is not a reliable indicator for the best path. As pointed out in [Couto et al., 2002], many routes with minimum hop count have poor throughput because of the existence of low-quality links with high retransmission rate. A more appropriate approach is through dynamic probing (e.g., [Gao and Steenkiste, 2002]), i.e., given a set of candidate service providers, a client sends a probe message to all of them and the one with the shortest response time is selected. However, such probing-based selection incurs more traffic overhead and waiting time.

III.3.1.3 Evaluation based on Service Provider

Services can also be evaluated based on the resource availability of their hosts. For example, in [Fei et al., 1998], in order to achieve load balancing, the least loaded server is selected among replicated services, which are services providing identical content or functionality using multiple servers in the network. Similarly, HTTP-redirect [Fielding et al., 1999] forwards a client's HTTP request from a more loaded server to a less loaded one to realize load balancing. In Intentional Naming System (INS) [Adjie-Winoto et al., 1999], when a set of services meet a client's requirements, the service is selected according to service providers' properties such as current load. In [Lee and Helal, 2003], the authors propose a service selection logic which considers attributes such as (network) distance to the service provider and load of the service provider. A noticeable feature is that such selection is hidden from clients.

The above selection implicitly improves QoS for the client, since less loaded servers generally offer better QoS. Especially, in *USoCo* environments where

services can be hosted on devices with limited resources, it is necessary to incorporate such factors into service evaluation. However, the above surveyed work largely implements service selection for the purpose of load balancing. They thus mainly consider the workload of a service provider and ignore other factors such as battery, memory, bandwidth, which are also limited and affect the QoS offered by the service provider. More importantly, the above work lacks a means to capture a client's preference among various different QoS properties in a comprehensive manner.

From a different angle, Day and Deters address the problem of how to select among syntactically identical Web services in [Day and Deters, 2004]. The selection is carried out within two steps: (1) adding semantics to QoS parameters; (2) adding rule-based reasoner based on historical QoS information acquired from a centralized QoS information storage (so-called QoS forum). For example, the reasoner evaluates a service instance based on all other clients' previous experiences with it and carries out evaluation based on aggregate QoS values (e.g., average latency). Such an approach assumes that a service instance's (advertisement) QoS values remain unchanged such that previous QoS experiences correspond to the same promised QoS from the service provider. It may be suitable for relatively static and stable environments, but falls short in *USoCo* environments, where thin service providers tend to adjust their QoS depending on the varying availability of resources.

Vu *et al.* propose a Web service selection method based on services' QoS feedbacks [Vu et al., 2005]. The truthfulness of the feedbacks is determined by their similarity with those of some pre-existing trusted QoS monitoring agents. Accordingly, the honesty of a reporter, who provides QoS feedbacks, is determined by that of its QoS reports. The services are then evaluated with respect to not only the predicted QoS based on honest QoS feedbacks, but also the semantic similarity between the offered functionality and the requested one. Their work targets Web services on the Internet, which can thus rely on pretrusted monitoring entities that offer honest QoS reports. A reporter is evaluated regarding its honesty in giving feedbacks regarding each QoS dimension, which does not seem necessary given that it is unlikely to have different strategies of giving feedbacks on different QoS dimensions. Moreover, the motivation of providing honest QoS feedbacks is not addressed. The work is highly related with the issue of incorporating recommendations in the reputation mechanism, to be discussed in Section III.4.

Besides service QoS, another factor affecting service selection is service price, which is determined by the pricing model.

III.3.2 Pricing Model

The pricing model can be static and simple. For example, telephone price is flat (static) and does not adapt to variation in demand and user requirement. Its main advantage is its simplicity such that service providers and clients have high predictability regarding the cost and revenue [Hille et al., 2000]. This approach, however, does not distinguish between services of different QoS and service providers of different capacity and resource availability. Therefore, it is inappropriate and insufficient for the client's service selection. The pricing model can be dynamic and service prices are adjusted with different conditions (e.g., load and capacity) of service providers. It is useful for reflecting capacity heterogeneity and changing resources of devices in our target environments.

III.3.2.1 Service Price

Service pricing mechanism has been under intensive discussion in the 1990s, when Internet became popular and charging mechanisms were proposed mainly to offer different QoS classes of Internet services and realize differentiated traffic control. One example is Paris Metro pricing mechanism in [Odlyzko, 1999]. The basic idea comes from the old Paris Metro System, where first and second class cars are identical, except that first class tickets cost twice as much as the second class ones. Because of the price difference, first class cars are generally less congested. But if they become too crowded, some people decide they are not worth the extra cost and change for the second class, reducing congestion in the first class and thus restoring the QoS difference between the two classes. Therefore, channels of different prices handle packets differently. A user selects a channel based on its budget, its QoS requirements and QoS feedback from other users. A distinguished feature of this pricing mechanism is that it is self-regulating: by setting two different prices, the QoS difference between the two classes are automatically maintained. The smart market mechanism [MacKie-Mason and Varian, 1994] charges users based on the congestion they create. Each packet is marked with a bid, stating its priority. Users are charged with the bid of the highest priority packet that is not routed. Both of the above mechanisms use price to regulate traffic, which can be utilized in the service selection to direct service demands.

Given the same level of QoS provided by service providers, service selection should favor the service with the lowest overhead, i.e., consuming the least resources. This is because it achieves realization of the best QoS at the lowest cost. If a client simply selects the cheapest service and pays the low-

est price, it does not necessarily select the one with the lowest overhead. This is because service providers can ask for prices higher than the real overhead of providing the service. Driven for more revenue, a service provider tends to increase its asking price, which can lead to a suboptimal service selection result. For example, assume both service providers A and B offer a service s with the same QoS, with overhead of O_a and O_b respectively ($O_a < O_b$). Ideally, A should be selected, as it is the most effective choice. However, it is possible that, due to different strategies of increasing revenues (e.g., A is more aggressive than B), A asks for a price higher than B, which leads to the selection of B. Therefore, it requires a pricing mechanism to incentivize the providers to reveal truthful prices, which can be realized using mechanism design, as explored as follows.

III.3.2.2 Auction-based Pricing Model

The field of mechanism design studies how to design systems so that entities' selfish behavior results in desired system-wide goals. The designed mechanism is called an *incentive scheme*. However, the game-theory literature on mechanism design neglects computational and communication complexity, which makes mechanism-design approach unpractical in many settings. This is addressed by distributed algorithmic mechanism design (DAMD).

In essence, *Distributed Algorithmic Mechanism Design* (DAMD) [Feigenbaum and Shenker, 2002] addresses the design of *incentive compatible* mechanisms (i.e., mechanisms that result in desired system-wide outcome from selfish behavior of individuals) at *tractable computational and communication expense*. It is an extension of algorithmic mechanism design [Nisan and Ronen, 2001], which designs incentive compatible mechanisms with tractable computation. DAMD lies in the intersection of economics science and computer science. More formally, consider a distributed system in which there is a set of possible outcomes $\mathcal{O}$ (e.g., result of service selection)[1]. For convenience, "agent" is used to refer to a software entity representing and working for the interest of a node. A *strategy* for an agent is a complete contingency plan, i.e., a plan describing what decision the agent should make under each possible situation that might occur. Each of the n autonomous strategic agents[2] has a utility function u_i: $\mathcal{O} \rightarrow \mathcal{R}$, where $u_i \in \mathcal{U}$ ($\mathcal{U}$ defines the set of utility functions of agents) and expresses an agent's preferences over these outcomes. The desired system-wide goals are specified by a *Social Choice Function* (SCF)

[1] We rely mostly on [Feigenbaum and Shenker, 2002] for definitions and notations.

[2] Since even random behavior can be considered as one kind of strategy, every agent is a strategic agent, strictly speaking.

$F : \mathcal{U}^n \rightarrow \mathcal{O}$ that maps the (actual) utility functions of agents to a particular outcome. However, each agent is usually reluctant to publicize its actual utility function, making it difficult to achieve any global goal.

For a given mechanism *M*, let *S* denote the *strategy space* of one agent, i.e., a set of strategies that can be taken by the agent, and $C_M(u) \subseteq S^n$ denotes all possible strategy vectors that could reasonably result from selfish behavior. The goal of mechanism design is to define a mechanism *M* that implements the SCF, i.e., $M(C_M(U)) = F(U)$, for all $U \in \mathcal{U}^n$. With such an *incentive compatible* mechanism, selfish behavior by agents results in desired system-wide outcomes.

In game theory, the strategy that is always to the best interest of one agent, no matter how other agents act, is named *dominant strategy*. A mechanism with dominant strategy is very desirable for scenarios featuring interactions among autonomous, automated agents, compared to those without dominant strategy: (1) the behavior of an agent is much simpler: it only needs to follow the dominant strategy regardless of other agents' behavior; (2) it saves the complex knowledge representation and logic evaluation for counterspeculating how other agents will behave. Thus, it is very desirable to have an incentive compatible mechanism with dominant strategy.

DAMD relates to designing a mechanism such that the dominant strategy for each agent is to reveal its true valuation, which is leveraged to achieve SCF. DAMD is suited for *USoCo* environments because (1) it is distributed; (2) the thin devices can only afford tractable computational and communication complexity. Additionally, the computation and communication overhead of a mechanism for service selection should be even *lighter* than "being algorithmic"; the overhead should be as small as possible. As the emphasis of mechanism design is put on the implementation of various types of auctions [Nisan and Ronen, 2001], which has been an efficient means for resource allocation [McMillan, 1994], service assignment [Vulkan and Jennings, 2000] and conflict resolution [Capra et al., 2003].

According to the definition of McAfee and McMillan [McAfee and McMillan, 1987], an auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants. An auction can be used to determine the value of a commodity that has undetermined or variable price. Sometimes, there is a minimum or *reserve price*. If the bidding does not reach the minimum, the item is not sold. In an auction, there exist a seller, an auctioneer and bidders (i.e., potential buyers). A seller hires and entrusts an auctioneer to host the auction and sell a commodity. The bidders offer their bids for the commodity and the auc-

tioneer then determines the winner (the highest bidder) and the commodity is sold.

The commonly seen auctions in the real world include English auction, first-price sealed-bid auction, Dutch auction and Vickrey auction.

- In *English auction*, the auctioneer starts with the reserve price and proceeds to solicit successively higher bids from the bidders until no one raises the bid. The highest bidder is the winner and pays the price she bids. The dominant strategy for one agent in English auction is to continuously raise its bid until it wins or the bid reaches the maximum price it is willing to pay for that item. A noticeable feature of English auction is that it is usually multi-round and the time and communication overhead is proportional to the difference between the starting price and the price at which the item is sold. However, it does allocate the item to the bidder who has the highest valuation of the item and is the only bidder willing to outbid all other bidders.

- In *first-price sealed auction*, each bidder submits one bid in ignorance of all other bids to the auctioneer, who determines the highest bid and sells the item to that bidder for its bidding price. This kind of auction can be executed in one-round and thus is communication-saving. However, since each agent's bid is based on her private valuation *and* prior beliefs of others' valuations, the item is not always awarded to the party who values it most.

- In *Dutch auction*, bidding starts at an extremely high price and is progressively lowered until a buyer claims an item by calling "mine". The winner pays the price at the current price. Dutch auction preserves maximal privacy: only the highest bid is revealed. However, like English Auction, it is multi-round, and like first-price, sealed-bid auction, one agent's bid is strategically based on its private valuations and its beliefs of others' valuation.

- Similar to the first-price sealed auction, Vickrey auction is sealed and executed in one-round. The highest bidder is the winner, but pays the price that is equal to the second-highest bid [Vickrey, 1961]. A distinguished feature of Vickrey auction is that the dominant strategy for every bidder is to bid her true valuation. Thus Vickrey auction always rewards the item to the bidder who values it most, i.e., realizes SCF.

Table III.1 lists the features of the above four auctions. It can be seen that only Vickrey and English auctions have dominant strategy and realize SCF.

Auction Type	Communication Complexity	Existence of Dominant Strategy	Optimal Item Allocation
English Auction	Multi-Round	Yes	Yes
First-price, Sealed Auction	One-Round	No	No
Dutch Auction	Multi-Round	No	No
Vickrey Auction	One-Round	Yes	Yes

Table III.1: Various Auction Settings

Furthermore, Vickrey auction only requires single-round execution. Thus, from the perspective of both existence of dominant strategy and communication overhead, Vickrey auction distinguishes itself as the best choice. It is therefore exploited to select the best service instance in our work.

III.4 Reputation Mechanism

Reputation mechanism has been widely used and deployed in online service provision (e.g., ebay[3]), wide-area wireless services [Chakravorty et al., 2005], peer-to-peer systems (e.g., [Aberer and Despotovic, 2001, Kamvar et al., 2003, Xiong and Liu, 2004]) and mobile ad hoc networking (e.g., [Michiardi and Molva, 2002]). During online service provision, especially e-commerce, it is commonplace for parties that are unknown to each other to encounter [Resnick et al., 2000, Josang et al., 2005]. This opens up an issue of lack of trust between two parties before an interaction takes place. P2P file sharing networks (e.g., Gnutella[4]) have many advantages (such as improved scalability) over traditional client-server approaches to data distribution. They are, however, subject to attacks from anonymous malicious peers, such as virus spreading and fake file attack [Kamvar et al., 2003]. In [Aberer and Despotovic, 2001], the authors address the issue of the recommendations' trustworthiness and storage in P2P systems at the same time. An entity files complaints against another if it detects cheating or it attempts to cheat. Such complaints are stored according to the key corresponding to the concerned entities. An entity is considered dishonest if the total of its filed complaints and the complaints against it exceeds considerably the general average. In mobile ad hoc networks, nodes rely on the service of "packet forwarding" provided by their

[3] `http://www.ebay.com`
[4] `http://www.gnutella.com`

neighbors in order to communicate with others that are out of their communication range. Reputation is thus used to evaluate a node's degree of being cooperative. It makes it possible to identify and exclude the misbehaving nodes from the routes and punish them by refusing to forward packets for them. In order to detect misbehavior, each node is assumed to operate in a promiscuous mode (e.g.,[Buchegger and Boudec, 2002, Marti et al., 2000]), such that it can listen to every packet transmitted by its neighbors even if the packet is not intended for it. When a node asks one neighbor to forward a packet, it can monitor whether the packet is actually forwarded as expected. Neighbors that are observed to be often dropping packets are singled out and excluded from any route. Some reputation mechanisms for packet forwarding on MANETs also incorporate reputation propagation (e.g., [He et al., 2004]). However, the acting of recommending is assumed to be voluntary and thus no incentive is provided. It is quite self-contradictory since the act of recommending requires packet sending. In the following texts, we survey existing reputation mechanisms, especially focusing on the handling of recommendations (or *ratings*).

Some reputation mechanisms do not distinguish between reputation of providing a service and providing a recommendation (e.g., [Zacharia and Maes, 2000, Kamvar et al., 2003, He et al., 2004]). They assume that trust on an entity's capability to provide services can be transferred to its opinions. For example, in [Kamvar et al., 2003], a peer that provides authentic files in a P2P file sharing system is trusted to give honest opinions. But such assumption can make the reputation mechanism vulnerable to reputation manipulation. For example, a good service provider can exploit it to demote the reputation of its competitors, as its opinions are considered as truthful as its services. Therefore, it necessitates the differentiation of reputation for providing services and recommendations, namely *service reputation* (SRep) and *recommendation reputation* (RRep) respectively. A trustor can evaluate the trustee's *overall reputation* (ORep) based on *SRep* and others' recommendations. The latter is taken into account depending on the recommenders' *RRep*s. For example, in [Abdul-Rahman and Hailes, 2000], *recommendation trust* is introduced to evaluate the credit of a rating.

In [Buchegger and Boudec, 2002], only negative recommendations are propagated since it is assumed that maliciousness is the exception rather than the norm, such that an entity without any negative experience is considered honest. This is an optimistic hypothesis because it assumes that any negative experience is well published and known. In [Michiardi and Molva, 2002], only positive recommendations are allowed to prevent the attack of *Denial of Service* (DoS), i.e., malicious nodes spread negative ratings such that the victim is considered dishonest and deprived of any service (e.g., packet forwarding).

The DoS attack can be handled by improving robustness to false accusations, such that the latter is identified and ignored. In addition, similar to [Buchegger and Boudec, 2002], it assumes that positive recommendations are well propagated. Hence, neither of the above two solutions preferring only positive or negative ratings is well-grounded. Recommendations can be positive or negative and should be equally taken into account for reputation evaluation.

It is possible that a recommendation does not correspond to the fact, i.e., it can be either false praise or accusation. Such recommendation is named a *rumor* (or a *lie*). Due to the existence of rumors, recommendations need to be carefully incorporated towards the trust decision of whether to interact with a service provider. In another word, the reputation mechanism needs to be able to handle rumors such that it is robust against them.

Yu and Singh [Yu and Singh, 2002, Yu and Singh, 2003] present a reputation model that aims to detect rumors in multiple agent systems. The recommendations are compared against the new direct experience to evaluate the recommenders' *RRep*s, which determine the credibility of their recommendations. Only recommendations from helpful nodes (i.e., with high *RRep*) are accepted and weighed according to their *RRep*s. Similarly, in [Huynh et al., 2005], a new direct experience is compared against the recommendations for evaluating the recommenders' credibility. Although both of the above solutions are capable of identifying and ignoring rumors, they give no penalty to either liars or free-riders, which can always benefit from others' recommendations.

In [Buchegger and Boudec, 2003, Buchegger and Boudec, 2004, Buchegger and Boudec, 2005], recommendations are utilized only as an additional source of information for deriving reputation. A trustor already has its own opinion (*SRep*) regarding the trustee before asking for any recommendation. Only recommendations that are similar enough to its own opinion are considered truthful and integrated and each accepted recommendation is given a small constant weight. An entity does not keep others' reputation in recommending. Moreover, the authors argue that liars should not be punished as it would discourage honest reporting of misbehavior. But, no incentive is given to encourage recommendation provisioning.

In [Whitby et al., 2004], all recommendations are aggregated to derive the *public opinion*. Each individual recommendation is then compared against the public opinion; too much deviation leads the recommendation to be considered false and thus excluded. The public opinion is then recalculated and compared against each remaining recommendation until no recommendation is

filtered out. This kind of approaches to identify rumors and assign weights to different recommendations is *endogenous* since the truthfulness of recommendations is judged depending on the recommendations themselves. In contrast, *exogenous* approaches use external factors, such as *RRep*, for doing so. The implicit assumption underlying endogenous approaches is that the majority of recommendations are honest such that they dominate the rumors. Therefore, a recommendation that deviates from the majority is considered a rumor. This assumption is not solid in open environments where recommendations can be very few in number, most of which can be rumors. A variant of *endogenous* approach is used in [Patel et al., 2005], where each entity records all the ratings and subsequent interaction experiences. Assume node *a* receives a recommendation from recommender *r*, *a* first picks out all the entities whom *r* has recommended with a similar value (e.g., within the range [a..b]). The accumulated experiences with those entities are calculated and compared against the rating range to obtain *r*'s *RRep*. Their approach is *exogenous*, because it is the accumulated direct experiences that are used to determine the trustworthiness of a recommendation. Meanwhile, it is also *endogenous* because such comparison is done only within the range of recommendation values that are considered relevant.

Jurca and Faltings [Jurca and Faltings, 2003] propose an incentive-compatible reputation mechanism to deal with inactivity and rumors. A client buys a recommendation about a service provider from special brokers named *R-nodes*. After interacting with the provider, the client can sell its feedback to the same R-node, but gets paid only if its report coincides with the next client's report about the same service provider. One issue is that if the recommendation from an R-node is negative such that a client decides to avoid the service provider, the client will not have any feedback to sell. Or in the existence of opportunistic service providers that, for example, behave and misbehave alternatively, an honest feedback does not ensure payback. This opens up the possibility of an honest entity to have negative revenue and thus is unable to buy any recommendation. Besides, the effectiveness of their work depends largely on the integrity of R-nodes, which are assumed to be trusted *a priori*.

In summary, although current reputation systems are capable of identifying rumors, they lack measures to enforce voluntary and honest recommendations. Therefore, they are not incentive-compatible, i.e., there does not exist any incentive for entities to actively provide honest recommendations. As there is no deterrent for liars, rumors can be rampant and honest recommendations can become difficult to acquire due to lack of motivation. Therefore, a reputation mechanism for ubiquitous computing environments not only needs to be robust against rumors, but also needs to enforce both active and

honest recommendations.

III.5 Concluding Remarks

Above has been surveyed the existing work on service discovery, especially service location, service selection and reputation mechanism. Generally, current work on service location does not provide enough awareness of device mobility, leading to low service reliability; current SDPs lack service selection that chooses the best service on behalf of the client, taking into account various QoS properties and price; current reputation mechanism that evaluates the trustworthiness of entities does not address the incentive issues of recommenders, who are not motivated to recommend honestly and actively. This leads us to devise three solutions addressing the above three issues respectively, which start with service location enhanced with mobility awareness, presented in the next chapter.

IV

Signal Strength based Service Location

Service location in ubiquitous computing environments is a challenging task: device mobility can lead to low service reliability if not taken good care of. Routing protocols which are in charge of updating and maintaining routing tables do not provide enough support for handling mobility. It thus opens up an issue that the services located by most service discovery protocols tend to fail, given the mobility exhibited in *USoCo* environments.

In this chapter, we propose a simple, yet efficient way to locate services using signal strength (SS) tendency. Our main contribution lies in the enhancement of mobility awareness during service location, which improves service reliability. In the rest of this chapter, we first study the background on signal propagation in Section IV.1, where received signal strength between moving nodes is explored and observed. Section IV.2 presents our signal strength based service location, which is followed by Section IV.3 that presents the performance evaluation of our proposal for service location, especially in terms of service reliability. Finally, this chapter finishes with concluding remarks.

IV.1 Background on Signal Propagation

Wireless communication takes place in the form of electromagnetic wave. It is different from guided media such as copper wire in that it is more susceptible to interference. In order to grasp the stability of a wireless link, we take a look at the principal mechanism of signal propagation.

When a signal is being received, the antenna senses the radio electromag-

netic waves, which cause electrons to flow in the conductor and thus create a current. To make that happen, radio waves have to carry along enough power. After the signals are received, they are further demodulated into digital bits by the receiver. Generally, the more information the signals carry, the more sensitive they are to noise. Following are some key concepts in the signal propagation.

Transmitted power is the strength of the signal emissions measured in Watts (or dBm[1]). Higher transmitted power helps to emit signals stronger than the interference, but at the cost of draining the battery faster. Too strong transmitted power also increases the interference between adjacent networks and thus decreases the possibility of *frequency reuse*. In order to make the unlicensed bands to be practically useful, regulation authorities in many countries pose a limitation on the maximum transmission power. For example, in Europe, ETSI regulates the maximal transmission power as 30 dBm. A Lucent Silver WaveLAN card has a transmitted power of 15 dBm (i.e., 31 milliwatts).

Receiver sensitivity is the power of the weakest signal that can be reliably detected and demodulated by a receiver. It is a benchmark of the performance of a receiver. Generally, receivers have weaker sensitivity at higher transmission rates. For example, a Lucent Silver WaveLAN Card has a sensitivity of -82 dBm at 11Mbps, and -87 dBm at 5.5Mbps.

The receiver sensitivity is not the only performance indicator for the receiver, because the received signal also includes noises from all other parties that share the same band, e.g., Bluetooth devices, microwave ovens, and other IEEE 802.11b wireless networks. Thus, *Signal to Noise Ratio* (SNR) is introduced and defined as the difference of received power between signal and noise. SNR is in the unit of decibel (dB) and is calculated as follows (with unit included in square brackets):

$$\begin{aligned} SNR[dB] &= 10 * \log_{10} [\text{(Signal Power [W])/(Noise Power [W])}] \\ &= \text{Signal Power [dBm]} - \text{Noise Power [dBm]} \end{aligned}$$

Intuitively, to be able to receive and demodulate the received signal successfully, the receiver requires a minimum SNR such that the signal is not overly "polluted" by the noise. Like receiver sensitivity, the threshold SNR varies with different transmission rates. For example, a Lucent Silver WaveLAN card has a SNR threshold of 16 dB at 11 Mbps, and 11 dB at 5.5 Mbps. Therefore, wireless communication succeeds only if the received signals satisfy both the

[1] $dBm = log(Watt) * 10 + 30$

receiver sensitivity and the SNR threshold. If the noise level is low, the communication is limited by the receiver sensitivity; while if the noise level is high, SNR becomes the bottleneck. Both SS and SNR of a received packet can be obtained from the wireless card driver[2]. In addition, a wireless link is possibly asymmetric, with signal quality from two ends of a link probably very much different due to difference in transmission power, interference extent and other factors. It leads to the possibility that one node can successfully send packets to the other but cannot receive any.

The radio wave (signal) propagation from the transmitter to the receiver is generally modeled by the combination of *large scale* and *small scale* propagation models [Rappaport, 2002]. Node movement over short distances (e.g., a few wavelengths) may cause the received signal strength, which is the sum of contributions of multiple components, to fluctuate rapidly, giving rise to small scale fading. As a node moves over longer distance, the local signal strength average (i.e., average of signal strength measured over a distance from 5λ to 40λ – about from 0.6m to 5m for 802.11b) gradually decreases. We call the distance *locality distance* and large scale propagation model essentially estimates the average signal strength over the locality distance. As shown in Figure IV.1 [Rappaport, 2002], the signal strength fluctuates in a large range with the increasing of the distance between transmitter and receiver, but the average signal strength over locality distance exhibits the tendency of decreasing.

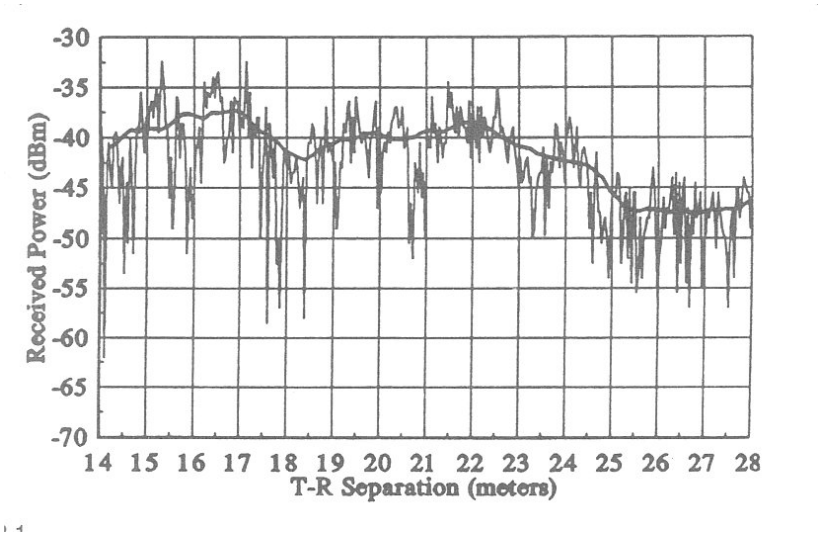


Figure IV.1: Signal Strength with different T-R distances

[2]e.g., `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Orinoco.html`

It is desirable to observe large scale propagation because we aim to capture the changing distance and thus the mobility between nodes. The most basic large-scale model of radio wave propagation is called *free space* radio wave propagation. In this model, radio waves emanate from a source point of radio energy, traveling in all directions in a straight line, filling the entire spherical volume of space with radio energy that varies in strength with $1/d^2$ rule (d is the distance from the source point). However, in the real world, signal propagation is often subject to reflection, diffraction and scattering [Rappaport, 2002]. The fading over a small range, or *small scale fading*, makes the SS collected at one point of time vary in a large range, giving no clear indication of link quality. This makes it difficult to recognize a node's moving tendency. To capture the effect of large scale propagation in spite of interference from small scale one, studying the tendency of the average signal strength over a locality distance poses as a good way to detect node mobility.

To verify the phenomenon shown in Figure IV.1, we measure the SS between two laptops equipped with Lucent WaveLAN IEEE Silver cards, both of which implement `IEEE 802.11b` standard and have omni-directional antennas. The two laptops are running on Linux (RedHat) with 2.4.26 kernel and connecting with each other via a wireless link. The experiments are all carried out indoors in the day. The measurement of signal strength was performed every 1 second by consecutively sending "HELLO" messages and measuring the signal strength from the returned replies.

In one experiment setting, one laptop moves away from the other at human walking speed for a period of 50 seconds (Figure IV.2), during which the link is connected. It can be observed that the average signal strength over locality distance (as shown with dotted lines) clearly drops with longer distance, in spite of large variation in a short distance.

In order to see whether the above can also be observed in scenarios when both nodes are moving, we also conduct experiments with both nodes moving in the same direction, but with different speeds. The experiments last 40 seconds, during which the link is connected. The signal strength and the average SS of "HELLO" messages are shown in Figure IV.3. It can also be seen that, with the distance between two nodes getting larger, the average SS (as shown with dotted lines) is also decreasing. The above two experiments have also been carried out with the presence of interference from another ad hoc network, which leads to similar results.

The above experiments observe and verify that node movement can be detected by studying the variation tendency of average received SS over the locality distance. This fact can be exploited to identify stable links and thus

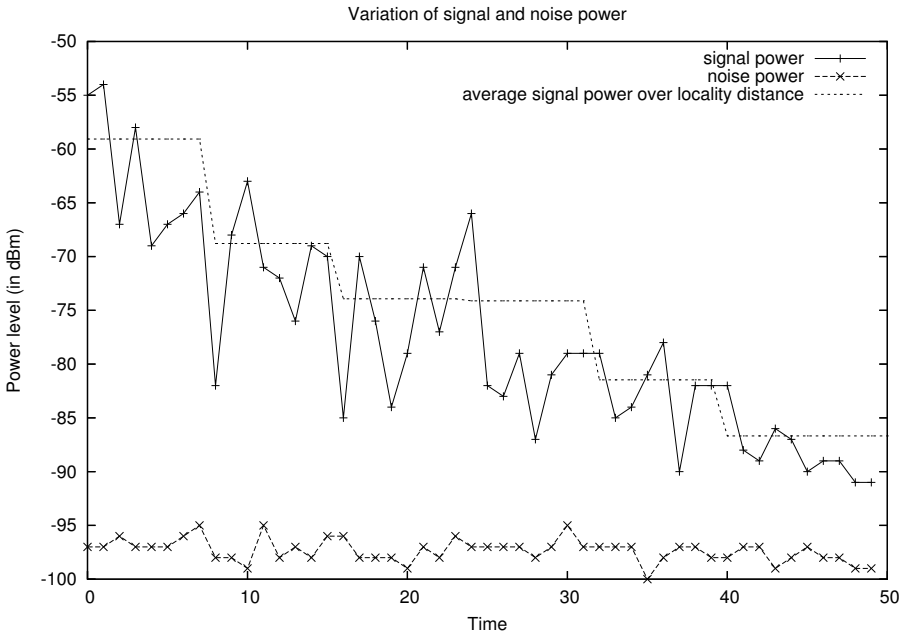


Figure IV.2: Signal and noise power when one node is moving

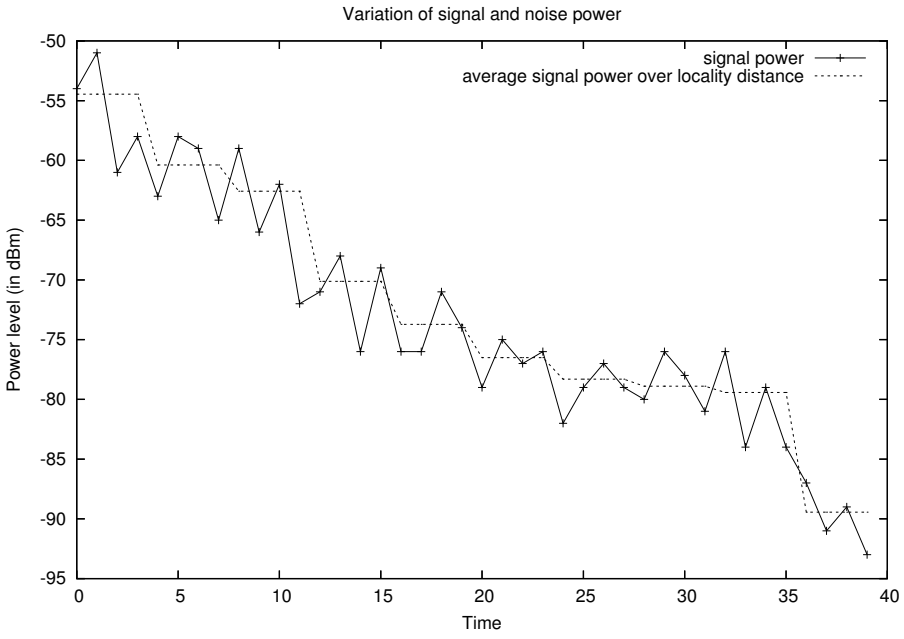


Figure IV.3: Signal and noise power when both nodes are moving

to locate reliable services. Note that different from other approaches that use

signal strength as the criteria of a good link, we use SS tendency to detect mobility.

The utilization of signal strength in service location falls into cross-layer design, where the physical and MAC layer knowledge of wireless medium is shared with higher layers, in order to improve efficiency [Shakkottai et al., 2003]. It is motivated by the fact that layer triggers [Conti et al., 2004], i.e., predefined signals to notify events (e.g., route breakage) between protocols of adjacent layers, fall short of providing enough mobility-awareness for service discovery in mobile environments. For example, a link breakage causes the MAC protocol to report the link failure to routing layer, which updates the routing table subsequently. However, relying solely on such a mechanism results in a considerable probability of service failure in presence of device mobility. Therefore, to improve robustness against device mobility, service location needs to obtain more information regarding the underlying wireless links. As received signal strength, which is available in physical layer, indicates node mobility, it can be exploited to locate services that are less likely to fail because of device mobility.

IV.2 Signal Strength based Service Location (S3L)

In brief, given every neighbor in the network sending beacons periodically, when a neighbor is moving further (resp. closer), the average of SS over *locality distance* (e.g., 40λ) is decreasing (resp. increasing) [Rappaport, 2002]. Here, *neighbors* refer to the nodes reachable in one hop. Therefore, by keeping the SS of recent consecutive beacons from a neighbor, the link's *tendency to break* can be recognized by studying whether the average SS over consecutive locality distances is increasing or decreasing. In addition, a neighbor's departure or failure can be detected by the absence of its beacons for some threshold length of time (e.g, three times the beacon interval in OLSR [Clausen and Jacquet, 2003]).

Motivated by the above observations, we consider a wireless link *stable* only when: (1) the signal qualities (i.e., SS and SNR) from both ends are above receiver-specific threshold values, (2) the average SS over consecutive locality distances is not decreasing, and (3) both ends of the link are active (i.e., exchange of beacon messages occurs recently enough). The neighbor on the other end of a reliable link is named a *Strongly Connected Neighbor* (SCN).

Subsequently, we propose a Signal Strength based Service Location (S3L) method that performs service discovery only along stable links, so that the path between service requester and provider is reliable. A path being reliable

refers to the fact that it is unlikely to break soon. The underlying method is that by regularly broadcasting beacon packets to neighbors, each node derives the stability of links from the signal strength and its variation tendency and recognize its SCNs.

S3L does not impose any requirement on how service information should be retrieved, whether push or pull based. It leaves open how services are described, matched or accessed, while focusing on how the service information is forwarded during service discovery. Using S3L, the service information is always sent (either pushed or pulled) along the links that are considered stable by S3L. For pull-based SDPs, each node sends service queries only to SCNs, which may further forward the service queries to their SCNs. For push-based SDPs, each node sends its service advertisements only to SCNs, which may publish the advertisements further to theirs SCNs. In other words, the service discovery and delivery are only carried out along stable links by avoiding those weak, obsolete and diminishing links.

IV.2.1 Service Location Process

With S3L, each node regularly broadcasts beacons to its (one-hop) neighbors and in the meantime receives beacons from them. The stable links are recognized based on the average signal strength of received beacons. We illustrate the process using a pull-based SDP, but it works similarly for push-based SDPs.

With a pull-based (reactive) model, a client sends its service discovery request along stable links. Providers of services that satisfy the user request respond by sending back their service information, including QoS values, access method, etc. Then, the service client invokes one service (if there are multiple responses, one is selected that best matches the client, to be presented in the next chapter), which is carried out along the path returned by S3L.

To implement the above process, the following five types of packets are defined and used:

- `serv_beacon` is for exchanging signal information among neighbors.
- `serv_disc` is a service discovery request from a service client.
- `serv_resp` is a service response message from a service provider.
- `serv_invo` is the invocation message from the client to access the service.

- `serv_ack` is the service result from the service provider to the client.

We illustrate how S3L locates services by at first explaining the periodic beacons sent by every node.

IV.2.1.1 Beacon

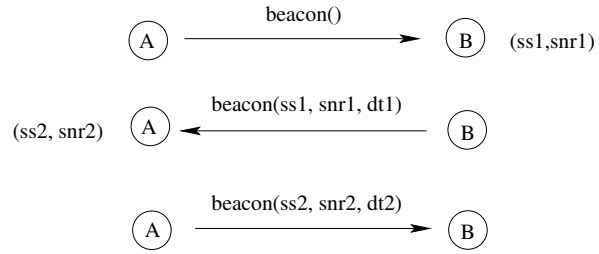


Figure IV.4: Beacon sending in S3L

A wireless link is asymmetric if

- one end has good link quality, i.e., with strong SS and high SNR, while the other end does not. We call it *different strength* between two ends: if one end receives beacons with strong signal strength and high SNR, it is considered strong; otherwise it is considered weak.
- one end can successfully receive beacons, while the other end cannot (e.g., due to different traffic conditions). We call it *different activeness* between two ends: if one end receives a beacon recently, it is considered active; otherwise it is considered inactive.

In order to observe whether a link is symmetric or not, a node *a* needs to inform the other end *b* of (1) the link quality from its end and (2) when *a* received a beacon from *b* last time. Note that care has to be taken regarding exchanging the timestamps of last received beacons because it can require synchronization of all nodes (e.g., with GPS), which is quite unpractical given the heterogeneity of devices (e.g., not every device is equipped with a GPS). To circumvent this problem, nodes exchange the last beacon's age instead of its arrival time. For example, in Figure IV.4, there is a wireless link between node A and B, which they might receive different link quality (e.g., different SNR). Assume A is a newcomer to this network and thus broadcasts its first beacon to signal its existence. Node B receives a beacon from A with a SS of `ss1` and SNR of `snr1`. Next time when B sends its beacon, it informs

A of the SS, SNR and the age of the last beacon from *A* (*dt1*), i.e., the length of time since it received *A*'s beacon last time. Similarly, *A* will include the information about the SS, SNR, and age of *B*'s last beacon (`ss2`, `snr2`, `dt2`). With accumulation of more beacons, nodes send average (SS, SNR) instead of the (SS, SNR) of last beacon to counter fluctuation. Note that when *B* is a new neighbor of *A*, i.e., *A* never receives *B*'s beacon before, *A*'s beacon does not have any signal information about the link (e.g., the first beacon in Figure IV.4). With the above handshake of beacons, two ends of a link can get to know the link quality from both ends and the ages of the last beacons from them. It can be easily extended to multiple neighbors, with a beacon including the information of all links to its neighbors. Therefore, for every beacon interval, a node broadcasts to its (one-hop) neighbors a UDP packet `serv_beacon`, in which it informs the neighbors about the average link qualities over the most recent locality distance from its end (Table IV.1). Note that the beacon includes the link information of all of its (one-hop) neighbors, whether they are SCNs or not, in order to dynamically evaluate link stability. Despite the amount of information carried in the beacon, the incurred packet size of a beacon is moderate because of the limited number of one-hop neighbors even in dense wireless networks. For example, in a network of 210 nodes with a communication range of 100 meters in an area of $350m \times 350m$, the average number of neighbor is about 21.2 [Williams and Camp, 2002]. On receiving a `serv_beacon`, a node extracts SS and SNR of its end (e.g., from the wireless card driver), notes down the reception timestamp of the beacon and extracts the signal quality (i.e., SS and SNR) and timestamp of the other end embedded in the `serv_beacon`. Subsequently, both nodes recognize the link quality from both ends.

$(nid_1, ss_1, snr_1, dt_1)$	$(nid_2, ss_2, snr_2, dt_2)$	...	

Table IV.1: A serv_beacon packet in S3L

A node only sends the average *SS* and *SNR* of its end (e.g., the average of most recent 5 samples) instead of multiple samples because it is not necessary: if a link is considered expiring at one end, it is also detected as being so at the other end, given the phenomenon of "average SS weakens with two ends moving away" as discussed in Section IV.1.

nid	$SS_{[1..2*k]}$	$SNR_{[1..k]}$	TS	nSS	nSNR	nAge	isSCN

Table IV.2: An entry of the neighbor table in S3L

Based on the above beacon mechanism, each node tracks the signal information of every wireless connection (to its neighbors), by keeping a *neighbor*

table (NT), which contains a list of its (one-hop) neighbors. As shown in Table IV.2, each entry contains:

- a neighbor's ID (nid).
- the SS of the most recent ($2*k$) received beacons from the neighbor. They are continuously updated with reception of new beacons. For example, the most 10 recent beacons have SS of (ordered by recency) (ss_1, ss_2, ...ss_{10}), The average of most recent 5 samples are compared against the 5 samples before that, i.e., $\sum_{i=1}^{5}(ss_i)/5$ against $\sum_{i=5}^{10}(ss_i)/5$. If the average is decreasing, this neighbor is considered to be leaving.
- the SNRs of the most recent (k) received beacons from the neighbor. They are used to calculate the average SNR of recent beacons. As there is no comparison to make, only half of the SNR samples need to be kept compared to the number of SS samples.
- the arrival time of the last beacon from the neighbor. This is for judging the activeness of this end.
- average SS and SNR from the neighbor's end (nSS and nSNR), for judging the signal quality from the other end of the link.
- the age of the last beacon received by the neighbor (nAge), for telling the activeness of the other end.
- an indicator showing whether the neighbor is a SCN or not, for facilitating lookup of SCNs among neighbors.

The neighbor table does not impose large storage overhead because of the limited number of (one-hop) neighbors. A node's departure or failure is identified by lack of beacons for some threshold time, noted δ_b. Moreover, for every δ_b, a node reevaluates its SCNs by going through all entries of the neighbor table and deciding whether a neighbor is an SCN according to the three criteria as stated at the start of this section. An SCN can be unmarked if it is considered no longer strongly connected, e.g., it starts moving away; similarly, a previous non-SCN can become an SCN, e.g., it stops moving away. A node whose beacons are older than δ_b is no longer considered as a (one-hop) neighbor and thus deleted from the neighbor table.

The beacon interval is an important parameter in S3L, not only because of its effect on message overhead on the network introduced by beacons, but also because of its impact on the number of beacons that need to be "grouped"

together for averaging. Given the relative speed between two nodes of v and locality distance of 40λ, the time to cover the locality distance is $40\lambda/v$. During this period of time, assuming that beacons are sent every b seconds, about $n = 40\lambda/(b \times v)$ beacons are collected. Since averaging requires multiple samples in order to counter fluctuation, the larger n is, the more effective the mobility detection is. However, given an average speed of v, larger n infers smaller b, i.e., more frequent beacon sending. For example, in an IEEE 802.11b network, the average speed is 1.5 m/s (human walking speed). In order to collect $n = 4$ beacons for averaging, it requires that b = $(40 \times 0.125)/(4 \times 1.5) = 0.9$, meaning that beacons should be sent at least as frequently as every 0.9 second.

IV.2.1.2 Service Location

A node looking for a service broadcasts a `serv_disc` (as shown in Table IV.3) in one hop to its SCNs, which includes: (i) a unique sequence number composed by the node's address and an increasing counter; (ii) a service path with the first address filled with its own address; (iii) the propagation range (in number of hops) of the request; (iv) the service's functionality requirement (we ignore the QoS attributes for the moment now, which will be addressed in the next chapter); (v) the addresses of SCNs, with the optional signal information of the links to its SCNs (similar to beacons).

seq #	functionality	path(n_1, , ,)	range	(scn_1, scn_2, ... scn_k)

Table IV.3: A serv_disc packet in S3L

One-hop broadcasting is preferred over multicasting here because the former is simple and does not need to maintain a tree or mesh-like structure as in multicast routing protocols on MANET (e.g., Ad Hoc On Demand Distance Vector (AODV) multicast [Royer and Perkins, 1999]). In a `serv_disc` packet, the list of destinations (i.e., SCNs) is specified such that non-SCNs ignore it. The only disadvantage of doing this is that, for non-destination nodes, the packets are dropped at the application layer instead of network layer. Note that if there is only one destined SCN, the packet is sent by unicasting. To avoid unnecessary rebroadcasting, each node keeps a cache of sequence numbers of the `serv_disc`s it has already handled. To prevent looping, the service path in `serv_disc` contains the intermediate nodes which the service query has traversed.

On receiving a `serv_disc` packet,

(1) a node verifies whether it has already handled the query by looking up

the sequence number in the cache. If it has, it simply discards the packet.

(2) Otherwise, it checks whether it provides the requested service. If that is the case, it appends its address and sends back by unicasting a `serv_resp` (as in Table IV.4) along the reverse path that is filled in `serv_disc`. The unicasting can be done with "source routing" by specifying in the IP header a complete, ordered list of nodes through which the packet will pass, if it is supported by the underlying routing protocol (e.g., Dynamic Source Routing (DSR) [Johnson et al., 2004]). As this support cannot always be assumed to exist, we put the service path in the payload of the packet, as shown in Table IV.4. The `serv_resp` packet also includes a sequence number to identify the service request (in case a client sends multiple requests for different services).

(3) Otherwise, it checks whether the `serv_disc` has already traversed the furthest distance required by the service client: the field of *range* is subtracted by 1 every time the packet is forwarded. When *range* reaches 0, the message is no longer forwarded. If the range is greater than 0, it calculates the destined SCNs, i.e., its SCNs that are not already in the service path of `serv_disc`.

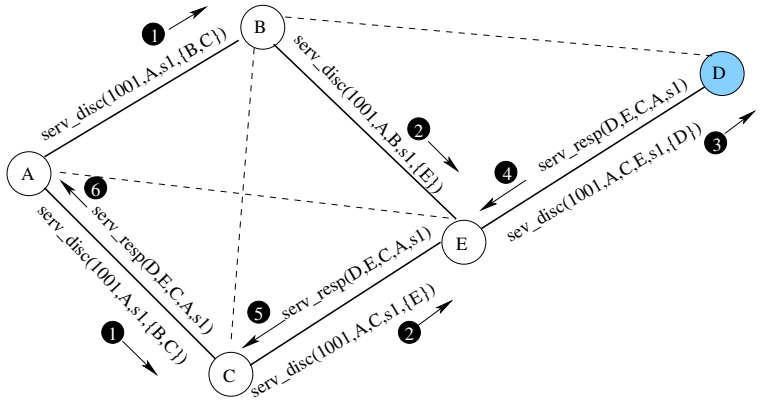
- If the destined SCNs include more than one node, it appends its address to the end of the service path and rebroadcasts the `serv_disc` to its SCNs.
- If the destined SCN only includes one node, the `serv_disc` is unicast to that SCN. For both of the above cases, the query sequence number is added into the cache of handled service queries.
- Otherwise, nothing needs to be done.

seq #	path(n_1,n_2,...,n_k)

Table IV.4: A serv_resp packet in S3L

On receiving a `serv_resp`, a node first reads the service path embedded in the message to check whether the packet is destined to itself, i.e., whether it is the client requesting for the service. If it is the client, it either waits for more service replies or invokes the service by sending `serv_invo` to the service provider along the reverse of the service path in the `serv_resp`. Otherwise, it forwards the packet to the next node in the service path.

For example, a network of 5 nodes is shown in Figure IV.5 with solid lines denoting stable links and dotted lines representing unstable links. Assume



Node	SCN
A	(B, C)
B	(A, E)
C	(A, E)
D	(E)
E	(B, C, D)

Figure IV.5: An example of service location with S3L

that node A is looking for a service named *s1* and broadcasts the request to its SCNs of nodes B and C at step ❶. The `serv_disc` packet includes the sequence number `1001`, service path `(A,...,)`, functionality requirement `s1` and the destined SCNs (`{B, C}`). Nodes B and C do not host any service of *s1*, thus they unicast a `serv_disc` to E since there is one destined SCN (A is already in the path). E receives `serv_disc` packets from both B and C. Assuming C's packet arrives earlier than B, E handles C's request and broadcasts a `serv_disc` to its SCNs of B and D (C is also its SCN, but C is already in the service path). C will ignore the request from B, so will B ignore the request from C, by checking the sequence number against the cache. Note that the caching of the sequence numbers of the handled `serv_disc`s is only an optimization means: a cache miss will not affect the correct operation of S3L, because the embedded service path is guaranteed to be loop-free thanks to the selective forwarding of `serv_disc` (i.e., to the SCNs that are not already in the service path). After D receives the `serv_disc` from E, it unicasts to A a reply of `serv_resp`, including its service information, along the path embedded in `serv_disc`. At step ❻, A receives the response from D and gets to know the service path of $A \rightarrow C \rightarrow E \rightarrow D$.

IV.2.2 S3L Analysis

In essence, S3L avoids the unstable links that, by its estimation, will break before the service finishes. The time span between when the service provider sends back a reply and when the service finishes can be estimated as the sum of service latency (l) and the message round trip time (rtt). During that period of time, the distance between the two nodes is increased by $d = (l + rtt) \times v$ (Figure IV.6), where v denotes the relative speed between the two. The smaller

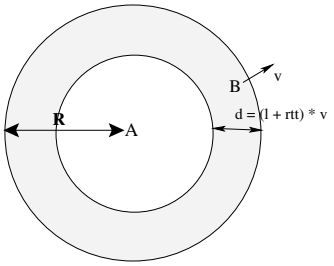


Figure IV.6: Link stability between two nodes

the service latency is, the shorter d is, the less possible that the link gets broken (as the link breaks only if B rests in the shaded area).

Therefore, the accuracy of S3L is greatly affected by the service latency l. If the service lasts a very short time (e.g., 1 second), S3L becomes unnecessary because even if it detects that the link is moving towards getting broken, it is very probable that the service already finishes when that happens. Meanwhile, if a service takes a long time (saying, 5 minutes), the estimation becomes less accurate because of two reasons. First, it is probable that the link becomes unstable (e.g., two ends start moving away from each other) after the service starts. Although this is also possible for services of smaller latency, the probability gets higher with longer period of time. Second, if the two nodes are approaching each other at the beginning, the link between them is considered stable by S3L. However, after passing each other, they start leaving each other and the link becomes expiring. Therefore, it necessitates stricter definition of stable links (e.g., neither approaching nor leaving) when services take a long time. Using long-latency remote services is thus more difficult to realize in mobile and dynamic environments such as *USoCo* environments. The other factor affecting d is speed. Very large speed leads to the emission of very few beacons when a node covers a long distance or before a link breaks. As insufficient number of beacons make it difficult to detect mobility tendency, S3L can fail to identify such unstable links. The performance of S3L under various mobility speeds and service latency is evaluated in the next section.

IV.3 Performance Evaluation

In this section, we evaluate the performance of S3L. Our main objective is to investigate whether our proposal of signal strength based service location improves robustness against device mobility and whether it exhibits satisfactory performance in terms of service location latency.

From a service client's perspective, its ultimate concern is whether its service request is served successfully. A failure can result from two possibilities. The first is the service location does not find a service which satisfies the client's request. This can be due to either there is no provider offering such a service in the vicinity or the service providers are intentionally omitted (e.g., if they are on unstable links in S3L). The second possibility is that the located services fail during delivery because of link disconnection. We thus measure the number of successful and failed service deliveries, the sum of which equals to the number of successful service discoveries (i.e., finding a service instance that satisfies the user's request). From these two parameters, we further derive *delivery success ratio*, which is the number of successful service deliveries divided by the total number of service invocations. In terms of overhead, we measure *service location latency*, which equals the length of the interval between when a client issues a service discovery request (i.e., a `serv_disc`) and when it receives a service discovery reply (i.e., a `serv_resp`). Therefore, overall, we investigate the following metrics: (1) service success and failure numbers; (2) delivery success ratio; and (3) service discovery latency.

IV.3.1 Simulation Environment

We evaluate our solution using *ns-2* with CMU wireless extensions [LBNL, 2001] because of its availability as open source and active community discussion[3]. Our simulated network consists of 40 mobile nodes in an area of from 350m $\times$ 350m to 700m $\times$ 700m (increasing by 50m for each simulation setting), with each node having a transmission range of 100m using *Ricean* propagation model [Punnoose et al., 2000]. Ricean fading is one way to model small scale fading, along with Rayleigh fading. The difference between the two lies in that the former is for small scale fading with line of sight while the latter is for no line of sight [Rappaport, 2002]. We assume the *Random Waypoint* mobility model with each node moving at walking speed, i.e., between 0.5m/s and 2m/s, with pause time of 0, i.e., nodes are always in motion. The speed will be also adjusted when we investigate the impact of speed on the performance. The Distributed Coordination Function (DCF) of the IEEE 802.11 protocol is used as the MAC layer protocol. Each wireless channel has bandwidth of 2 Mbps.

We assume that each node can host up to *nServ* services, out of 50 services (thus $nServ \leq 50$). Each node randomly requests a service that it does

[3] `http://www.isi.edu/nsnam/ns/ns-lists.html`

Parameter	Value
Mobility Model	Random Way Point
Moving Speed	0.5 - 1.5 m/s
Pause Time	0
Propagation Model	Ricean Fading
Transmission Range	100 m
Bandwidth	2 Mbps
Area	from 350m x 350m to 700m x 700m
Number of Nodes	40
# of serv per Node (out of 50)	25
service latency	from 1 to 33 seconds

Table IV.5: NS-2 simulation parameters for evaluating S3L

not provide. Every service is served as a Constant Bit Rate (CBR) streaming application of 56 Kbps. The larger the *nServ* is, the more likely that a node is able to locate the requested service at nearby nodes. This is already taken into account with different simulation areas, since the smaller the area is, the more dense the nodes are and with higher possibility that a nearby node provides the requested service. Therefore, we set $nServ$ with a fixed value, e.g., 25, while measuring the metrics with different areas.

As the focus is to evaluate the robustness against mobility of the service location, the client always selects the first service reply and invokes the service. Beacons are sent every 2 seconds, and locality distance is set to 5 meters, i.e., about 40 λ for IEEE 802.11b. The whole simulation lasts 500 seconds. Because it takes time for the random waypoint model to reach a stable distribution of mobile nodes [Camp et al., 2002], the initial part of the simulation is not representative of the reality and is therefore discarded. At the end of the simulation, because the service lasts some time and the service client needs some time to wait for the arrival of service results, no queries are sent at the end of the simulation. Therefore, we choose to skip the first 200 seconds of the simulation result. From $200s$ to $450s$, for every second, there is a node sending a service discovery request in a round-robin way. Each simulation setting is executed 30 times. Thus there are a total of $250 \times 30 = 7500$ service discovery attempts for each setting and the average is presented.

For the sake of comparison, we implement two simple reactive service discovery protocols: one base on S3L (for the simplicity of presentation, we still name it S3L) and the other, named *DIST*, which is almost identical with S3L

[4]It refers to beacons sent during the service discovery.

Setting	S3L	DIST
Routing Protocol	none	OLSR
Service Request Propagation	Selective Broadcast	MPR's Broadcast
Beacon[4]	every 2 seconds	none

Table IV.6: Difference between DIST and S3L

except that it does not distinguish between stable and unstable links and it uses a routing protocol. We choose Optimized Link State Routing Protocol (OLSR) [Clausen and Jacquet, 2003] because of its wide usage and the availability of simulation codes for ns-2[5]. OLSR is a proactive routing protocol for MANET, i.e., each node exchanges topology information with others regularly. Every node selects a set of its neighbor nodes as *multipoint relays* (MPR), which are the only nodes responsible for forwarding control traffic. MPRs provide an efficient mechanism for flooding control traffic by reducing the number of transmissions. This makes OLSR a good choice as the underlying routing protocol for DIST as it propagates the service requests using broadcast. In contrast, S3L does not use any routing protocol, as it finds a service path by itself, which is used for later service delivery. The difference between S3L and DIST is shown in Table IV.6. In DIST, a client broadcasts its service request in the vicinity network while in S3L, it only sends the request along stable links. As only SCNs respond to the broadcast packets, we name it "selective broadcast". The ranges of service request propagation in both S3L and DIST are set to 4 hops. The purpose of this comparison is two-fold: (1) to evaluate whether service location based on signal strength tendency does improve service reliability; (2) to justify our choice of cross-layer design, i.e., instead of relying on routing protocols for handling mobility, S3L extracts signal information directly from the physical layer to detect node mobility and improve service reliability.

IV.3.2 Evaluation Results

Delivery Success Ratio. At first, by setting the service latency to 25 seconds, we explore the number of success and failed services with different simulation areas. The performance of S3L and DIST with respect to the number of service success and failure (out of 250 queries) are compared in Figure IV.7. It can be observed that S3L outperforms DIST by having more successful service deliveries. Particularly, it cuts down the number of service failures by 50% to 70%. From the numbers of service success and failure, we derive *delivery*

[5] `http://hipercom.inria.fr/olsr/`

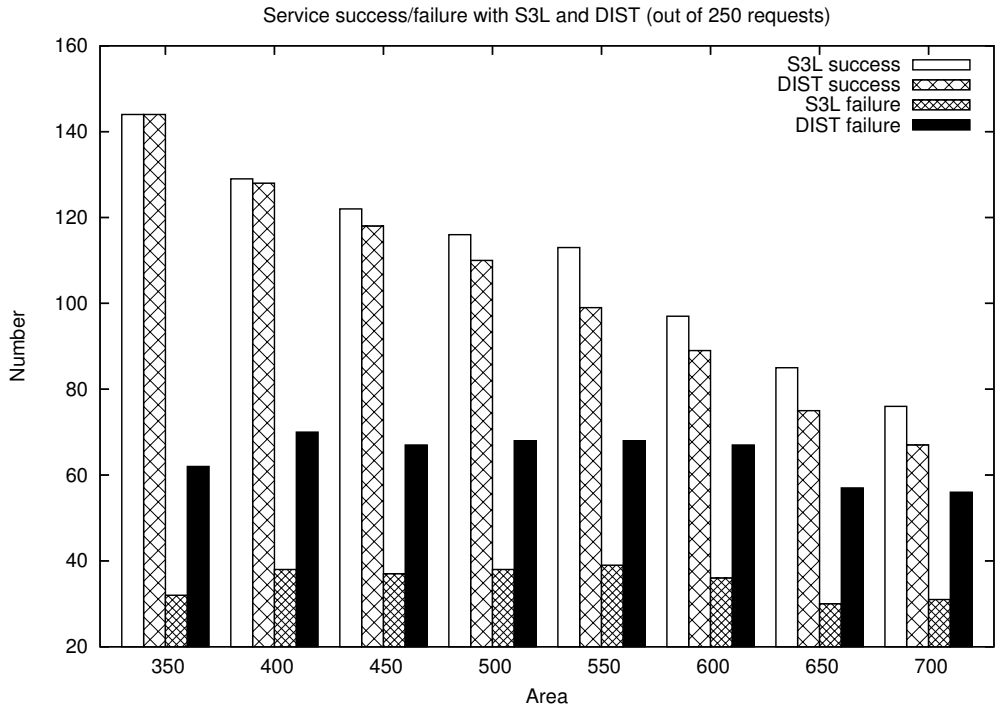


Figure IV.7: Number of successful and failed service deliveries

success ratio (Figure IV.8), which confirms S3L's better performance than DIST.

area	latency (in ms)	
	S3L	DIST
350m x 350m	16.873	16.130
400m x 400m	18.231	16.512
450m x 450m	16.401	17.964
500m x 500m	15.873	15.560
550m x 550m	16.269	15.810
600m x 600m	14.753	15.734
650m x 650m	13.786	16.142
700m x 700m	13.771	16.278

Table IV.7: Service location latency of DIST and S3L

Service Discovery Latency. Service discovery latency for S3L and DIST are presented in Table IV.7. It can be seen that there is no much difference (less than 3 ms in between), although S3L outperforms DIST to a small extent.

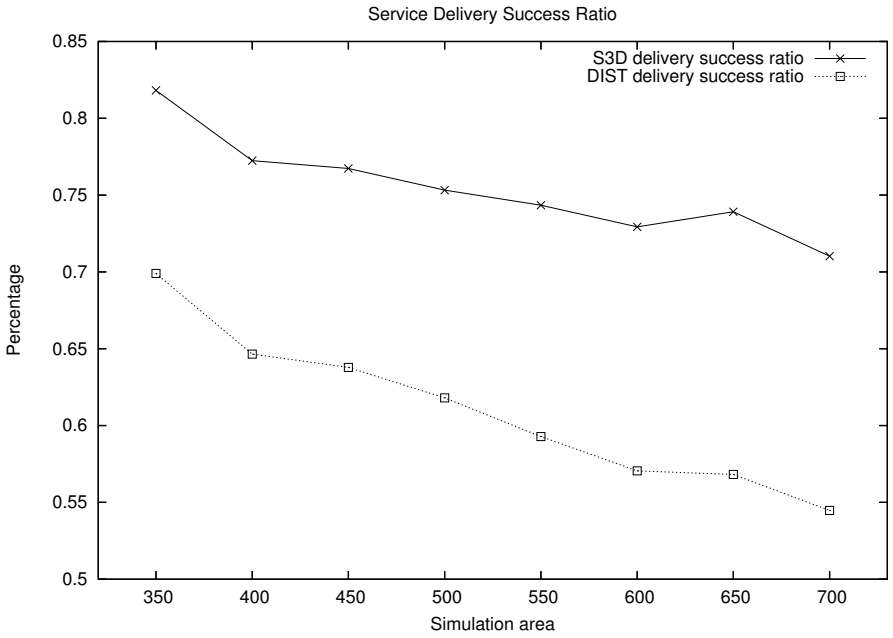


Figure IV.8: Service delivery success ratio

This is because both S3L and DIST have similar overhead in sent messages: in S3L, only SCNs rebroadcast service discovery messages; while for DIST, only MPRs rebroadcast them.

Impact of Mobility. The impact of mobility on the performance of S3L and DIST is investigated (Figure IV.9). In *ns-2*, the speed of a node is randomly selected from the range between *minspeed* and *maxspeed*. By fixing minspeed at 0.5 m/s, we evaluate the performance with maxspeed changing from 2 m/s to 21 m/s. It can be seen that with maxspeed less than 11 m/s, S3L outperforms DIST, although the advantage is getting less and less with larger speed; with speed higher than 11 m/s, DIST has almost as good performance as S3L. This is due to the fact that, with nodes moving at high speeds, the nodes cannot collect enough number of beacons for deriving mobility and thus cannot identify unstable links.

Impact of Service Latency. The impact of service latency on the performance of S3L and DIST is also studied (Figure IV.10) by adjusting the latency from 1 to 33 seconds (increasing by 4 seconds for each setting). It can be observed that with longer latency, both S3L and DIST suffer worse performance, because there is higher probability to fail with longer latency. When service latency is

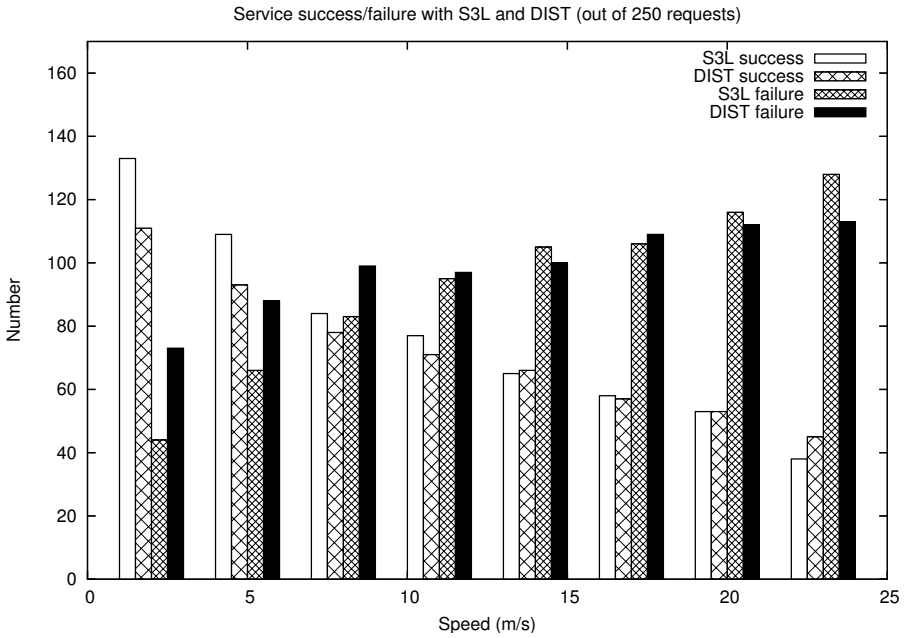


Figure IV.9: Number of successful and failed service deliveries with different speeds

small (e.g., 1 second), S3L outperforms DIST with smaller number of service failure while the latter has larger number of service success. This is because with the latency of as short as 1 second, services can succeed even over unstable links. But starting from latency of 8 seconds, S3L outperforms DIST in both number of successful and failed service deliveries thanks to its enhanced robustness to mobility.

Summary. It can be concluded from the above experiments that S3L improves service reliability, compared to an alternative service location approach that is indifferent about link stability (i.e., DIST), as long as service latency is not too small (larger than 1 second) and speed is not too high (less than about 11 m/s). Such loose conditions show the wide range of S3L's applicability.

IV.4 Concluding Remarks

In this chapter, we have presented signal strength based service location (S3L) for improving service reliability in presence of device mobility. Our contribution lies in (1) the elicitation of needs for mobility awareness during service

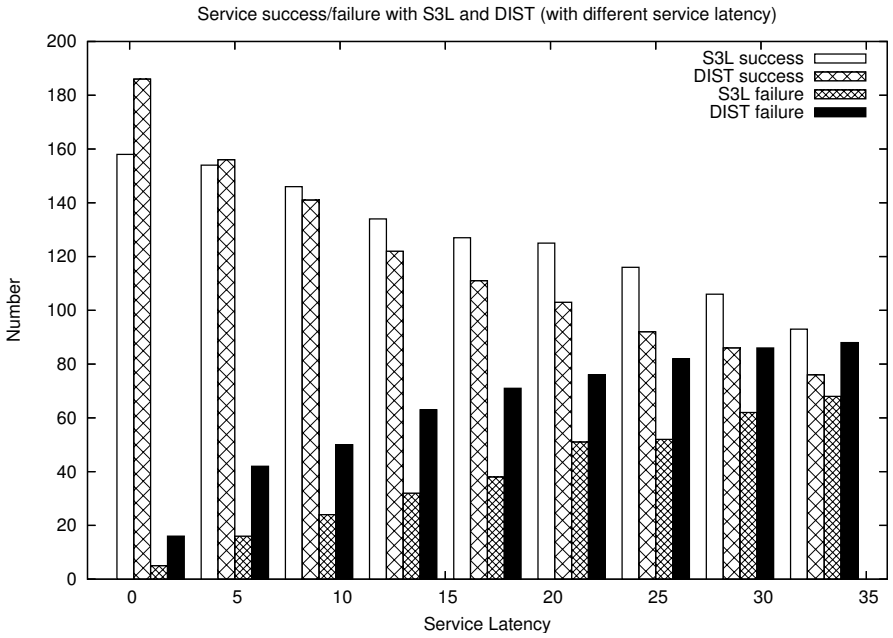


Figure IV.10: Number of successful and failed service deliveries with different service latency

location and (2) a service location method based on signal strength tendency (i.e., S3L) that proves to improve service reliability.

S3L focuses on improving robustness to node mobility and thus service reliability. Such an enhancement is generally at the cost of impairing other QoS properties. Aggressive measures to improve reliability can lead to longer service latency, weaker service availability, etc, and should be justified only when the reliability gain outweighs the loss regarding other QoS properties. S3L forwards service information only along stable links and improves service reliability at the cost of weaker service availability – the services situated on the unstable links are excluded during service location and thus are unavailable to the client. However, in most cases, service discovery is followed by service delivery. An aborted service delivery after successfully finding a service instance is essentially equivalent to failure to find a requested service, which justifies our sacrifice of availability for reliability.

The extension beyond the work in this chapter can be carried out along incorporation of other QoS dimensions, besides reliability, into service location. However, it has to be noted that not every QoS dimension can be considered during service location. It only applies to the QoS properties whose values experienced by the client are affected by the service path. For example, the

experienced service latency, which is the sum of service processing latency at the service provider and the network transmission time, is applicable. Therefore, a service location request can be aborted if the discovery latency so far has already surpassed the maximum latency acceptable for the client. However, it is a non-trivial task because it requires synchronization of all nodes to be able to calculate the passing time since the sending of the request. In contrast to service latency, the cost-related dimensions are inappropriate, as they are only affected by service providers and independent of service path.

V

QoS-aware Service Selection Using Vickrey auction

Service location potentially finds multiple instances that satisfy the client's requirements. It is then the responsibility of service selection to choose the best one on behalf of the client. Most current SDPs lack such a feature, which forces the client to manually make choices. Service selection can be a very tedious and error-prone process as it involves with various QoS dimensions of different units, which can be further complicated by the client's different preferences among different aspects (e.g., price-driven or QoS-oriented). This calls for service selection that evaluates the service instances based on their QoS properties, prices and the client's preference.

Moreover, given the same level of service QoS offered by service providers, it is desirable to choose the service incurring the lowest overhead, i.e., consuming the least resources – because it realizes the system-wide goal of realizing the best QoS at the lowest cost. However, the client's selection is determined by service QoS and price, the latter of which is up to the service providers' strategies and thus does not always reflect the real overhead. Therefore, there exists a gap between the global goal and individual interests.

In this chapter, we propose a solution to address the above two issues during service selection. Our contribution is three-fold: (1) giving an extensible QoS model that includes a service's generic QoS properties; (2) proposing a comprehensive utility function that evaluates a service instance in satisfying the client's needs; (3) applying Vickrey auction [Vickrey, 1961] as the pricing model to enforce the revelation of truthful service price to achieve the global goal.

The rest of the chapter is organized as follows. The next section presents

a QoS model that includes generic QoS properties. Section V.2 details the selection process, including service evaluation by measuring the overall QoS of a service and the pricing model for determining service price. The service selection is analyzed in Section V.3, focusing on the overall QoS evaluation. This chapter finishes with concluding remarks in Section V.4.

V.1 A QoS Model

In this section, we present a QoS model that includes generic QoS properties of a service. The QoS properties are carefully selected by considering their genericness and suitability for services in *USoCo* environments. Meanwhile, the QoS model is extensible in that new QoS criteria (e.g., domain-specific QoS properties) can be easily added without altering the underlying service evaluation shown in Section V.3. It is used by service clients to specify their QoS requirements and service providers to specify their QoS offers. It is closely related to QoS guarantees and commitments defined in Service Level Agreements (SLA) (e.g., [Beckman et al., 2002]), although the latter also needs to define penalty clauses and QoS monitoring which are related to the enforcement of SLAs.

Based on current work such as [Ran, 2003, Zeng et al., 2003, Sabata et al., 1997, Avizienis et al., 2001], QoS properties of a service can be divided by relevance into categories of *performance*, *dependability*, *transaction*, *trustworthiness*, *cost* and *service behavior*. A *category* represents a group of related QoS properties, named *dimensions*. A quantitative dimension is named a *metric*, while a qualitative one is called a *policy*. Of the above six categories, *performance* is considered basic and incorporated into many QoS models (e.g., [Frolund and Koistinen, 1998, Ran, 2003]). *Dependability* is regarded as a one of the four fundamental properties (along with *functionality*, *performance* and *cost*) of computing systems [Avizienis et al., 2001]. Service *cost* is also included since it is widely regarded as a vital property of a service (e.g., in [Chalmers and Sloman, 1999] and [Cardoso et al., 2004]). *Transaction* is integrated into the QoS model as well in order to facilitate mobile commerce applications. *Service behavior*, which describes a service's policies in terms of adaptation and service guarantees, is necessary to capture a service's dynamics. These categories are further developed and explained as follows.

Performance measures the speed in completing a service request [Sabata et al., 1997, Chalmers and Sloman, 1999]. Common performance dimensions include *latency*, *throughput* and *jitter* [Ran, 2003]. Latency refers to the total time taken to complete a service request; throughput represents the number

of completed service requests over a time period; jitter means the variation in latency.

Dependability of a system refers to the ability to deliver services that can justifiably be trusted [Avizienis et al., 2001]. Dependability encompasses the following dimensions:

- *availability*: the probability that a service is available when clients attempt to use it [Koistinen, 1997];
- *reliability*: the probability of a service performing its purpose adequately for a period of time intended under the operation conditions encountered [Reibman and Veeraraghavan, 1991];
- *safety*: absence of catastrophic consequences on the user(s) and the environment [Avizienis et al., 2001];
- *confidentiality*: absence of unauthorized disclosure of information;
- *integrity*: absence of improper system state alterations;
- *maintainability*: ability to undergo repairs and modifications [Avizienis et al., 2001].

Among the above dimensions, both *availability* and *reliability* are metrics and assume values between 0 and 1, while the others are policies and assume boolean values. *Security* has not been included since it is a concurrent category with *dependability* and it includes (a) availability for authorized users only; (b) confidentiality and (c) integrity with the "improper" in the above definitions meaning "unauthorized" [Avizienis et al., 2001, Pfleeger, 1997]. Thus all of the three dimensions are covered in the umbrella of *dependability*.

Since many real-world services are transactions (e.g., ticket booking) and transaction support is essential to allow for wide adoption of mobile commerce, we list it as an individual category for its significance. Moreover, services are paid using virtual currency in *USoCo* environments (Section II.1.2) and thus service provisioning and consumption involve transactions. The *transaction* category embodies ACID (Atomicity, Consistency, Isolation and Durability) properties [Gray and Reuter, 1993], leading to the definition of the corresponding policies, which are all boolean. In addition, to facilitate transactions between service client and provider, we also include three main transaction processing styles: *direct transaction processing (DTP)*, *queued transaction processing (QTP)* and *compensation-based transaction processing (CTP)* [Gray and Reuter, 1993]. Service provider and consumer can agree on the transaction

style(s) to be supported. Consequently, we include these three policies in the QoS model to indicate whether a service supports that style. Note that the above three policies are not orthogonal in that a service can support one or multiple processing styles. QTP style seems to fit better transactions between mobile entities because they are normally short-lived and the involved service provider and client are loosely-coupled.

Trustworthiness evaluates the degree to which an entity will provide a service as expected. Different from *dependability* that focuses on the ability to deliver services, trustworthiness evaluates the willingness of an entity to do so. In another word, when a service provider does not fulfill the client's requirements as expected, it is undependable if it is incapable of doing so, while it is considered untrustworthy if it is able to but unwilling to do so. Trust towards a node can be considered as a prediction of that node's future action. One way to measure a node's trustworthiness is to evaluate its reputation. Subsequently, we include the dimension of *reputation* in the trustworthiness category. It assumes the value of beta reputation of (α, β), where α and β represent the accumulated positive and negative experiences respectively (to be detailed in Chapter VI).

Cost is a fundamental property of a service [Chalmers and Sloman, 1999, Cardoso et al., 2004, Avizienis et al., 2001], because if there is no notion of cost involved in QoS description, there is no reason for the user to select anything other than the highest level of quality of service [Roscoe and Bowen, 2000]. However, a concrete means to evaluate service cost is rarely specified. For example, in [Cardoso et al., 2004], cost is considered to include *enactment cost* and *realization cost*, which are associated with non-technical factors such as labor cost. Since the resource consumption introduced by a service on the resource-constrained handheld devices is the main overhead of service provisioning, we put the metric *resource consumption* in the *cost* category. By doing so, it expresses device heterogeneity in terms of resource richness: given a service, it costs less for a powerful provider than a less powerful one. Note that resource consumption refers to that of an elementary service, while for composite services, it is calculated according to the composition logic (e.g., [Cardoso et al., 2004]). We further consider the following 4 resources: CPU load, memory, bandwidth and battery. Each resource consumption is evaluated by dividing the consumed resource by available resource and the value range is between 0 and 1, as explained below.

CPU load describes the work load on the CPU(s) of a host. Available CPU load of a host is defined as the utilizable percentage of the CPU (i.e., 1 - utilization percentage). The CPU load introduced by a service is further defined as

the CPU time of the service[1] divided by the total service time. *Memory* refers to the size of the primary memory of a host. The available memory of a host is thus defined as the host's available primary memory, and the *memory consumption* of a service is defined as the amount of physical memory it utilizes. The available *bandwidth* for a host represents the actual capacity of its wireless link and the *bandwidth consumption* of a service refers to the volume of data the service sends and receives per some time units. The available battery for a host represents the power level a host has (an AC plugged-in host is considered to have infinite battery) and the *battery consumption* of a service refers to the power a host consumes for executing the service.

As various resources may bear different importance to the service host, *relative importance* is used to characterize the criticality of the various resources. For example, battery can be very important to an AC-unplugged host if it has some important tasks to execute in the near future. Service cost thus can be derived from the consumption of each resource and its relative importance. Assume that for some resource r, a service s consumes $ac_{s,r}$ units of the total available resource tar_r. We evaluate the resource consumption of r for the given service s by:

$$rc_{s,r} = \frac{ac_{s,r}}{tar_r}$$

The reason that $rc_{s,r}$ is defined as a relative instead of absolute value is that the same amount of resources are of different importance to different hosts (e.g., running a service that consumes `5 MB` memory has a different impact on a host with `250 MB` memory available than on a host with `10 MB` available). Consequently, service cost is formulated by:

$$cost \quad = \quad \sum_{r \in R} rc_r \times w_r \qquad where \quad \sum_{r \in R} w_r = 1$$

where R represents all the resources and w_r refers to the *relative importance* of resource r. The value of *cost* falls into $[0..1]$. Note that as service cost measures the overhead for a service provider to host a service, it is not necessarily published by the service provider, depending on environments (cooperative or not) and the service provider's strategy.

In face of resource variation, QoS guarantees can be violated and actions need to be taken. Therefore, *service behavior* is used to describe a service's level

[1]The CPU time of a service can be measured with tools like Java Virtual Machine Tool Interface.

of QoS guarantees and actions to be taken in case of resource variation, resulting in the following two dimensions: *level of service* and *service adaptation*.

Level of service specifies the degree of certainty that QoS levels requested at the time of demand will be honored [Aurrecoechea et al., 1998]. It gives the probability that the QoS values will be honored, regardless of how they are enforced (e.g., guaranteed or best-effort).

Fluctuation of resource availability requires a service to adapt to such change, which either happens locally (e.g., battery is running out) or in the surrounding environments (e.g., received signal is getting weaker). It has been widely considered necessary (e.g., [Noble et al., 1997, Bowers et al., 2000]) to have multiple feasible implementation alternatives of a software component, which can be dynamically selected based on current conditions of the application environment and the client's preferences. Each implementation alternative, named *fidelity* in [Narayanan et al., 2000], meets the basic goals but differs in the quality of service that it provides. In general, lower fidelity consumes less resources, but offers worse service to the client. Therefore, we use the *service adaptation* policy to capture the degree of QoS adaptation that a service can tolerate and scaling actions to be taken in the event of violation of the current QoS. During runtime, when the availability of resources varies significantly, a service provider can allow dynamic adjusting (so called renegotiation) depending on the resource availability - e.g., whether or not service is upgraded/downgraded when resources improve/deteriorate. Note that such adjusting needs to obey the level of service promised by the service provider. For example, for guaranteed level of service, the QoS cannot be degraded to be lower than the promised QoS. In addition, such dynamic adaptation is taken into account in evaluating service cost (e.g., using the average cost over multiple runs during which adaptations are made).

The above QoS dimensions along with their values are summarized in Table V.1. These dimensions are measured during service invocation. Note that the resource consumption is evaluated by dividing the service's consumed resources by the available resource on the node, both of which need to be measured. Along with other QoS dimensions, the measurement of them is explained below.

For *CPU load*, the system utilities *uptime* and *vmstat* on UNIX/Linux platforms[2] provide measurement of CPU availability on time-shared systems. Typically, the *uptime* utility reports CPU load average as the average number of processes in the run queue over the past one, five and fifteen min-

[2]Similar utilities are also available for other platforms, e.g., Performance Data Helper (PDH) library provided by MS Windows.

Category	Dimension	Value
Performance	Throughput	number of requests per time
	Latency	Service Time (in ms)
	Jitter	Time (in ms)
Dependability	Availability	[0..1]
	Reliability	[0..1]
	Safety	Boolean
	Confidentiality	Boolean
	Integrity	Boolean
	Maintainability	Boolean
Trustworthiness	Reputation	(α, β)
Transaction	Atomicity	Boolean
	Consistency	Boolean
	Isolation	Boolean
	Durability	Boolean
	DTP Support	Boolean
	QTP Support	Boolean
	CTP Support	Boolean
Cost	Resource Consumption	[0..1]
Service Behavior	Level of Service	Percentage=[0..1]
	Service Adaptation	adaptation policy

Table V.1: QoS model for services in ubiquitous computing environments

utes. Meanwhile, the *vmstat* utility reports percentages of user time, system time, idle time and Input/Output time. The fraction of CPU occupancy time for a full-priority standard user process can be evaluated as [Wolski et al., 1997, Wolski et al., 1999]:

$$availableCPU = T_{idle} + (T_{user}/rp) + (T_{user} \times T_{system}/rp)$$

where T_{idle}, T_{user} and T_{system} represent CPU idle time, user time and system time respectively, all in terms of percentage of total CPU time; and rp represents number of running processes. The rationale of this formula is that a new process would be entitled to all of the idle time, a fair share of the user time, and a part of system time proportional to the user time. The CPU time of a process with process id *pid* can be e.g., gathered from the `/proc/pid/stat` of Linux `proc` file system, which actually provides the system time, the user time, the children processes' system time and user time.

For memory, there are three kinds of virtual memory pages available to a service process: (1) pages already owned by the process – its *resident set*; (2) the unused *free pool*; (3) the pages owned by some other process, but not in the working set: *inactive pages*. All of them, along with the memory utilization of a process, can be obtained from the operating system, e.g., Linux `proc` file system. The bandwidth of an wireless ad hoc network can be measured by probing (e.g., [Chen et al., 2005]); the bandwidth consumption of a service can be obtained by monitoring the number of network bytes transmitted and received during the execution.

The current energy capacity of a node can be obtained via Advanced Configuration and Power Interface (ACPI) [ACPI, 2004], which gives the current capacity of the batteries. To measure the battery consumption of a service, intuitively we can measure the battery capacity difference between before and after the service is executed. It, however, needs to exclude the energy consumption of other concurrent tasks executing at the same time. An alternative approach to measure a service's battery consumption is to obtain the correlation between power consumption and other metrics such as CPU load, network traffic, etc (e.g., in [Nash et al., 2005, Feeney and Nilsson, 2001]). The latter parameters can be acquired from utilities provided by operating systems (e.g., Performance Data Helper library on MS Windows).

As for other QoS dimensions, for example, *availability* can be calculated by $A_s = T_s/\theta$, where T_s is the total amount of time when service s is available during the last θ amount of time. And the more frequently the service is accessed, the shorter θ should be [Zeng et al., 2004]. *Latency* can be measured as the length of the interval between when the service provider receives the service request and when it returns the result. Note that the latency given by the service provider (L_s) can be different from the experienced latency at the service client (L_c), because the latter also includes the network transmission time ($Trans_s$), i.e., $L_c = L_s + Trans_s$. Note that service providers advertise L_s (i.e., processing time of the service) instead of L_c because different service clients can experience different transmission time. Other dimensions can also be measured during the process of service invocation. Policies are valued according to the existence of mechanisms to enforce that policy. For example, *confidentiality* can be evaluated as being `true` only when a service does apply some encryption schemes during communication. The measured metric values can be further used by the service provider to forecast future values, using techniques such as mix-of-experts [Wolski, 1998, Gurun et al., 2004].

V.2 QoS-aware Service Selection

Besides QoS properties of a service, the overall service QoS is also determined by the client's preferences. We use an integrated metric called *user benefit* to evaluate the overall QoS of a service instance, indicating the expected benefit brought by a service to the client.

It needs to be noted that the QoS values claimed by the service provider can be different from the experienced QoS by the client (abbreviated as *experienced QoS*) due to two reasons. First, a service provider can be lying about *service QoS*, i.e., its truly offered QoS. This can be handled by trustworthiness evaluation of the service provider, which will be detailed in the next chapter. Second, experienced QoS is also affected by other factors, besides the service QoS. For instance, reliability and latency are two such QoS dimensions. As explained in the previous chapter, node mobility can degrade considerably service reliability if not taken good care of. One possibility is to remove the impact of mobility on service reliability, i.e., by guaranteeing that the service path does not break during service delivery. But it is difficult to achieve given the device mobility in the environment. S3L takes a step forward to alleviate, although not completely remove, the impact of node mobility on service reliability, by locating services along stable links. Similarly, a client's experienced latency is the sum of service latency and other factors, mainly the round trip time of sent messages (the message to invoke the service from the service client to the provider and the result sent back in the other direction). We measure the round trip time with a simple probing-based way without incurring any traffic overhead. The round trip time is estimated with *service location latency*, which is the interval between when the client sends a discovery request and when it receives the reply. It serves a decent estimation because it is a probing-based value – the path with small delay is probably short and uncongested and implies a similarly small delay during service delivery to be happening soon [Dykes et al., 2000]. Service location latency is thus added to service latency to estimate experienced latency of the client.

Different clients may have different preferences of QoS, which should be considered during the evaluation of *user benefit*. We reuse *relative importance*, which was used to describe how critical a resource is to a host, to represent the priority of a QoS dimension to a service client. For example, a less patient client can give a higher *relative importance* to service latency than other QoS dimensions to express its preference for fast services. The values of relative importance can be elicited using schemes such as reinforcement learning (e.g., [Lee et al., 2004]). The users give feedbacks after each service execution, e.g., increasing the *relative importance* of *latency* if the last execution took too long

a time. The preferences are thus continuously updated and remembered until the users are satisfied.

V.2.1 User Benefit

Based on QoS values and relative importance of each QoS dimension, *user benefit* computes the overall QoS brought to the client [Liu and Issarny, 2004b]:

$$\text{User Benefit} = \sum_{i=1}^{n}(d_i \times w_i) \qquad where \quad \sum_{i=1}^{n} w_i = 1 \tag{V.1}$$

Where d_i is the value of a QoS dimension and w_i denotes the client's assigned relative importance to the dimension. Note that if d_i is of a dimension whose value is boolean, d_i equals 1 if it is `true`, 0 otherwise. As dimensions can be of different units (e.g., *latency* is millisecond and *availability* in percentage), in order to allow for a uniform measurement of service QoS independent of units, data normalization is applied, which essentially transforms values of different units into comparable ones. Assume that two service instances have values of *orig1* and *orig2* for a QoS dimension, they are normalized to *norm1* and *norm2* respectively. In general, the normalization needs to

- keep the order of values, e.g., if *orig1* is stronger than *orig2*, *norm1* should be greater than *norm2*;
- maintain the range of values. Although a larger original value leads to a larger normalized value, the latter cannot be indefinitely large and should fall into a range;
- avoid the situation when an exceptionally large value completely overshadows other values by making them negligible, because it leads to complete discount of those values when evaluating the overall QoS.

Currently, there exist two common normalization techniques: (i) decimal scaling, (ii) standard deviation normalization. Assuming that $d(i)$ is the value of dimension d for a service instance i, decimal scaling normalizes every data by moving the decimal point:

$$d'(i) \quad = \quad \frac{d(i)}{10^k} \qquad \text{for the smallest k such that } max(|d'(i)|) < 1$$

	d_1	d_2	d'_1	d'_2	d''_1	d''_2
a	10	50	0.01	0.5	0.03	1.66
b	30	40	0.03	0.4	0.09	1.33
c	1000	1	1	0.01	3.00	0.03

Table V.2: QoS values before and after decimal scaling

Decimal scaling preserves most of the original character of the value and a typical scale maintains the values in the range of from -1 to $+1$. For example, consider 3 service instances in Table V.2: a, b and c. For simplicity, assume that only two dimensions need to be considered, e.g., d_1 and d_2, and that d_1 and d_2 are both stronger with larger values and have the same relative importance. After decimal scaling (d_1 becomes d'_1 and d_2 becomes d'_2, as shown in Table V.2), c is the best instance considering the sum of the two dimension values. But c does not have balanced properties (i.e., it is strong in d_1, but too weak in d_2). In general, when data values exhibit a wide range of magnitudes, it can be difficult to properly compare them without the aid of transformation. This is because the large values dominate the small ones, which makes it difficult to see other details in the rest of data [Seigel, 1988]. Given the heterogeneity of service providers in capability (e.g., computing power), the range of QoS values can be quite large, making decimal scaling a bad choice for normalizing data. A variant of decimal scaling is to divide every value by the average (e.g., in [Liu et al., 2004]). This approach has the same problem as decimal scaling, as shown in Table V.2 (d''_1 and d''_2).

Standard deviation normalization transforms data in a more radical way using means and standard deviation:

$$d'(i) = \frac{d(i) - m(d)}{\delta(d)}$$

where $d(i)$ is the value of dimension d for the service instance i, and $m(d)$ and $\delta(d)$ are the mean and standard deviation values for dimension d respectively. In addition, we need to set the maximum normalized value to deal with those exceptional values, using Chebyshev's theorem:

Theorem V.1. (Chebyshev's theorem) *The portion of data that lies within k standard deviations to either side of the mean is at least $1 - \frac{1}{k^2}$ for any data set, where k is a number greater than 1.*

By considering a 75% confidence interval, we let $k = 2$, leading to division of the space into $(-\infty..m - 2 \times \delta]$, $(m - 2 \times \delta..m + 2 \times \delta]$ and $(m + 2 \times \delta.. + \infty)$.

	d_1	d_2	d'_1	d'_2
a	10	50	0.35	0.70
b	30	40	0.36	0.59
c	1000	1	0.79	0.22

Table V.3: QoS values before and after standard normalization

Hence, the dimensions that are stronger with larger values (e.g., *availability*) are normalized according to the following equation:

$$d'(i) = \begin{cases} 1 & if(d(i) - m(d) > 2 \times \delta(d)) \\ 0 & if(d(i) - m(d) < -2 \times \delta(d)) \\ \frac{d(i)-m(d)}{4\times\delta(d)} + 0.5 & otherwise \end{cases}$$

While for QoS dimensions that are stronger with smaller values (e.g., *latency*), they are normalized according to the following equation so that smaller values contribute more to the *user benefit*:

$$d'(i) = \begin{cases} 0 & if(d(i) - m(d) > 2 \times \delta(d)) \\ 1 & if(d(i) - m(d) < -2 \times \delta(d)) \\ 0.5 - \frac{d(i)-m(d)}{4\times\delta(d)} & otherwise \end{cases}$$

Table V.3 lists the normalized values for the example in Table V.2. Obviously, a is the best instance using the above normalization. Standard deviation normalization works better in picking up the most balanced instance instead of those that are very strong in one aspect while too weak in another.

We thus apply the above normalization to every dimension in the *User Benefit* function (Equation V.1). It leads to $0 < d_i < 1$ and thus *user benefit* of a service falls into the range of (0 .. 1].

V.2.2 Utility Function

User benefit, together with service price, contribute altogether to the evaluation of a service, through a utility function:

$$\text{Service Utility} = \text{User Benefit} \times w_1 - \text{Service Price} \times w_2$$

where $w_1 + w_2 = 1$. Since both *user benefit* and *service price* fall into $[0..1]$, *service utility* falls into $[-1..1]$. By adjusting the value of w_1, the utility function

covers three variants of cost effective analysis (CEA) [Sassone, 1988]. The first variant is minimizing cost for a given level of effectiveness (i.e., *user benefit*). By setting $w_2 = 1$ and specifying QoS requirements in its service discovery request, a client can find the cheapest service for a given *user benefit* determined by its QoS requirements. The second variant is maximizing effectiveness for a given level of cost. By setting $w_1 = 1$ and specifying the reserve price in its service discovery request, a client finds the service with highest user benefit at an expected cost. The third variant is finding optimal tradeoff between effectiveness and cost. It can be realized by giving w_1 and w_2 various values other than 0. The flexibility in expressing different preferences makes the above utility function a better choice than benefit-cost ratio (e.g., [Venkatasubramanian and Nahrstedt, 1997]), since the latter allows for only one possibility (i.e., the ratio).

V.2.3 Vickrey Auction based Pricing Model

Using the above utility function, the service instance with the highest utility is chosen. If there is a tie, the winner is randomly chosen among the services with the highest utility. The paid price is then determined by a pricing model (Section III.3.2). We use *reverse* Vickrey auction to determine service price, because it is the service client that is "buying" a service, i.e., selecting among services. The winner gets paid with the price that would make its utility equal to the highest utility of all other instances. For example, the highest service utility is $su_1 = ub_1 \times w_1 - sp_1 \times w_2$, where ub_1 and sp_1 are *user benefit* and *service price* respectively. Given the highest utility of the other instances of su_2 ($su_1 \geq su_2$), the paid price equals $(ub_1 \times w_1 - su_2)/w_2$. Note that if the service utility is only determined by *user benefit* (e.g., w_1 equals 0) and thus independent of price, the paid price is equal to the asking price of the winner service.

Vickrey auction is applied here because it is executed in one round, different from other auction forms such as English auction, and thus does not incur further communication overhead. Every node has a dominant strategy (to be proved in the next paragraph) and thus the bidders (i.e. service providers) do not need to speculate about their competitors' strategies. Moreover, thanks to the incentive compatibility of Vickrey auction, service providers are motivated to reveal the truthful service cost, which ensures that a client's service selection is based on real service price. The service providers are motivated by the possible gain, since the winner is paid with a price higher than the overhead (i.e., cost). For the client, it may seem that it overpays as it pays according to the second lowest utility (thus higher than the winner's asking price) rather than the lowest utility. However, if the client pays according to

the lowest utility (e.g., using first-price sealed auction), the service providers are not motivated to bid the truthful price and ask for higher prices than in Vickrey auction. Moreover, such a selection achieves load balancing – because heavy load leads to high service cost and high bid, and thus low probability of winning the auction and being selected.

The above utility-based pricing model is not exactly the same as Vickrey auction, since the selection result is determined by service utility instead of service price. Despite of the difference, the dominant strategy for each service provider is still to bid the truthful overhead of the service, i.e., the service cost. It is proved in the following:

Lemma V.1. *The dominant strategy for a service provider in the above service selection based on Vickrey auction is to bid its truthful service cost.*

Proof. Assume a service provider A offers a service that has service cost of c and brings user benefit of b to the client. Let su be the highest service utility of any other service provider. If A wins the bid, it gets paid with $p = (b * w_1 - su)/w_2$ and thus has a net gain of $p - c$ (could be negative); if it does not win, it has a net gain of 0. We now analyze the bidding under three scenarios:

- $b \times w_1 - c \times w_2 > su$. If A asks for more than p, it does not win and gains 0. If it asks for less than p (including c), it wins the auction and has a positive gain of $p - c$. Thus, bidding c is strictly better than bidding more than p and at least as good as any bid less than p.

- $b \times w_1 - c \times w_2 < su$. If A asks for more than p (including c), it does not win and gains 0. If it bids less than p, it wins the auction and has a negative gain of $p - c$. Thus, bidding c is strictly better than any bidding less than p and as good as any bidding greater than p.

- $b \times w_1 - c \times w_2 = su$. If A asks for more than p, it does not win and gains 0. If it asks for less than p, it wins the auction and has a gain of $p - c$, which is also 0. Bidding exactly p leads to a tie. The winner will be randomly selected from those bidders. But either it is randomly selected or not, its payoff is always 0. Thus, bidding c is as good as any other bidding.

In summary, under all scenarios, bidding c brings the maximum possible gain for A. □

Therefore, utility based Vickrey auction motivates each service provider to reveal its truthful price, which is the resource consumption introduced by providing a service.

V.2.4 QoS-aware Service Location and Selection

QoS awareness can be integrated into not only service selection, but also service location process. Given a pull-based service discovery model, a client specifies its QoS requirements in the service request, as shown in Table V.4:

seq #	path(n_1, , ,)	range	destined SCNs	functionality	QoS requirements	price

Table V.4: A serv_disc packet for QoS-aware location and selection

Different from the `serv_disc` in S3L as introduced in the previous chapter, two fields are added into the packet:

- QoS requirements, which are in the form of $\prod_{1 \leq i \leq n}(d_i, v_i)$, meaning that the value of QoS dimension d_i $(1 \leq i \leq n)$ of the required service has to be no weaker than v_i. We use $\trianglerighteq$ to denote the relationship of "being no weaker than". The interpretation of being stronger (or weaker) depends on the specific dimension. A metric (i.e., quantitative dimension) can be stronger with larger value (e.g., availability), or with smaller value (e.g., latency). For a policy (i.e., qualitative dimension), supporting a policy is considered to be stronger than lack of such support. For example, a service supporting *confidentiality* is stronger than a service that does not support it, i.e., $(confidentiality = YES) \trianglerighteq (confidentiality = NO)$.
- Price, which is the highest acceptable price (i.e., reserve price) for the service client.

Before the client propagates the service request to its SCNs, it records the timestamp when the request is sent along with the sequence number, for the purpose of calculating service location latency. Then, the `serv_disc` is sent through stable links to the vicinity network defined by the propagation range. On receiving such a request, a node checks whether it provides any service that satisfies the QoS requirements posed by the client and at the cost lower than the reserve price. For example, a client has a QoS requirement of $(x_1, x_2, x_3, ..., x_n, p)$, with $x_1 ... x_n$ being the minimum values of QoS dimensions and p referring to the reserve price. A service provider has a service with QoS properties of $(v_1, v_2, v_3, ..., v_n, c)$ with $v_1 ... v_n$ representing the offered QoS values for the dimensions and c representing the service cost. The conformance checking is passed only if:

$$(x_1 \trianglerighteq v_1) \wedge (x_2 \trianglerighteq v_2) \wedge ...(x_n \trianglerighteq v_n) \wedge (p \leq c)$$

If the checking passes, the service provider sends back a reply along the embedded path, incorporating the QoS values of its service and its asking price, as shown in Table V.5. The field of *QoS values* is in the form of $\prod_{1 \leq i \leq n}(d_i, x_i)$, meaning that the service offers the value of x_i for QoS dimension d_i ($1 \leq i \leq n$).

seq #	path(n_1,n_2,...,n_k)	functionality	QoS values	price

Table V.5: A serv_resp packet for QoS-aware service location and selection

Whenever receiving a reply, the client extracts the sequence number and retrieves the timestamp it has noted down when it sent the service discovery request. The timestamp is then compared against the current time to obtain service location latency. The latter is then added to service latency advertised by the service provider to estimate the experienced latency for the client. After the timeout of waiting for service replies, the client selects the best one among the received replies. The timeout can be set with a length of time since the discovery request is sent or the number of received replies. For example, service selection is triggered after receiving 5 replies. These replies are evaluated using the utility function and the winner gets paid according to the Vickrey auction based pricing model.

Push based service discovery carries out service selection in a similar way. The only difference lies in the fact that the client already has all the service information at hand. It thus only needs to filter those that do not satisfy its requirements in terms of QoS and price and the remaining instances are selected based on the utility function as presented above. A possible issue is that it can be argued that, when publishing services to the SCNs, nodes tend to be reluctant to reveal true prices to potential competitors. However, it is safe to publish the price information because of the following two reasons: firstly, since the client's utility function is kept private, a service provider cannot determine its service utility, although it can change the utility of its service by manipulating its asking price; secondly, even after knowing price information of other services, it is always to the best interest of a service provider (i.e., its dominant strategy) to reveal its truthful price.

V.3 Service Selection Analysis

Our service selection is essentially carried out in two steps: (1) services are evaluated using the utility function; (2) the paid price is determined by a pricing model based on Vickrey auction. As the validity of the second step

is proved in Lemma V.1, we focus on the first step, i.e., the evaluation of service utility. Therefore, in the following, we analyze our approach to evaluate the overall service QoS.

name	Latency(0.8)		Availability(0.2)		QoS	Price	Service Utility
	orig.(in ms)	norm.	orig.	norm.		weight = 0.2	
s1	3.00	0.76	0.70	0.25	0.66	0.90	0.36
s2	4.50	0.47	0.90	0.75	0.52	0.60	0.26
s3	5.50	0.27	0.80	0.50	0.31	0.70	0.14

Table V.6: QoS values and utilities of three example services

Table V.6 shows three service instances *s1*, *s2* and *s3* with their QoS values (*orig.* and *norm.* denoting original and normalized values respectively). The relative importance of *latency* and *availability* are set to 0.8 and 0.2 respectively. Service utility evaluation is price-driven by giving a weight of 0.8 to price. $s3$ has the highest service utility and is thus selected.

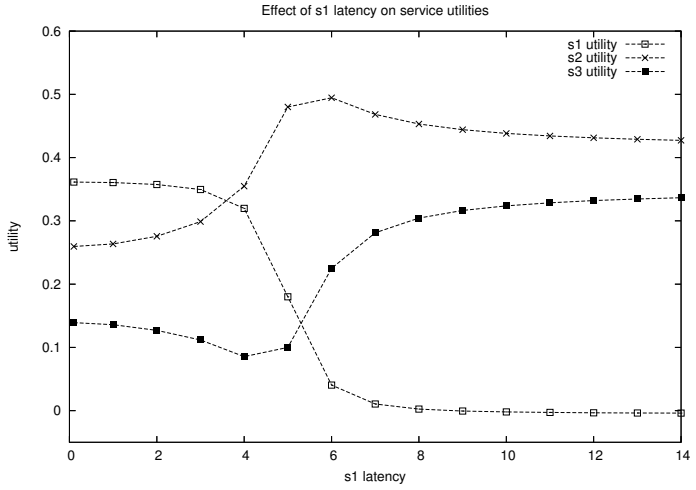


Figure V.1: Impact of a service's latency on others' service utilities

Recall that normalization of QoS values should not allow an exceptional value to overshadow others. Therefore, we investigate the relationship between one QoS dimension and the overall service utility. To do this, we study the impact of one QoS dimension on the service utility by adjusting service *s1*'s latency. In Table V.6, *s1*'s latency is 3.00, leading to a service utility of 0.36. By adjusting its latency from 0.1 to 15.00, the utilities of the three services are shown in Figure V.1. It shows that with latency varying over a large range

(between 0.1 and 15.00), *s1*'s utility does not fluctuate much (between about 0 to 0.36). Neither do the utilities of *s2* and *s3*. *s1* has the highest utility when its latency is less than about 3.5 ms. When *s1*'s latency exceeds 9 seconds, the service utilities become steady. It shows that with normalization based on standard deviation, a single dimension's value cannot dominate other values in affecting the evaluation outcomes.

V.4 Concluding Remarks

In this chapter, we have proposed a solution for service selection which includes: (1) evaluating each service with a utility function, taking into account service QoS, service price and client's preferences in a comprehensive manner; (2) using utility-based Vickrey auction to determine service price. Service selection chooses the best instance among the returned replies from service providers, on behalf of the client.

Despite Vickrey auction's impressive theoretical properties, Vickrey auction has the following two major shortcomings [Rothkopf et al., 1990, Sandholm, 1996]: the *fear of dishonest auctioneer* and the *reluctance of bidders to reveal their true valuation*. Since it is sealed, the winner can doubt whether the price the auctioneer tells it to pay is actually the second highest price. Therefore, fair execution of auctions needs to be guaranteed. Moreover, the valuation of goods or tasks are sensitive and private information that bidders are unwilling to reveal [Brandt and Weiβ, 2001, Rothkopf et al., 1990]. An approach for solving the above problems is to transform a single trustworthy entity into *a jury of trust* (e.g., in [Liu and Issarny, 2004c]) such that not every jury member needs to be trustworthy. Currently, there are two variations of this approach. The first is to include multiple auctioneers in the jury, most of which are assumed to be trustworthy. After each bidder sends shares of their bid to each auctioneer, only a majority of the auctioneers can open the bid with threshold computation (e.g., Verifiable Secret Sharing (VSS) [Pedersen, 1991]). The second approach is to include a semi-trusted third-party and an auctioneer as jury members. Fair execution of auctions and privacy of loser bids are guaranteed if the third-party does not collude with the auctioneer [Naor et al., 1999] or with any bidder [Baudron and Stern, 2001]. These approaches, however, incur numerous encryption operations.

An alternative to address the limitation of Vickrey auction is to use reputation, which is a much lighter-weight solution. Reputation is detailed in the next chapter.

VI

A Robust and Incentive Compatible Reputation Mechanism

In the previous chapter, we have presented a service selection method depending on the overall service QoS and price and the client's preference. Besides, service provider's reputation also needs to be taken into account during service selection. Reputation evaluation needs honest recommendations from others due to the probable lack of direct experiences.

Current reputation mechanisms (e.g., [Buchegger and Boudec, 2003, Huynh et al., 2005]) only focus on improving robustness against dishonest recommendations (i.e., rumors). They do not enforce incentive compatibility, i.e., entities are not motivated to recommend actively and honestly. In this chapter, we address this limitation by presenting a reputation mechanism that not only shows robustness against rumors, but also stimulates active and truthful recommendations. It achieves this by guaranteeing different treatment for different recommenders [Liu and Issarny, 2006]: the entities contributing more to the community by actively providing honest recommendations can benefit more from others, while rumor spreaders are identified and isolated.

In the rest of this chapter, Section VI.1 shows our representation of reputation based on Beta distribution. Then we explain how the reputation is formed based on direct and indirect experiences (i.e., recommendations) in Section VI.2. It is followed by the evolution of *Service Reputation* (SRep) and *Recommendation Reputation* (RRep) in Section VI.3. Then we proceed to present the propagation of reputation and the incentives for active and honest recommendation provision in Section VI.4. In Section VI.5, the reputation mechanism is evaluated with respect to its different treatment for recommenders of different honesty and activeness. This chapter finishes with concluding remarks in

Section VI.6.

VI.1 Reputation Representation

Since reputation essentially aggregates past experiences and dynamically evolves, it bears great similarity with Bayesian analysis, which is a statistical procedure that estimates parameters of an underlying distribution based on observations. Starting with prior distribution, which is the initial state before any observation is made, Bayesian analysis continuously takes into account new experiences and derives posterior probability [Casella and Berger, 2002]. An extensively used distribution in Bayesian analysis is Beta distribution.

VI.1.1 Beta Distribution

According to probability theory, posterior probability for binary events can be estimated by beta distribution. For example, given a process with two possible outcomes (T, $\neg T$), let r, s be the observed number of T and $\neg T$ respectively, the *Probability Density Function* (PDF) of the probability p of having the outcome T for the next time can be given by beta distribution (with $\alpha = r + 1$ and $\beta = s + 1$):

$$\begin{aligned} f(p|\alpha,\beta) &= \frac{1}{B(\alpha,\beta)} p^{\alpha-1}(1-p)^{\beta-1}, \text{ where } 0 \leq p \leq 1, \alpha, \beta \geq 0 \\ B(\alpha,\beta) &= \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt \end{aligned}$$

where α and β are two parameters used to index the continuous family of Beta distribution and $B(\alpha,\beta)$ is the beta function. $f(p|\alpha,\beta)$ represents a probability distribution of p in terms of integrals. Formally, the probability of p falling into $[a,b]$ is $\int_a^b f(p|\alpha,\beta)dp$. As p can only fall into $[0,1]$, $\int_0^1 f(p|\alpha,\beta)dp = 1$, referring to the trivial fact that p falls into $[0,1]$ with probability of 1. The prior distribution (the initial state) is $f(p|1,1)$, leading to uniform distribution (Figure VI.1). It reflects the fact that without any knowledge, the probability of having T for the next time can be any value between 0 and 1 with equal possibility. New observations are used to update the PDF of p. For example, having observed 8 times T and 2 times $\neg T$, the PDF can be expressed as $f(p|9,3)$, as plotted in Figure VI.1.

The expected (mean) value of the beta distribution $f(p|\alpha, \beta)$ assumes a simple form:

$$\mathrm{E}(p) = \frac{\alpha}{\alpha + \beta}$$

It gives the mean value of p, based on $(\alpha + \beta - 2)$ observations accumulated so far. For example, in Figure VI.1, the expected values of both $f(p|9, 3)$ and $f(p|21, 7)$ equal to 0.75. It can be interpreted as that the probability of observing outcome T in the future is uncertain, but the expected value is 0.75. In addition, $f(p|21, 7)$ has more confidence saying so (i.e., $f(0.75|21, 7) > f(0.75|9, 3)$), thanks to more accumulated observations.

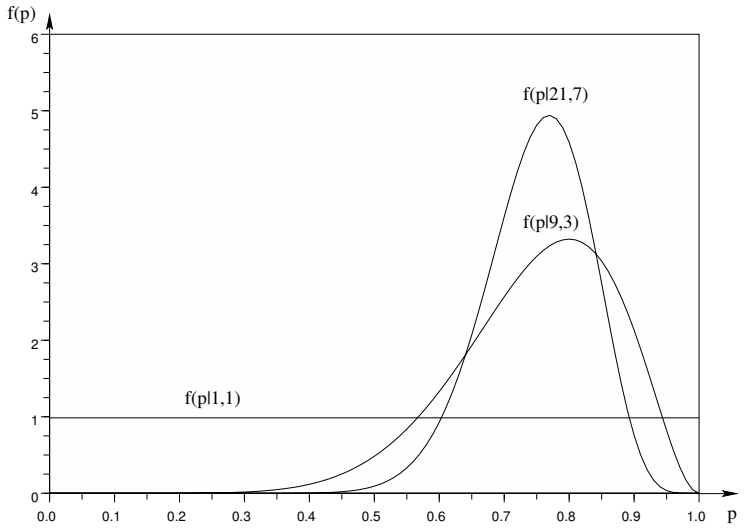


Figure VI.1: Beta Distribution values

VI.1.2 Beta Reputation

As reputation is essentially an *a posteriori* estimation based on historic experiences (either direct or indirect), beta distribution has been recognized as a useful model to model reputation [Mui et al., 2001, Jøsang and Ismail, 2002, Buchegger and Boudec, 2004]. Therefore, we represent reputation based on beta distribution (abbreviated as *beta reputation*). A reputation value assumes a tuple of (α, β) (α, $\beta \geq 1$), with α and β representing positive and negative experiences respectively.

As beta distribution only considers binary events, it is not enough to describe the experience of service consumption, which can fall into the range between being completely satisfactory and completely unsatisfactory. Specifically, an experience is evaluated with a *Quality of Experience* (QoE), saying, between 0 (completely unsatisfactory) and 1 (completely satisfactory). This experience is thus split into two parts: QoE contributing to the positive experience and $(1 - QoE)$ contributing to the negative experience. Therefore, beta reputation $f(p|\alpha, \beta)$ gives the PDF of *the probability of having a complete satisfactory experience*, i.e., the expected QoE.

Thanks to sound statistical properties of beta distribution, beta reputation has the following advantages:

(1) It is easy to assess the trustworthiness of an entity with reputation of (α, β), i.e., by calculating $\frac{\alpha}{\alpha+\beta}$.

(2) It is easy to evaluate how many experiences (i.e., $\alpha + \beta - 2$) have contributed to the current reputation. The larger this value is, the more probably the reputation assumes the expected value. Only newcomers' reputation is based on 0 experience.

(3) It facilitates the combination of experiences from multiple sources, including the trustor itself and different recommenders. For example, given two recommendations of $f(p|\alpha_1, \beta_1)$ and $f(p|\alpha_2, \beta_2)$, the combination of the two is $f(p|\alpha_1 + \alpha_2, \beta_1 + \beta_2)$ if they are considered to be of the same impact. In another word, the add operation of beta reputation is straightforward: $f(p|\alpha_1, \beta_1) + f(p|\alpha_2, \beta_2) = f(p|\alpha_1 + \alpha_2, \beta_1 + \beta_2)$.

(4) It reflects the nature of reputation, which is the aggregation of observations. An entity dynamically adjusts the reputation with more experiences being accumulated, which is similar to deriving posterior distribution after observations are made.

(5) It captures the uncertainty of reputation. Beta distribution only gives the PDF of the probability of having an outcome, which matches the fact that reputation only gives probabilistic estimation of an entity's future behavior.

Alternatively, reputation can be also represented with a single value from discrete (e.g.,*very trustworthy*, *trustworthy* .. in [Abdul-Rahman and Hailes, 2000]) or continuous value space (e.g., $[-1.. + 1]$ [Marsh, 1994]). Compared to beta reputation, single-value based reputation representation does not reflect the amount of experiences that contribute to the reputation. In addition,

with single value based reputation, *ignorance*, which refers to the reputation without any knowledge, generally bears the value of 0. It can not be distinguished from the 0 reputation values that result from a mixture of positive and negative experiences (e.g, [Marsh, 1994, Mui et al., 2002]). While with beta reputation, only newcomers have a reputation of $(1, 1)$.

Beta distribution's feature of easy experience aggregation facilitates the derivation of a node's reputation, which is formed based on the trustor's direct experiences and others' recommendations, explained as follows.

VI.2 Reputation Formation

Before we proceed to show how reputation is formed, we first explain the notations to be used in the reputation mechanism. As reputation is always about a node o (i.e., trustee) held by some node a (i.e., trustor), we denote o's reputation from the point of view of a as $Rep_a(o)$. Table VI.1 lists the notations we use, including service reputation (*SRep*), recommendation (*Rec*), recommendation reputation (*RRep*) and overall reputation (*ORep*). They are expressed using beta reputation, with two parameters representing positive and negative experiences respectively.

Label	Value Range	Meaning
$SRep_a(o)$	(s_p, s_n)	*a*'s direct experiences with *o*
$Rec_a(o)$	(c_p, c_n)	Recommendation made by node *a* regarding node *o*. Helpful recommenders give recommendations based on their own direct experiences, i.e., $Rec_a(o) = SRep_a(o)$
$RRep_a(o)$	(r_p, r_n)	Recommendation reputation of node *o* held by node *a*
$ORep_a(o)$	(o_p, o_n)	Overall reputation of node *o* held by node *a*

Table VI.1: Notations in the reputation mechanism

aID	SRep			RRep		
	s_p	s_n	t_s	r_p	r_n	t_r

Table VI.2: An entry of the acquaintance table

Each node keeps both *SRep* and *RRep* of its *acquaintances*, the entities with which it has interacted before (either as a service client or a recommendation requester), as shown in Table VI.2. This table of acquaintance records is named

acquaintance table. In the table, *aID* denotes the identity of the acquaintance and t_s and t_r represent respectively the timestamps when the $SRep(s_p, s_n)$ and $RRep(r_p, r_n)$ were updated last time. If the table is too small to accommodate all entities the node has encountered, some replacement policy is applied, e.g., records are purged depending on their ages.

ORep can rely solely on the trustor's direct experiences (i.e., *SRep*) if they are significant enough, i.e., the accumulated direct experiences are plentiful enough to derive a trust decision. This can be judged by checking whether the total accumulated $(s_p + s_n - 2)$ experiences reach a certain threshold. Otherwise, it asks for recommendations from others. The recommendations and the node's own direct experiences are then combined to evaluate the overall reputation (*ORep*) of the trustee.

Therefore, assume that a trustor *a* is evaluating the reputation of a trustee *o*. It has at hand service reputation $SRep_a(o)$ based on direct experiences, which are too few and thus trigger the elicitation of recommendations (i.e., $Rec_r(o)$, where r is a recommender) from others. Assuming a recommender r gives a recommendation regarding o (i.e., $Rec_r(o)$) to client c and $RRep_r(c) = (r_p, r_n)$, the recommendation is accepted (i.e., considered trustworthy) if (1) r is honest enough, by checking whether $\frac{r_p}{r_p+r_n}$ is high enough and (2) the *RRep* is evaluated based on enough evidence by checking whether $(r_p + r_n - 2)$ is large enough. If the recommendation is taken into account, it is given a weight w_r of $\mathrm{E}(Beta(r_p, r_n))$, i.e.,

$$w_r = \frac{r_p}{r_p + r_n}$$

The weights of different recommendations are further normalized by dividing with the sum of all weights. Therefore, *ORep* can be evaluated using *SRep* and the recommendations from helpful recommenders:

$$ORep = \delta \times SRep + (1 - \delta) \times \frac{\sum_{r \in R}(Rec_r(o) \times w_r)}{\sum_{r \in R}(w_r)} \tag{VI.1}$$

where δ is the weight given to its direct experience (*SRep*) and is generally greater than 0.5. The favor of direct experiences over recommendations is due to the fact that entities tend to rely on their own experiences more than on others' recommendations, as suggested by experimental studies of Kollock [Kollock, 1994]. Therefore, an entity can make a trust decision based on the overall reputation (*ORep*) of the trustee.

$ORep$ is not kept as a field of the acquaintance record, instead it is dynamically evaluated when needed, since it evolves with time and new experiences.

VI.3 Reputation Evolution

An entity can change its behavior over time, making old experiences become irrelevant for the actual reputation evaluation [Jøsang and Ismail, 2002, Liu and Issarny, 2004a]. This calls for discount of past, which gives more weight to recent experiences than old ones. Such discounting also prevents an entity from capitalizing on its previous good behavior forever. Hence, reputation fades with time, as shown as follows.

VI.3.1 Time Fading

Since both recent behaviors and past histories contribute to the reputation, their assigned weights decide how fast the reputation builds up. For example, if recent behavior is assigned a very high weight, an entity's reputation tears down very fast after a few misbehaviors. We assign more weight to recent behavior, as suggested by the results of psychological studies in [Karlins and Abelson, 1970] and empirical studies of ebay feedback mechanism [Dellarocas, 2003].

Given the time interval of Δt, the reputation (α, β) evolves after every Δt:

$$\begin{aligned} \alpha' &= 1 + (\alpha - 1) \times \rho^{\Delta T} \\ \beta' &= 1 + (\beta - 1) \times \rho^{\Delta T} \end{aligned}$$

where ρ is *time fading factor*, whose value falls into the range of $[0..1]$. The lower value ρ has, the more quickly histories are forgotten. When ρ equals 0, histories are immediately forgotten; while when ρ equals 1, the history is forever kept and considered equivalent regardless of age.

Note that a reputation starts with $(1, 1)$ and only the experiences (i.e., $\alpha - 1$ and $\beta - 1$) fade with time, making $\alpha + \beta > 2$ for any reputation value that takes into account an experience. But as shown in the above equations, when $\Delta T \to +\infty$, $\rho^{\Delta T} \to 0$, which expresses the fact that inactivity between two entities for a long time leads to complete discount of experience, making their reputations the same as that of newcomer. This is because when the experiences are too old to be indicative of the trustee's trustworthiness, they are useless. Both *SRep* and *RRep* fade according to the above equations. For simplicity, the reputation value in the rest of this chapter does not bear a timestamp and always refers to the current reputation unless indicated otherwise.

Reputation also evolves with new experiences, as reputation aggregates the overall experiences with an entity. This is reflected in both *SRep* and *RRep*, which aggregate the experiences of consuming services and utilizing recommendations respectively.

VI.3.2 Evolution of Service Reputation (SRep)

Since $SRep(s_p, s_n)$ combines all direct experiences, it is updated whenever a new experience occurs. An experience is described with a metric called Quality of Experience (*QoE*). As the goal of the reputation mechanism is to identify dishonest service providers that do not comply with their advertised QoS, *QoE* is accordingly measured based on the QoS conformance of the service provider. More specifically, given *n* QoS dimensions of d_i ($i = 1..n$) (e.g., *availability*, *latency*) which client *a* cares about, service provider *o* states in its service advertisement (p_1, p_2, .., p_n) in which p_i is the promised value for dimension d_i. After the service completes, the QoS that *a* receives is represented by (a_1, a_2, .., a_n), in which a_i is the actual value for dimension d_i. The $QoE_a(o)$ can be assessed by:

$$QoE = \sum_{1 \le i \le n} comp(a_i, p_i)/n \tag{VI.2}$$

where $comp(a_i, p_i)$ is a function to calculate one-dimension degree of conformance between the actual and promised QoS. Depending on the dimension, it assumes the following forms:

(1) $comp(a_i, p_i) = MIN(1, a_i/p_i)$ when dimension *i* is quantitative and stronger with larger values, for example, *availability*.

(2) $comp(a_i, p_i) = MIN(1, p_i/a_i)$, when dimension *i* is quantitative and stronger with smaller values, for example, *latency*.

(3) $comp(a_i, p_i) = 1 - (a_i \otimes p_i)$ when dimension i is qualitative and bears Boolean values, for example, confidentiality. $\otimes$ represents XOR function, i.e., $x \otimes y = 0$ if x equals y, and 1 otherwise.

(4) For dimensions whose value space is literals (e.g., *service adaptation*), $comp(a_i, p_i)$ equals 1 when the policy is satisfied, 0 otherwise.

For example, given a service provider's advertisement of (*latency* = 0.8 ms, *availability* = 99%), a service client's actual experienced QoS is (*latency* = 1.0 ms,

availability = 100%), then $QoE = (MIN(1, 0.8/1.0) + MIN(1, 100\%/99\%))/2 = 0.9$.

With a new QoE, the $SRep(s_p, s_n)$ is updated as in the following:

$$\begin{aligned} s'_p &= s_p + QoE \\ s'_n &= s_n + (1 - QoE) \end{aligned}$$

VI.3.3 Evolution of Recommendation Reputation (RRep)

Similarly, *RRep* dynamically evolves with new recommendations being elicited and new service consumption experiences. A recommendation bears the form of (c_p, c_n), which is equal to *SRep* for an honest recommender. Given a new *QoE* of $e \in [0..1]$, the honesty of recommender is adjusted according to the helpfulness of its recommendation.

$$\Delta e = \int_{MAX(0,e-0.4)}^{MIN(e+0.4,1)} f(p|c_p, c_n)dp$$

At first, Δe evaluates the probability of having a *QoE* in the range of $[MAX(0, e-0.4), MIN(e + 0.4, 1)]$, according to the recommendation of (c_p, c_n). As shown in Figure VI.2, if new experience *e* equals 0.8 and the recommendation is $(4, 2)$, Δe is equal to the size of the shaded area. It is compared against the probability if the trustor has no knowledge about the trustee, i.e.,

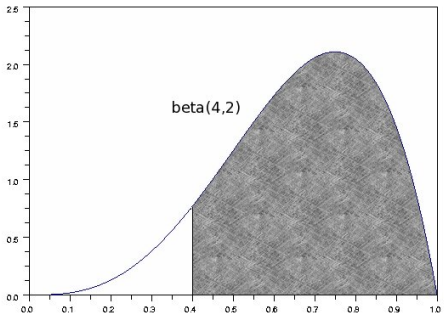


Figure VI.2: Calculation of Δe

$$\Delta min = \int_{MAX(0,e-0.4)}^{MIN(e+0.4,1)} f(p|1, 1)dp \qquad \text{(VI.3)}$$

Therefore, a recommendation with Δe larger than Δmin is considered helpful; otherwise it is regarded as unhelpful. $RRep(r_p, r_n)$ is updated accordingly:

$$\begin{aligned} e' &= MAX(MIN(\Delta e - \Delta min + 0.5, 1.0), 0.0) \\ r'_p &= r_p + e' \\ r'_n &= r_n + (1 - e') \end{aligned}$$

where *e'* represents the helpfulness of using the recommendation (c_p, c_n) (the MAX and MIN operators are used to ensure that *e'* falls into $[0..1]$). The recommender who provides helpful recommendations are considered honest, and dishonest otherwise.

Assume that before a client *c* has a new QoE (e) of 0.8 with service provider *o*, it has received recommendations of $(2, 4)$ and $(4, 2)$ from two recommenders *a* and *b* respectively. As $\Delta min = 0.6$ (using Equation VI.3), the helpfulness of *a*'s recommendation is $e' = 0.24$ and *b*'s recommendation leads to $e' = 0.81$. Thus helpful and unhelpful recommendations are distinguished. So are recommendations of different helpfulness. For example, a recommendation of $(4, 13)$ makes $e' = 0$, leading to the degrading of the recommender's *RRep* to a larger degree.

Based on the features of beta reputation, the value of $(r_p + r_n - 2)$ is high if an entity is active in providing recommendations; the expected value of $f(p|r_p, r_n)$ is high if an entity is honest in providing recommendation. This can be used to recognize whether an entity is active and honest in providing recommendations. With two values δ_h and δ_a defined as threshold trustworthiness and activeness in providing recommendations, a recommender with $RRep(r_p, r_n)$ is considered active if $r_p + r_n - 2 \geq \delta_a$, and inactive otherwise; it is considered honest if $\frac{r_p}{r_p + r_n} \geq \delta_h$, and dishonest otherwise. It leads to 5 possible states of a recommender: active truth-teller (AT), inactive truth-teller (IT), active liar (AL), inactive liar (IL) and newcomer (Figure VI.3).

A recommender can convert from one state to another, depending on its behavior. An active truth teller enforces its state by continuing recommending honestly and weakens its state by lying. If it keeps lying, with the fading of previous good behavior, the accumulated experiences eventually will work against it and degrade it to an *active liar*. Meanwhile, if a recommender has not provided any recommendation for so long a time that its *RRep* decays, it is considered as an inactive recommender (either *inactive truth-teller* or *inactive liar*). Long time of inactivity makes the reputation decay to 0 and the recommenders become *newcomers*.

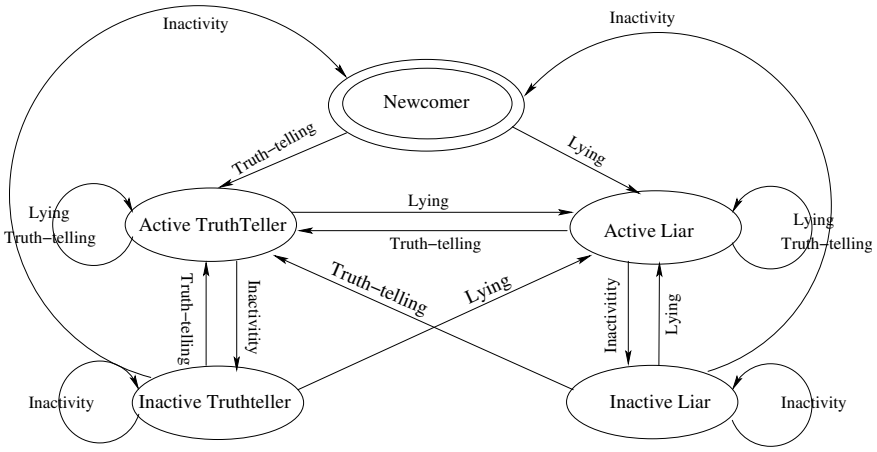


Figure VI.3: The states of a recommender

Note that an active and honest recommender (i.e., a recommender is very much willing to help) can be considered inactive due to the fact that it does not have any direct experience with the trustee being evaluated by the recommendation requester. Therefore, being *inactive* is only a state of a recommender. Although inactivity can result from an entity withholding recommendations on purpose, it does not necessarily infer free riding. But from the point of view of contribution to others, the order of helpfulness of different recommenders is AT > IT > IL > AL. In order to motivate an entity to become an active truth-teller, the entities who are more helpful should also benefit more from others. More specifically, active and honest recommenders should be able to have more success in identifying dishonest entities using the reputation mechanism. This is realized during reputation propagation, where different recommenders are treated differently in terms of accessibility to helpful recommendations.

VI.4 Reputation Propagation

Lack of enough direct experiences triggers a trustor's elicitation of recommendation from nearby nodes. It does so by broadcasting the request for recommendation to its neighbors, potentially including all types of recommenders. Of all the collected recommendations, only those from truth-tellers are taken into account. This corresponds to exogenous discounting of rumors, because the trustor's own experiences of using the recommender's recommendation (i.e., *RRep*) constitute external evidences other than recommendations. Recall that it is generally preferred over endogenous approaches to identify rumors,

because the latter assumes that honest recommendations dominate dishonest ones, which is unpractical in open environments.

If there is no such recommendation, the trustor takes into consideration those from inactive and first-time encountered recommenders by calculating the average. With the recommendations from others, the trustor evaluates the trustee's *ORep* using Equation VI.1. Otherwise, the trustor has to rely on its direct experiences which are too few to make a sound trust decision. The decision then has to be made depending on other factors, e.g., the trustor's attitude towards strangers. If the trust decision leads to a service consumption and thus a new Quality of Experience (QoE), the *QoE* is compared against all recommendations to update the recommenders' *RRep*s.

A trustor elicits recommendations indiscriminately but accepts only those from honest recommenders. This is for the purpose of ensuring robustness against rumors, while empowering them with the capability of recognizing new recommenders and continuously updating dishonest recommenders' *RRep*s. More specifically, even though the recommendations from dishonest recommenders are not taken into account, they are used to update the *RRep*s of the (dishonest) recommenders, which can be improving if they become honest or deteriorating if they continue lying.

When an honest recommender *a* receives a request for recommendations regarding an entity *o*, it first checks whether its direct experiences with *o* are significant enough for recommending. If that is not the case, *a* does nothing as it cannot be of any help. Some work (e.g., [Yu and Singh, 2002] and [Mui et al., 2002]) allows a chain of recommendations, i.e., an entity can recommend a recommender, who can send back recommendations or further recommend another recommender, until the depth limitation of the chain is reached. This practice is not incorporated in our reputation mechanism, because it requires introducing reputation of *recommending a recommender* in order to evaluate whether a recommendation about a recommender is truthful or not. It is different from *o*'s reputation of recommending a service provider (i.e., *RRep*), because one entity can be honest in recommending a recommender by giving its *RRep* in the recommendation, but can be dishonest in recommending a service provider by not giving truthfully its *SRep*. It thus requires the modeling of more reputations, which increases considerably the complexity of the reputation mechanism. Therefore, an entity in our reputation mechanism does not recommend recommenders.

Otherwise, if *a* has enough direct experiences for recommending, it handles the request for recommendations depending on the state of the recommendation requester:

- If the requester is considered as an active truth-teller, *a* sends back its $SRep_o(a)$ immediately.
- If the requester is considered as an active liar, *a* simply ignores the request.
- If the requester is considered inactive, *a* gives back its recommendation with a probability depending on *diff*$= \delta_a - (r_p + r_n - 2)$. Basically the smaller *diff* is, the higher probability (saying, $1-$*diff*) *a* sends its recommendation. The better treatment for inactive recommenders than liars is due to the fact that inactive recommenders do not necessarily withhold their recommendations. To distinguish inactive truth-tellers (IT), newcomers and inactive liars (IL), the IT and IL's probabilities are increased and decreased with an small value of ϵ respectively. Therefore, the less active an entity is, the less possible that it receives recommendations from others. Note that newcomers also suffer from low probability of eliciting honest recommendations.

Note that only honest recommenders go through the above process, which treats different type of recommendation requesters differently. The reason of doing this is that if honest recommenders are over-generous by treating everybody alike, other entities are not motivated to return the favor because doing that does not give them any advantage. Eventually, rational entities choose to withhold their recommendations while liars remain unpunished. Honest recommenders will suffer by having less and less useful recommendations from others and will eventually draw no favor back. It is similar to an ecological example given in [Dawkins, 1989], which explains the survival chances of birds who have to groom parasites for each other as they cannot clean by themselves. Dawkins introduces two types of birds into the system, "suckers" who always help and "cheaters" that have other birds groom parasites off their heads but never return any favor. They are both driven to extinction over time. But with the introduction of a third type of birds, named "grudgers" that start out being helpful, but hold grudges against the "cheaters". Dawkins shows by simulation that the grudgers survive and drive the other two species to extinction. Therefore, honest recommenders have to assume the defensive strategy of holding 'grudges' against liars by keeping others' *RRep*s, in order to guard their own interests. In contrast, dishonest recommenders, i.e., rumor spreaders, treat all types of recommendation requesters alike as their utmost goal is to spread rumors as widely as possible to take advantages, such as promoting their colluders' reputations.

The deterrent for nodes to spread rumors lies in the consequence of shutdown of supply of helpful recommendations from honest recommenders, who

hold grudges. At the same time, refusing to provide recommendations leads to less accessibility to helpful recommendations. Lack of recommendations forces a trustor's trust decision to be solely dependent on its direct experiences, which often can be too few or old to be helpful in open environments such as *USoCo* environments. This causes wrong trust decisions, making the client either interact with dishonest service providers or avoid honest ones. It is more clearly demonstrated in the next section, which evaluates the performance of our reputation mechanism.

VI.5 Reputation Mechanism Evaluation

In this section, we evaluate the performance of our reputation mechanism in helping nodes distinguish honest and dishonest service providers and to identify honest and active recommenders, based on simulations.

VI.5.1 Experiment Setting

The simulation is carried out with Network Simulator (ns-2) with CMU wireless extensions [LBNL, 2001]. The simulation parameters are shown in Table VI.3.

Parameter	Value
Mobility Model	Random Way Point
Moving Speed	0.5 - 1.5 m/s
Pause Time	0
Propagation Model	Ricean Fading
Transmission Range	100 m
Area	400m x 400m
Number of Nodes	40
Routing Protocol	OLSR

Table VI.3: NS-2 simulation parameters for reputation mechanism evaluation

Our experiment is set up with 40 nodes includes 8 types of entities with different behavior in service providing (honest or not), recommendation providing (honesty or activeness), as shown in Table VI.4. Each type of entity has the same population, i.e., 5 each (different settings with different population sizes will also be investigated).

Type	Service Providing Honesty	Recommendation	
		Honest	Active
1	+	+	+
2	+	+	-
3	+	-	+
4	+	-	-
5	-	+	+
6	-	+	-
7	-	-	+
8	-	-	-

Table VI.4: The types of nodes with different behavior

For simplicity without losing generality, we assume that every node can be the service provider for the other. Starting from time 50[1], every 1 second, a node (i.e., a service client) makes a trust decision regarding whether to interact with a random node (i.e., a service provider) in its routing table, in a round robin way. The trust decision is made as follows: (1) the service client first checks whether the *SRep* of the service provider has enough experiences to make a decision (threshold $\delta_a = 1.0$); (2) if yes, it calculates whether the expected value of *SRep* reaches a threshold value ($\delta_h = 0.6$); (3) if not, it elicits its recommendations from its neighbors (we set the request broadcast range to 2 hops). If the aggregation of *SRep* and others' recommendations are still not enough for making a trust decision, the node decides to whether to interact with a service provider with a certain probability. In our experiments, the probability is set to 1.0, assuming an optimistic attitude facing uncertainties.

A total of 60 rounds have been executed. An honest service provider offers a *QoE* of 0.9, while a dishonest service provider offers a *QoE* of 0.1. Honest recommenders recommend with its $SRep(s_p, s_n)$ regarding the trustee; while dishonest recommenders send back rumors which are complementary to the *SRep*s, i.e., a recommendation assumes the value of $(r_p = s_n, r_n = s_p)$. Active recommenders offer recommendations with 90% probability, while inactive ones offer with 10% probability.

We investigate and compare the performance of the 4 different types of recommenders: active truth-teller (type 1 + type 5 in Table VI.4), inactive truth-teller (type 2 + type 6), active liar (type 4 + type 8) and inactive liar (type 3 + type 7). The advantage of being an active truth-teller is reflected in the fact that they can elicit more honest recommendations, which help them make right

[1]This aims to give OLSR enough time to build routing table, as OLSR is a proactive protocol.

trust decisions regarding whether to interact with an entity or not. Therefore, we show (1) the number of honest recommendations obtained by the four types of recommenders respectively. When a client fails to acquire any helpful recommendation, it has to base its trust decision solely on its direct experiences, which are not significant enough for a sound decision. Namely, the client has to make a *blind decision*. Generally, the more likely an entity elicits honest recommendations, the less blind decisions it needs to make. We thus measure (2) the number of blind decisions made by the four types of nodes respectively. A blind decision can lead to a *mistake*, which refers to either a false positive (when an honest service provider is identified as an untrustworthy one) or false negative (when a dishonest service provider is not identified as being so). Thus, (3) the number of mistakes made by different recommenders are also displayed. These metrics are recorded every 200 seconds to show the evolution of reputation. They are detailed below.

VI.5.2 Evaluation Results

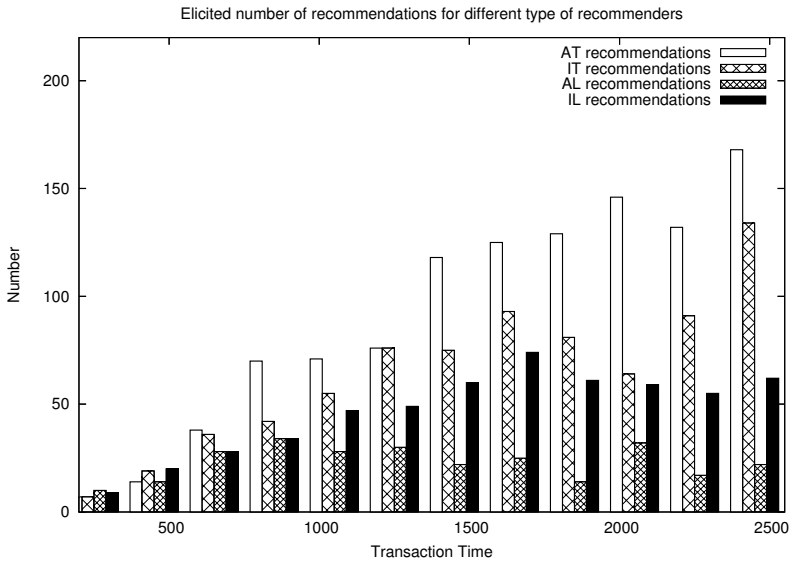


Figure VI.4: Number of elicited honest recommendations

Elicited Honest Recommendations. Figure VI.4 shows the number of elicited honest recommendations for different type of recommenders. It can be observed that at the beginning (before time $500s$), very few recommendations are propagated and the four types of recommenders do not have much differ-

ence in the number of obtained honest recommendations. With the accumulation of experiences, the honest entities have enough experiences to recommend. Recommendation reputation is gradually recognized and the order of benefit (AT > IT > IL > AL) starts to be established, from time 1500s in Figure VI.4.

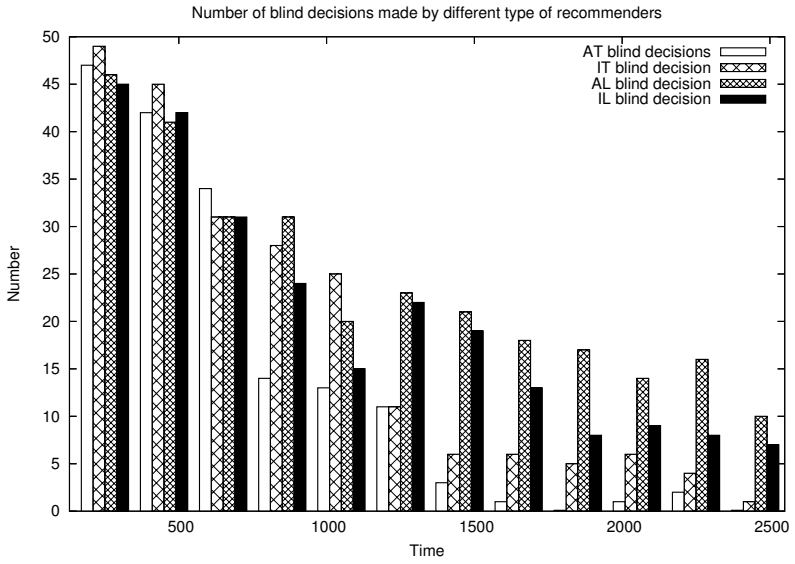


Figure VI.5: Number of blind decisions

Blind Decision. Lack of recommendation leads to blind decisions. Figure VI.5 presents the number of blind decisions for the four types of nodes. Note that during the span of 200 seconds (which is the interval between snapshots), 200 trust decisions are made, including 50 for each type of recommenders.

It can be seen that at the beginning, almost every trust decision for a node is blind due to lack of direct experiences and recommendations. With more accumulated experiences, the nodes make less and less blind decisions. Especially, AT nodes are exposed to the least number of blind decisions (less than 5 after time 1500s), while AL nodes suffer by making the most number of them.

Mistakes. Blind decisions can lead to mistakes. Figure VI.6 presents the number of mistakes made by the four types of recommenders. It can be seen that, at the beginning, every type of nodes make mistakes as many as half of the total transaction number. It is because most decisions are blind and honest service providers occupy half of the population.

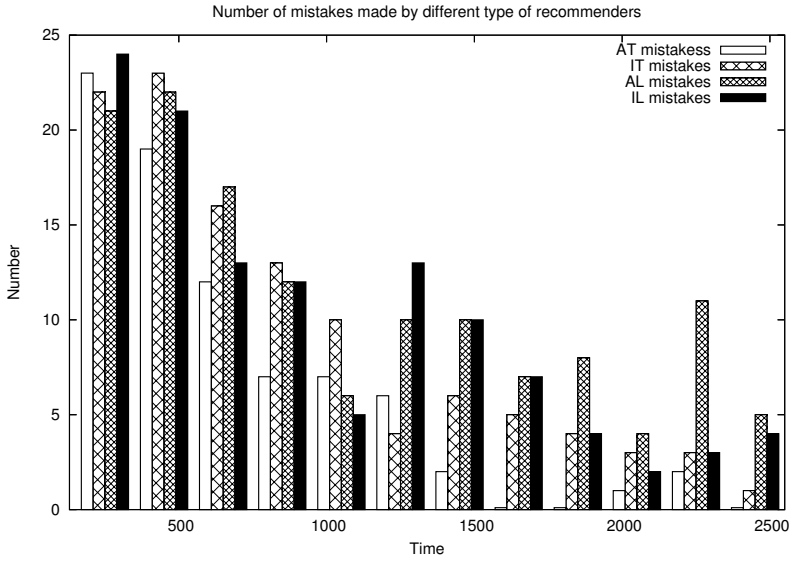


Figure VI.6: Number of made mistakes

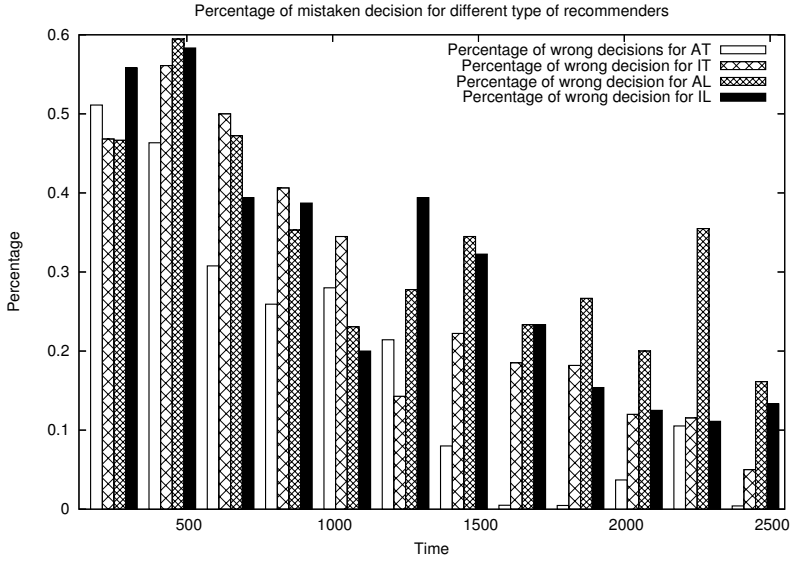


Figure VI.7: Percentage of wrong trust decisions

With more accumulated experiences, every type of recommender makes less and less mistakes. Especially, with the help of honest recommendations, AT nodes make the least number of mistakes and AL nodes make the most (the order of AT > IT > IL > AL is enforced). Note that dishonest or inactive

recommenders can also tell the honesty and activeness of a recommender using the reputation mechanism. However, they have access to less number of truthful recommendations for making right decisions.

In order to demonstrate more clearly the advantages brought by helpful recommendation, the percentages of mistakes out of all transactions for different recommenders are shown in Figure VI.7. It can be seen that, starting from time $1500s$, ATs make less than 5% of mistakes while ALs suffer more than 20% of mistakes.

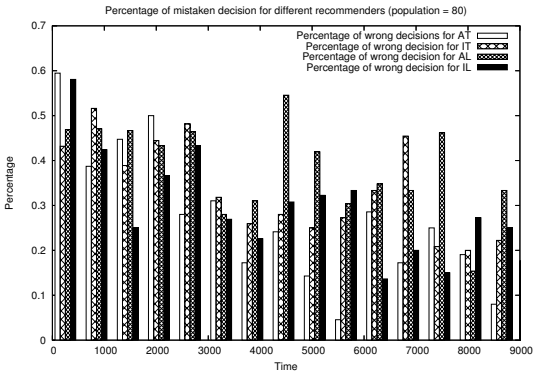


Figure VI.8: Percentage of wrong trust decisions with larger population

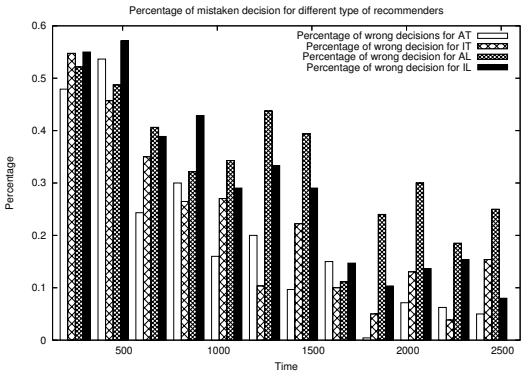


Figure VI.9: Percentage of wrong trust decisions with different population composition

Other Results. In the above simulation, we have set the population size to be relatively small (40) to lessen the time for bootstrapping (about 1500 seconds for 40 nodes), because nodes need to acquire experiences to be able to give useful recommendation. Basically, a larger population takes longer time to bootstrap, but the reputation mechanism shows similar effects. We did similar simulations with larger population (80) and the percentage of mistakes is shown in Figure VI.8. It can be observed that the order of benefit is established eventually, although it takes more time than in a community of 40 nodes (about 3500 seconds into simulation).

The reputation mechanism also exhibits similar performance with different population percentage of different recommenders. We have carried out the simulations by decreasing the population of active truth-teller (*AT*) to 10% and increasing active liar (*AL*) to 40%. The percentages of mistakes for different recommenders are presented in Figure VI.9, which shows that the order of the treatment (i.e., AT > IT > IL >AL) is also established.

VI.6 Concluding Remarks

In this chapter, we have presented a distributed reputation mechanism for recognizing the trustworthiness of a service provider in *USoCo* environments, including reputation representation, formation, evolution and propagation. Our contribution includes: (1) proposing a simple yet effective reputation mechanism that not only is rumor-proof, but also motivates active and truthful recommendation sharing; (2) modeling a reputation that continuous evolves, with time and with new experiences; (3) evaluating the effectiveness and performance of the proposed reputation mechanism via simulation tests.

As an entity has to handle the reputation independently and autonomously, in our reputation mechanism, it stores the reputation values of all acquaintances. This might raise an issue if the population of the acquaintances is so large that it brings considerable overhead in reputation storage and manipulation. A possible solution is to manage nodes by groups, each of which shares a common reputation [Sabater and Sierra, 2001, Mui et al., 2002]. The reputation of an entity depends on the group it belongs to; the behavior of a member affects the reputation of its group. This requires strong group support [Liu et al., 2005], as the group members need to trust each other and have common interests such that they are motivated to protect the group's reputation.

An important issue in reputation mechanism is identity changing. Most online reputation systems protect privacy and each agent's identity is normally a pseudonym. It causes problems because pseudonym can be changed easily [Zacharia and Maes, 2000, Mui et al., 2002]. When a user ends up having a reputation lower than that of a new comer, she can capitalize on the reputation system by discarding her initial identity and start from the beginning. This calls for the necessity of special treatments of newcomers. We partly address this issue by putting newcomers in a unfavorable position, such that they have difficulties obtaining helpful recommendations, until they accumulate enough good behavior. But active rumor spreaders can still benefit by restarting as newcomers, as the former suffers the total shutdown of helpful recommendation supply once they are identified as active liars. This problem can not be solved by simply lowering the newcomer's treatment to be even worse than all liars, such that the latter are not motivated to change for the worse. This is because in order to incorporate newcomers into the community, newcomers have to be given access to honest recommendations to be able to bootstrap. If active liars can have better accessibility to honest recommendations than newcomers, the deterrent for dishonest behavior hardly exists. Therefore, to solve this issue, it would have to rely on other mechanisms, such as introducing an "entry fee" for each pseudonym [Friendman

and Resnick, 2001] or use of once in a lifetime pseudonym that is bound to a real-world entity [Friendman and Resnick, 2001] or cryptographically generated unique identifiers [Buchegger and Boudec, 2002]. A very related issue is called Sybil attack [Douceur, 2002]: if there is no control over creation of new entities, a real-world entity can create as many identities as it wishes to challenge the use of majority in reputation systems. The only challenge this attack can bring to our reputation system is when there is no recommendation from an active truth-teller, the trustor relies on the average of all recommendations from unknown (or barely known) recommenders.

The trustworthiness evaluation via reputation mechanism enables identifying dishonest service providers. Along with signal strength based service location and QoS-aware service selection, they empower service discovery in *USoCo* environments with awareness and utilization of a service's QoS properties. They are integrated towards an overall solution for QoS-aware service discovery, as presented in the next chapter.

VII

QoS-aware Web Service Discovery Middleware

In this chapter, we present a QoS-aware Web service discovery (QoWSD) middleware, which gives an overall solution for discovering Web services in ubiquitous computing environments in a QoS-aware manner. *QoWSD* integrates our proposals described in the three previous chapters, i.e., service location based on signal strength (Chapter IV), QoS-aware service selection using Vickrey auction (Chapter V) and a robust and incentive-compatible reputation mechanism (Chapter VI). As interactions in ubiquitous computing environments are preferably carried out over Mobile Ad hoc NETworks (MANET) thanks to their flexibility and spontaneity, *QoWSD* middleware mainly targets MANETs. Web service is chosen as the underlying technology because it has been widely utilized and deployed thanks to the pervasiveness of Web Services Architecture in various environments [Issarny et al., 2005], including ubiquitous computing environments.

The rest of this chapter is organized as follows. Section VII.1 gives background information about the Web Services Architecture and related technologies. Then we detail the *QoWSD* middleware in Section VII.2. Section VII.3 evaluates the performance of a prototype implementing *QoWSD*. In particular, the overhead introduced by QoS awareness is measured and analyzed. This chapter finishes with concluding remarks in Section VII.4.

VII.1 Background on Web Services

A Web service, as defined by the W3C Web Services Architecture Working Group[1], is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [W3C, 2004b]. The main components of the Web Services Architecture [W3C, 2002] include WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol).

WSDL (Web Services Description Language) [W3C, 2001] is a declarative language for describing the interfaces of Web services. It separates the description of the abstract functionality offered by a service from concrete details of a service such as "how" and "where" that functionality is offered, as shown with two parts in Figure VII.1 (with WSDL terms marked with **bold** fonts). A *service* includes a set of *ports*, which associate network addresses with *bindings*. A binding is a concrete protocol and data format type that implements a *port type*, which includes a set of abstract *operations* supported by one or more endpoints. Each operation is an abstract description of an action supported by the service, which involves a named set of *messages*. Messages are abstract descriptions of the data being communicated. WSDL 1.1 defines syntactic signature for a service, but does not specify any non-functional aspects.

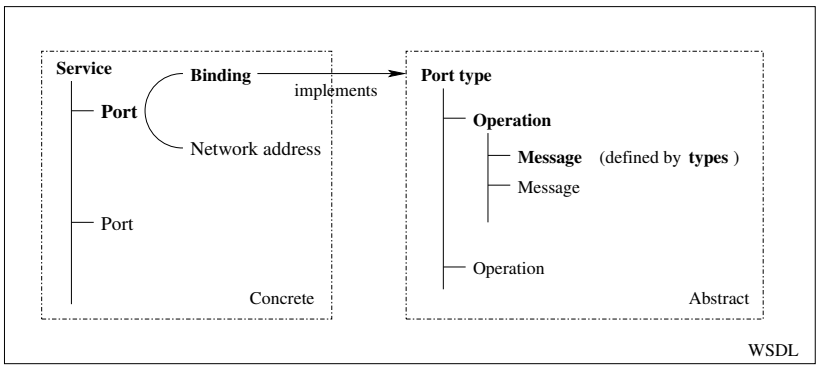


Figure VII.1: Structure of WSDL document

SOAP (Simple Object Access Protocol) [W3C, 2003] defines a lightweight protocol for information exchange. It uses XML technologies to define an extensible messaging framework, which provides a message construct that can

[1] http://www.w3.org/TR/ws-arch/

be exchanged over a variety of underlying protocols (e.g., HTTP, SMTP). A SOAP message includes an optional SOAP header and a SOAP body.

The Web services architecture is further complemented by UDDI (Universal Description, Discovery and Integration)[2], which is a specification of a registry for dynamically locating and advertising Web services. UDDI acts as a centralized directory, which is generally unavailable in environments featuring impromptu interactions, such as ubiquitous computing environments. In the next chapter, we present the *QoWSD* middleware supporting QoS-aware Web service discovery in a fully distributed manner.

VII.2 QoS-aware Web Service Discovery (QoWSD)

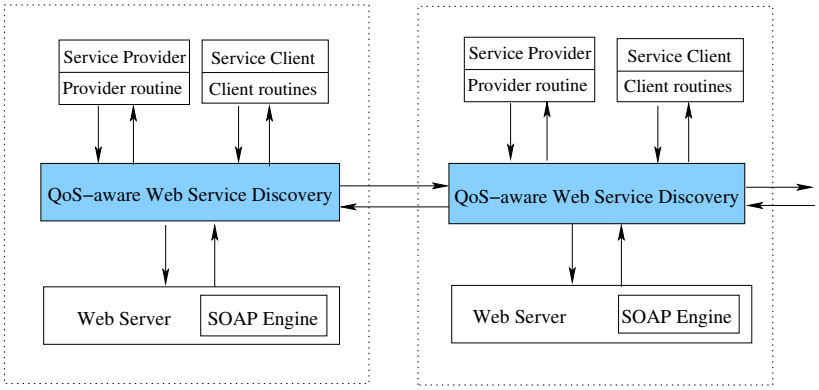


Figure VII.2: QoWSD Architecture

QoWSD is a standalone middleware instance running on every node, as shown in Figure VII.2. Note that the software entities in a dotted square in the figure are sitting on the same node. They are considered *local* to each other. Otherwise, if they reside on different hosts, they are considered *remote*. The *QoWSD*s communicate with each other using UDP packets.

QoWSD provides a routine that allows local service providers to register their services. A routine is an operation supported by *QoWSD* that can be invoked by applications. Meanwhile, clients can discover services, whether local or remote, using a routine provided by *QoWSD*. To distinguish between the above two, the routines used by service providers are named *provider routine*, in contrast to *client routine* used by service clients. Through the service discovery routine, a service client can invoke the services along the returned

[2] `http://www.uddi.org`

service paths. Since the support of specifying service path during service execution (i.e., source routing) can not be assumed to exist for all routing protocols, for the reason of completeness, *QoWSD* also allows a client to invoke services, which can be local or remote. Therefore, *QoWSD* provides routines for the client to discover and invoke services and a routine for the service providers to register services. For the purpose of illustration, *QoWSD* carries out pull-based service location, and the push-based alternative can be handled similarly. The supported routines are detailed and explained later in the section.

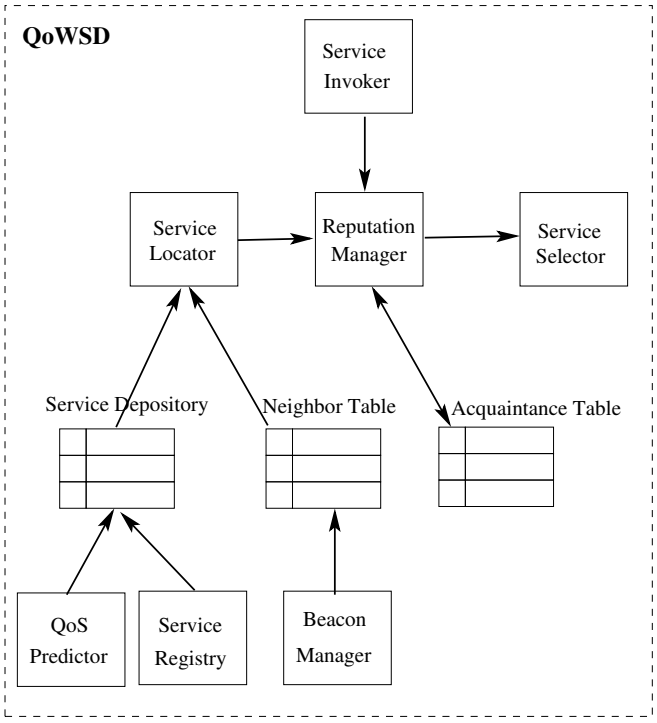


Figure VII.3: The internal structure of the QoWSD middleware

In order to support the above routines for service providers and clients, *QoWSD* includes components implementing *beacon manager*, *service registry*, *service locator*, *service selector*, *reputation manager*, *service invoker* and *QoS predictor*, as shown in Figure VII.3. *QoWSD* also manages three tables of *service depository* (for keeping registered services), *neighbor table* (for keeping the signal information of one-hop neighbors) and *acquaintance table* (for keeping reputation information of acquaintances). The arrows in the figure represent the data flow directions. More specifically, an arrow from a table to a component means that the component modifies the table while an arrow in the other direction means that the component only looks up the table (without any mod-

ification); an arrow from a component to another means that the former is executed before the latter.

In brief, as shown in the following pseudocodes, to discover services required by clients, at first the service locator elicits and collects service replies from providers, which are then forwarded to the reputation manager. The latter identifies the service replies from honest service providers. Finally, the service selector chooses the best instance on behalf of the client.

```
servReplies = servLocator.locateServices();
honestServReplies = repManager.identifyHonestServices(servReplies);
bestService = servSelector.select(honestServReplies);
```

As for other components,

- the beacon manager sends beacons and maintains the neighbor table;
- the service registry is responsible for registering the services from service providers;
- the service invoker invokes the services and records the direct experiences with the service provider, which is given to the reputation manager for updating the reputations;
- the QoS predictor is in charge of predicting the QoS values of the services based on histories.

The latter two maintain services' functional and QoS properties in the service depository respectively. These components and tables are detailed as follows.

Beacon Manager. The beacon manager periodically broadcasts and receives beacons to and from its one-hop neighbors. It builds and maintains a *neighbor table* (as explained in Chapter IV), which keeps the signal information of the links to its one-hop neighbors. Recall that an entry of the neighbor table (as shown in Table VII.1) includes the neighbor's IP address, the signal strength samples of the most recent $2 \times k$ beacons (for detecting signal strength tendency), the SNR samples of the most recent k beacons, the timestamp of the last received beacon, the average signal strength and SNR from the neighbor's end, the age of the last beacon received by the neighbor and an indicator stating whether this neighbor is a *Strongly Connected Neighbor* (SCN).

IP Address	$SS_{[1..2\times k]}$	$SNR_{[1..k]}$	TS	nSS	nSNR	nAge	isSCN

Table VII.1: An entry of the neighbor table in QoWSD

A beacon packet from a node gives the signal information of the links to its one hop neighbors. For each neighbor in the neighbor table, its IP, the average SS and SNR of the most recent k samples and the age of the last received beacon are extracted and calculated according to its entry in the neighbor table. They are embedded in the beacon messages, as shown in Figure VII.4.

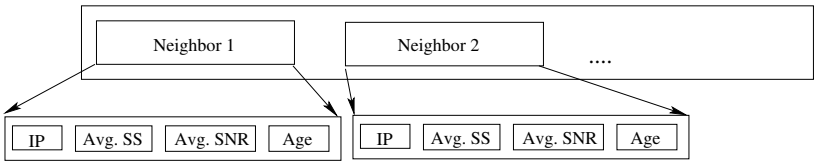


Figure VII.4: A serv_beacon packet in QoWSD

Meanwhile, the beacon manager monitors the signal strength of every link (e.g., using `iwspy` utility available on Linux), which gets updated whenever a packet is received. Signal information provided by the wireless driver (i.e., from the local end) and embedded in the received beacons (i.e., from the neighbor's end) are extracted and used to update the neighbor entry corresponding to the beacon sender, i.e., the fields of $SS_{[1..2\times k]}$, $SNR_{[1..k]}$ and TS are updated. If a beacon includes signal information of the link from the sender's end (i.e., the receiver is in the beacon sender's neighbor table), the fields of *nSS*, *nSNR* and *nAge* are also updated. Every some time (e.g., three beacon intervals), the beacon manager browses through the table and determines whether a neighbor is a SCN or not, depending on the strength and activeness of both link ends. The field of *isSCN* is then marked accordingly. It also deletes the entries of those expired links (e.g., the last beacon is older than three beacon intervals).

Service Registry. The service registry is responsible for registering Web services and maintaining a table of local Web services, i.e., *service depository*. A service provider can register its Web service(s) via `servRegister` of the provider routine shown in Table VII.2, giving the WSDL document describing its Web service(s). Since WSDL 1.1 does not support QoS properties, we extend the current WSDL document with a part for QoS description. It is called the *QoS part* of the WSDL document, to be distinguishable from *functional part* (i.e., the rest) of the WSDL document. The functional part can be further divided into the *abstract part* and the *concrete part*, as presented in Section

VII.1. An example of an extended WSDL document is shown in Figure VII.5. The tags of `<wsdl:definitions>` surround the functional part, with the abstract part referring to the definition of `<wsdl:porttype>`, while the concrete part includes definitions of `<wsdl:binding>` and `<wsdl:service>`. With the QoS extension, a service client can specify its service request in a WSDL document that includes an abstract and QoS part that state its functional and QoS requirements respectively. Meanwhile, a provider can specify its offered service in a WSDL document that includes a functional and QoS part, which give the concrete details such as data format binding and service deployment address as well as offered QoS values.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="urn:FormatConvertService"...>
   <wsdl:portType name="FormatConvertorInfo">
      <wsdl:operation name="getOtherFormat" parameterOrder="in0">
       ...
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="IFormatConvertServiceSoapBinding"...>
        ...
   </wsdl:binding>
   <wsdl:service name="FormatConvertorInfoService">
        ...
   </wsdl:service>
</wsdl:definitions>
<QoS>
   <Performance>
        <Latency>50</Latency>
   </Performance>
   <Dependability>
       <Confidentiality>true</Confidentiality>
   </Dependability>
   <Cost>
        <ResourceConsumption>0.55</ResourceConsumption>
   </Cost>
</QoS>
```

Figure VII.5: An example of extended WSDL document

```
public static void servRegister(FILE WSDLDocument);
```

Table VII.2: The routine provided by QoWSD for service providers

Whenever receiving a request of service registration from a service provider, the service registry parses the WSDL document and creates an entry of the service depository, as shown in Table VII.3, including:

- WSDL document, which is provided by the service provider as the parameter for the routine `ServRegister`.
- Definition of the service, which is the object resulting from the parsing of the functional part of the WSDL document (e.g., using WSDL4j utility[3]). It is utilized for matching functional properties of Web services.
- QoS, which stores the service's QoS values after parsing the QoS part of the WSDL document.

WSDL Document	Definition	QoS

Table VII.3: An entry of the service depository in QoWSD

Service Locator. A client uses the routine `servDiscover` (as shown in Table VII.4) to discover Web services. The parameter `seekServ` is set with the WSDL document (including the abstract part and QoS part) that specifies the client's requirements. The routine sends the parameter to the local *QoWSD*, which is then handled by the *service locator*.

`public static Vector servDiscover(String seekServ);`
`public static Vector servInvoke(String servPath,String servName,String opName,Vector opParams);`

Table VII.4: The routines provided by QoWSD for service clients

On receiving a service discovery request from a local client, the service locator does the following two things simultaneously:

- It parses the WSDL document from the client, with respect to both functional and QoS properties. The service locator then matches the request with every service instance in the service depository, in terms of both functional and QoS properties:
  - It is checked whether a service instance satisfies the client's request with respect to functional properties. More specifically, it is verified whether they match in terms of service name and port types' operations, including operation names and parameters (i.e., number and type of parameters).

[3]http://sourceforge.net/projects/wsdl4j

  - It is checked whether a service instance satisfies the client's request with respect to QoS properties. It is considered a matching only if, for every QoS dimension, the QoS value of the service instance is stronger than that required by the client.

  If a service instance satisfies the request in terms of both functional and QoS properties, it is appended to the list of service replies, each of which includes the service's WSDL document and the service path (i.e., the path from the client to the provider).

- It constructs a `serv_disc` message (Figure VII.6), which is sent to its SCNs (according to the neighbor table). The message includes the following fields:
  - sequence number, which is composed by the host's IP address and an increasing counter;
  - extended WSDL document, which is set to `seekServ` of `servDiscover` routine;
  - (propagation) range (in number of hops), which defines the range for propagating the `serv_disc`;
  - service path, which is initialized with the current host's IP address;
  - a list of destined SCNs, which is extracted from the neighbor table.

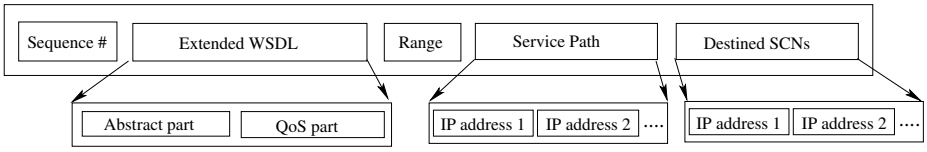


Figure VII.6: A serv_disc packet in QoWSD

On receiving a `serv_disc` packet from one of its SCNs, the service locator first checks whether the packet has been handled before by checking the sequence number against the cache of the sequence numbers of recently handled `serv_disc` packets. The packet is discarded if it has been handled before. Otherwise, it puts the sequence number into the cache and does the following two things in parallel:

- It matches the service request with every service instance in the service depository as described above. If there is a match, it unicasts a `serv_resp` packet (Figure VII.7) to the source node (i.e., where the client resides), which includes:

  - sequence number, which is set with that of the `serv_disc` packet;
  - WSDL document of its provided services that match the client's requirements, including both functional and QoS parts;
  - service path, which is set with the inverse of the complete service path (i.e., the service path of the received `serv_disc` packet appended with the current host's IP address).

- It extracts the propagation range and service path from the received `serv_disc` packet. If the range does not reach 0 yet, it forwards the packet with the following fields modified:

  - the range is subtracted by 1.
  - the service path is appended with the host's IP address.
  - the list of destined SCNs is replaced with the current host's SCNs that are not already in the service path.

  The forwarding is done by unicasting if there is only one destined SCN and by broadcasting otherwise.

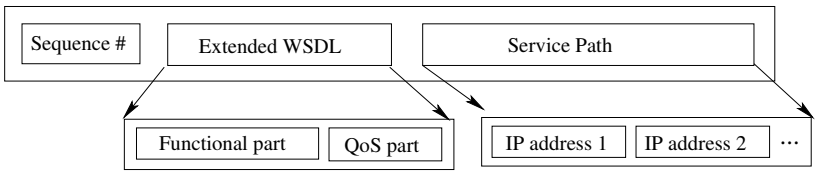


Figure VII.7: A serv_resp packet in QoWSD

When the service locator of the destined host receives the service reply (i.e., `serv_resp`), it extracts the extended WSDL and service path, which make a service reply.

Reputation Manager. After collecting the replies from service providers, the service locator hands the list of service replies to the reputation manager for identifying honest service providers. The latter also maintains an *acquaintance table*, which keeps the *SRep*s and *RRep*s of acquaintances. An entry of the acquaintance table is recalled in Table VII.5, where *ID* denotes the identity of an entity (e.g., MAC address), t_s and t_r represent respectively the timestamps when the $SRep(s_p, s_n)$ and $RRep(r_p, r_n)$ were updated last time.

Given the list of service providers for trustworthiness evaluation, the reputation manager handles it as follows. It first checks the *SRep*s of the evaluated entities. Recall that for a service provider with *SRep* of (α, β):

ID	SRep			RRep		
	s_p	s_n	t_s	r_p	r_n	t_r

Table VII.5: An entry of the acquaintance table in QoWSD

- It is considered *honest* if and only if $(\alpha + \beta - 2) \geq \delta_a$ and $\alpha/(\alpha + \beta) \geq \delta_h$, where δ_h is the threshold honesty value and δ_a is the threshold confidence value in making the judgment.
- It is consider as a *newcomer* if $(\alpha + \beta - 2) < \delta_a$.

If there exist one or multiple honest service providers, their expected reputation values (i.e., $\alpha/(\alpha+\beta)$) along with their IDs are returned. The returned reputation values allow for quantitative evaluation of the services. Otherwise, the reputation manager checks whether there is any newcomer. If that is not the case, it means that all the replies are from dishonest service providers and an empty list is returned. Otherwise, the reputation manager broadcasts a `rec_requ` packet to request for recommendations regarding the newcomers. As shown in Figure VII.8, a `rec_requ` is composed of:

- sequence number, similar to that of `serv_disc`, which is used to identify the request;
- (propagation) range of the request;
- the list of trustees to evaluate.

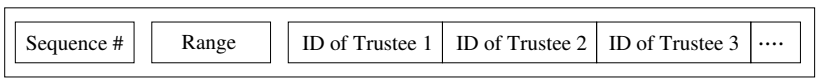


Figure VII.8: A rec_requ packet in QoWSD

After receiving a `rec_requ` packet, the reputation manager first checks whether the packet has been handled before according to the sequence number. It then looks up the $RRep(r_p, r_n)$ of the requester in the acquaintance table to determine how to handle the request. Recall that the requester is considered an active recommender if $(r_p + r_n - 2) > \delta_a$ and an inactive one otherwise; it is considered an honest recommender if $r_p/(r_p + r_n) > \delta_h$ and a dishonest one otherwise.

(1) If the requester is an active and honest recommender, the reputation manager looks up the *SRep*s of the trustees, which the requester is evaluating and returns them to the requester within a `rec_resp` packet (Figure VII.9), which includes

- Sequence number, which is set to that of the received `rec_requ`;
- *SRep*s of the trustees.

(2) Otherwise, if the requester is an inactive recommender, the reputation manager sends back its recommendations with the probability of $\delta_a - (r_p + r_n - 2) + \epsilon$, where ϵ is set to 0.05 for inactive honest recommenders and -0.05 for inactive dishonest ones, for the purpose of distinguishing their treatment. The recommendations are embedded in a `rec_resp`, as shown above.

(3) Otherwise, i.e., the requester is an active and dishonest recommender, its request is ignored.

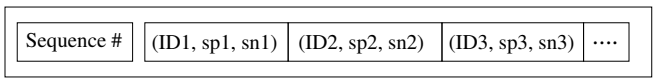


Figure VII.9: A rec_resp packet in QoWSD

After the timeout of recommendation elicitation, the reputation manager takes into account the received recommendations (i.e., `rec_resp`s) depending on the *RRep*s of the recommenders:

(1) If there exist recommendations from active and honest recommenders, they are used to evaluate the overall reputation of the service providers. A recommender's recommendation is weighed with the expected value of its *RRep*.

(2) Otherwise, if there exist recommendations from inactive recommenders, the average of those recommendations is used to evaluate the overall reputation of the service provider.

(3) Otherwise, it means that all the recommendations are from dishonest recommenders and the client fails to elicit any honest recommendation. Thus its trust decision has to be based on other factors (e.g., whether it is optimistic with strangers).

The reputation manager is also responsible for updating and maintaining *SRep*s and *RRep*s. A new interaction experience with a service provider is evaluated by the service invoker (to be described soon) and sent to the reputation manager. The latter then updates the *SRep* of the concerned service provider. Meanwhile, the *RRep*s of the recommenders (if any) who have recommended the service provider are updated accordingly, depending on the difference between the recommendation and the actual experience (as presented in Chapter VI).

Service Selector. Given a list of service replies, the reputation manager as presented above returns a list of trustworthy service providers along with their reputation values. If the list is empty, meaning that the providers of all located service instances are dishonest, the client is thus informed and advised to search with other requirements (e.g., with less strict requirements). If the list includes only one service provider, its service reply is returned directly to the client and no further selection is necessary.

Otherwise, the service selector parses the QoS part of the WSDL documents embedded in the service replies to extract the QoS properties and service price. For the reason of simplicity, we assume that the utility function of the client, i.e., the client's preferences among service QoS properties (e.g., preferences between *latency* and *availability*) and between QoS and price (e.g., QoS-driven or price-driven) is fixed and known *a priori* by the service selector. It can be easily extended by requiring the client to specify its utility function as a parameter of the `servDiscover` routine. Then all quantitative dimensions in the utility function are normalized based on standard deviation as presented in Chapter V. Each service instance is then evaluated using the utility function. The instance with the best utility is chosen and its WSDL document is returned, along with its service path, as the result for the `servDiscover` routine.

Service Invoker. A client invokes services using the routine `servInvoke` provided by *QoWSD* (Table VII.4). The parameters of the routine include the service path, the service name, the operation name and the parameters, which are received by the service invoker of the local *QoWSD*.

The service invoker handles the request in the following way:

(1) It checks whether the service path only includes the local IP address, i.e., whether the client is invoking a local service. If yes, it invokes the service with the parameters and returns the result to the client.

(2) Otherwise, it prepares a `serv_invo` packet (Figure VII.10), including the fields of

- sequence number, which is used to identify this invocation;
- service name of the target service to invoke;
- operation name of the target service to invoke;
- parameters for invoking the operation;
- service path that leads to the destined service provider.

The packet is then sent to the next hop in the service path.

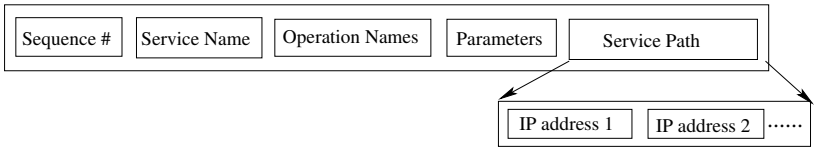


Figure VII.10: A serv_invo packet in QoWSD

On receiving a *serv_invo* packet, the service invoker handles it as follows:

(1) It extracts the service path in the packet and checks whether it is the destination of the packet. If not, it forwards the packet to the next hop in the service path.

(2) Otherwise, it invokes the service and send back the service result. It does this by preparing a `serv_ack` packet (Figure VII.11), which includes

- sequence number, which is set to that of the received `serv_invo` packet;
- service result, i.e., the result after executing the service;
- service path, which is the reverse of that of the received `serv_invo` packet. It is used to forward the result back to the service client.

The `serv_ack` packet is then sent to the next hop in the service path and forwarded by intermediate nodes until it reaches the destination *QoWSD*, where the the service client sits. The result is then returned to the client.

After invocation, the client also records its experienced QoS and compares it against the claimed QoS by the service provider. It then calculates QoE (Quality of Experience) according to Equation VI.2 presented in Chapter VI. It

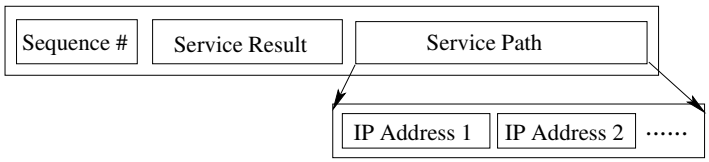


Figure VII.11: A serv_ack packet in QoWSD

is sent along with the service provider's ID to the reputation manager, which in turn updates the *SRep* of the service provider and the *RRep*s of the recommenders who have recommended the service provider, as described previously.

QoS Predictor. After a service invocation, the service provider records the QoS values, such as CPU load. Meanwhile, the service invoker also records the service latency, the time between when the invoker invokes the service and when it receives the result. These QoS values are sent to the QoS predictor, which is responsible for predicting future QoS values. It does this by applying a mix-of-experts approach [Wolski et al., 1997, Gurun et al., 2004] using the following three forecasters[4]:

- *last value*, which predicts the future value with the last measurement;
- *sliding window average* of window size 10, which predicts with the average of the most recent 10 measurements;
- *exponential smoothing forecaster*, which predicts a QoS value at time *t* (p_t) with $p_t = \alpha \times m_{t-1} + (1 - \alpha) \times p_{t-1}$, where m_{t-1} and p_{t-1} represent measurement and prediction values at time *t-1* respectively, and α is gaining factor and assumes the value of between 0 and 1.

Whenever a prediction is requested, these forecasters are ranked according to their prediction deviation, i.e., the difference between the measurement and the predicted values, during the most recent 10 runs. The most accurate forecaster is then chosen to predict the next QoS value. This process is executed whenever a prediction value is needed.

[4]Other forecasters can be added for more accuracy, but at the cost of more time taken for predicting.

VII.3 QoWSD Prototype

We have developed and deployed a prototype implementing *QoWSD*. In the following, we first give an overview of the prototype in Section VII.3.1. Then in Section VII.3.2, we evaluate the performance of the prototype.

VII.3.1 Prototype Overview

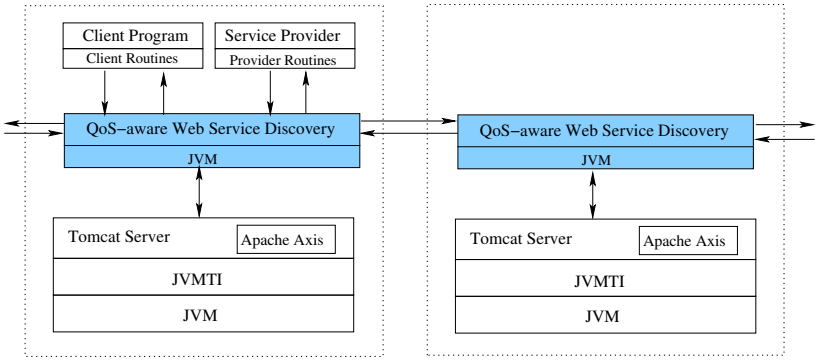


Figure VII.12: QoWSD Prototype Architecture

The prototype is developed using JAVA (J2SE 1.5) and deployed on 6 laptops with 500 MHz Pentium III CPU, 256KB of cache and 192 MB of memory, running Linux 2.6.8 (Mandrake 10.1). These machines are believed to be well-suited to estimate the performance of our middleware, as they are almost as powerful as currently portable devices (e.g., the SHARP Zaurus SL-6000 is equipped with Intel 400MHz processor and 64 MB RAM; the HP iPAQ hx2495 Pocket PC is equipped with Intel 520MHz processor and 64 MB RAM)[5].

As shown in Figure VII.12, the local Web services are deployed on Tomcat Server version 4.1.31[6] using Apache AXIS SOAP engine version 1.2.1[7]. They are based on J2SE 1.5 and JVM Tool Interface (JVMTI)[8], which gives an interface for performance profiling (e.g., measuring CPU load). Note that *QoWSD* middleware is independent of the Web application server and SOAP engine where the local Web services are deployed, because *QoWSD* interacts with the local Web application server only for invoking Web services through the interfaces which are given by service providers.

[5]The prototype is currently being ported to PDAs.
[6]`http://tomcat.apache.org/`
[7]`http://ws.apache.org/axis/`
[8]`http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/`

The wireless interface is 2.4 GHz DS Lucent IEEE 802.11 Wavelan PC "Silver" of 11 Mbps. The network topology assumes a line as shown in Figure VII.13, for the purpose of setting up the scenarios with different network distances (in hop number) between the service client and provider (from 1 to 5). Each node is connected only to its two neighbors (except the two ends have only one neighbor). Others are disconnected using IP filtering (e.g., iptables[9]), although they are in the communication range of each other. Therefore, at any point of time, only one node can send any packet, whether a beacon or any other packet as described above. The network is thus more congested than other multihop networks, where packets can be sent simultaneously as long as they are far enough from each other (e.g., 3 hops away). No routing protocol is deployed to avoid the effect of different routing protocols. Therefore, the unicast packets are forwarded by the intermediate *QoWSD* instances.

Figure VII.13: The network topology for QoWSD prototype evaluation

VII.3.2 Performance Evaluation

In the following, we evaluate the performance of the prototype in terms of overhead of QoS awareness on service discovery latency, the time between when the client invokes the routine of `servDiscover` and when it receives the reply. With *QoWSD*, service discovery latency can be decomposed into three parts:

- the time spent by the service locator, i.e., *service location latency*, which starts from when the local service locator of the client sends a `serv_disc` until the timeout of waiting for service replies. The length of the timeout is generally set according to the waiting time for a service reply to arrive. For example, a short waiting time for service replies leads to a short timeout. Therefore, we investigate the service location latency with timeout set as the waiting time for only one service reply.

- the time spent by the reputation manager, which checks the reputation of service providers. It includes the lookup of reputation values in the acquaintance table and possibly involves recommendation elicitation time, if recommendations are needed.

[9] `http://www.netfilter.org/`

- the time spent by the service selector, which chooses the best among service replies (from honest service providers).

In contrast, for service discovery that is "QoS-unaware", the service discovery latency only involves the first part and the services are randomly selected and reputation is not checked. In addition, its service location latency is smaller than that of *QoWSD*, since no QoS description or beacon is used. Therefore, the overhead of introducing QoS-awareness is reflected in (1) the increase of service location latency; (2) reputation checking time and (3) service selection time.

The efficiency of using signal strength to detect node mobility and the improvement of service reliability using S3L have been presented in Chapter IV. Therefore, we only evaluate the performance of the prototype when the devices are stationary. In the following measurements, a client sends a request for service only after it has received the reply for the previous request. The experiments are carried out 500 times and the average is presented.

Service Location Latency. We first measure the service location latency. The experiments have been carried out with service providers at different network distances (i.e., from 1 hop to 5 hops) and with beacons sent at every 2 seconds (beacon's effect on the service location latency will be discussed later). For the purpose of illustration, the functional part of the WSDL document includes 1 operation which has an input parameter; the QoS part of the WSDL document specifies 11 QoS dimensions, including both quantitative (e.g., *latency*) and qualitative ones (e.g., *confidentiality*). The service depository for each *QoWSD* instance has only one service, such that the client only needs to match the service request against one service instance. This is for the purpose of avoiding the variation of latency due to the position of a qualified instance in the service depository (e.g., if a qualified instance is at the first position, it takes less matching time than if it is at the 10th position).

In order to investigate the overhead, we also present the breakdown of the latency, including (1) WSDL parsing time spent by the service locator parsing the WSDL document embedded in the client request, including the time for parsing both functional and QoS parts; (2) WSDL comparison time for matching the client's request with service instances in the service depository, in terms of both functional and QoS properties; (3) time for message sending and receiving between a client and its local *QoWSD* and between *QoWSD*s; and (4) other time (e.g., OS overhead, etc). Table VII.6 shows the service location latency and its breakdown with service providers at different network distances in hops. Basically, WSDL parsing time and comparison time do not

increase with larger network distance because the service locator propagates the discovery request and compares a client's request against its local services at the same time. Therefore, the waiting time for a service reply only includes the WSDL parsing and comparison of the matched service (i.e., in the service reply). In addition, the WSDL comparison time is generally negligible (less than 1 millisecond) because it only involves the comparison of the operations and QoS dimensions, which are fields of objects (i.e., definition and QoS fields in the service depository) stored in the service depository. It can also be observed that the time for sending and receiving messages is the main contributor of service location latency, which obviously increases with larger network distance between the service client and provider. And as it involves with more nodes with larger network distance, the overhead such as those related to OS (e.g., message waiting time in the buffer) also increases. The result shown in Table VII.6 is used as the basis for evaluating the overhead of introducing QoS awareness on service location latency.

Hop Number	1	2	3	4	5
WSDL Parsing time	22	22	21	23	24
WSDL Comparison Time	< 1	< 1	<1	<1	<1
Message sending/receiving time	44	57	71	87	102
Others	1	13	19	34	44
Total Time (ms)	67	92	111	144	170

Table VII.6: Service location latency and its breakdown

Overhead of QoS Description. *QoWSD* takes into account QoS description of Web services, leading to the overhead in terms of larger message size (such as `serv_disc` messages) and the additional time required for QoS parsing and matching. We study its overhead in terms of service location latency.

Figure VII.14 compares the service location latency with and without introducing QoS description, for service providers at different network distances. It is measured when beacon is sent every 2 seconds. It can be seen that the introduction of QoS description increases the latency by from 4 to 10 ms, which is less than 6% increase over the service location latency without QoS description.

Overhead of Beacon. As beacons are introduced in order to evaluate link stability, we also study the overhead of beacons by comparing the service location latency with and without using beacons. The comparison is done both

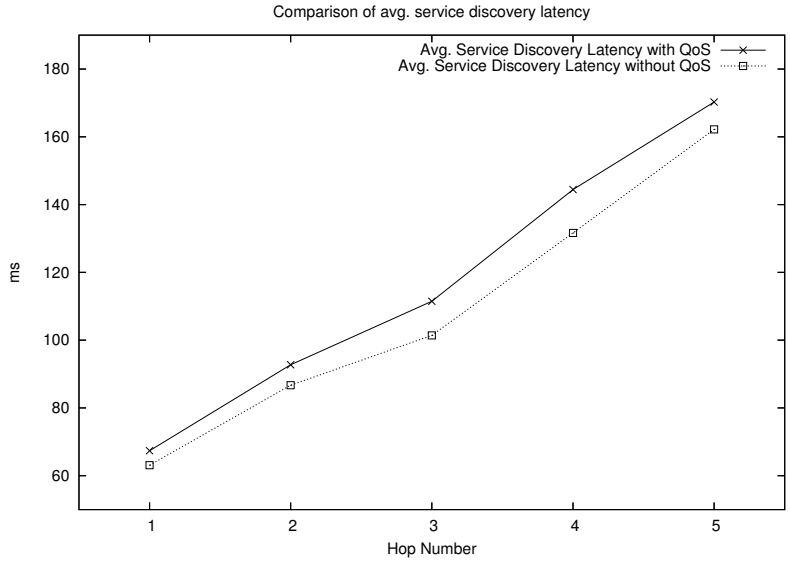


Figure VII.14: Service location latency with and without QoS description

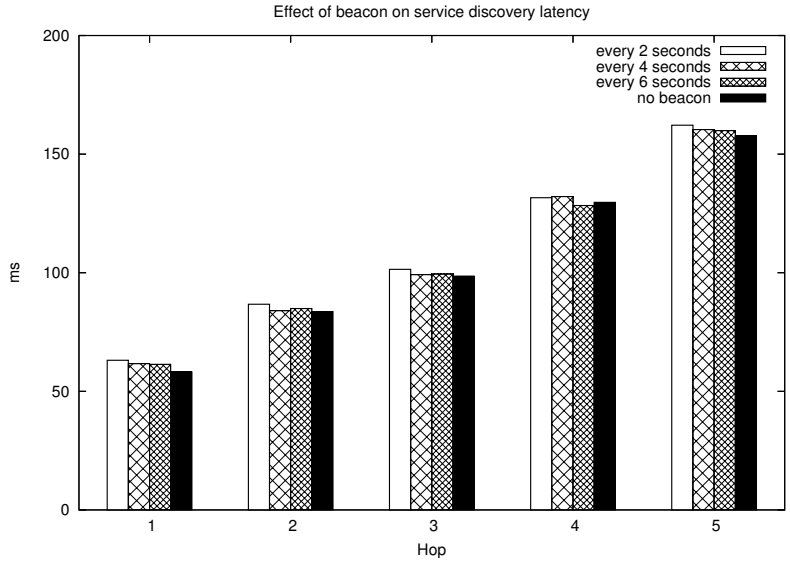


Figure VII.15: Impact of beacons on service location latency

with and without introducing QoS description, which show similar results. We thus present the service location latency without QoS description, with different beacon frequencies. The beacon frequency is set to every 2, 4, 6 seconds and no beacon. Beacons' impact is reflected in the created congestion, since at any point of time, only one packet can be sent in the network. Their

impact on service location latency is shown in Figure VII.15. It can be observed that the beacons increase the service location latency by from 3% to 8%.

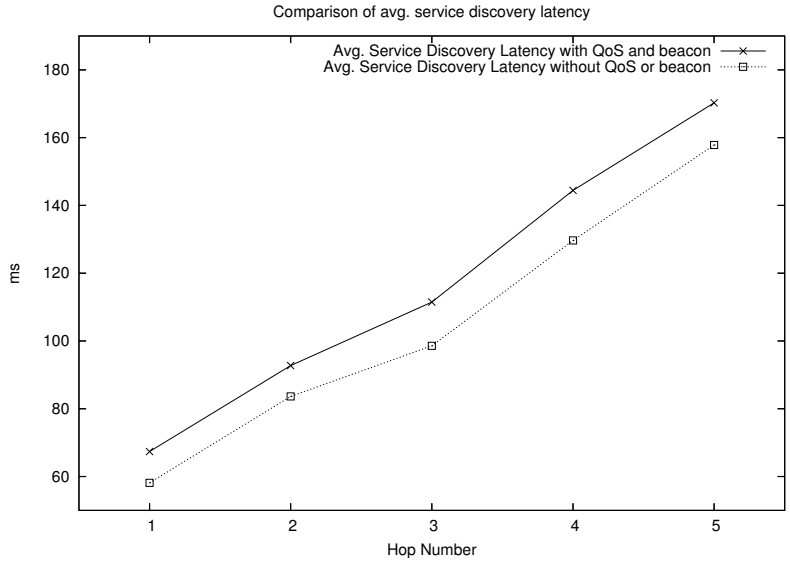


Figure VII.16: Impact of beacons and QoS description on service location latency

The total overhead brought by QoS description and beacons is shown in Figure VII.16. It shows that the beacons and QoS description increase the service location latency by about from 8% to 15%.

Reputation Checking Time. The reputation manager looks up the reputation of service providers to evaluate. If there exist enough direct experiences with a service provider in the acquaintance table, the reputation checking time is short because it only involves lookup of the table. Otherwise, the reputation manager asks for recommendations from others in the network vicinity. Similar to service location, the recommendations are aggregated after a timeout, which is defined in terms of the length of time (e.g., after 0.5 second). The length of timeout is set according to the waiting time for a recommendation, named *Recommendation Elicitation Time* (RET). It is defined as the time between when the reputation manager sends a request for recommendations and when it receives one. *RET* poses as the main overhead of reputation checking time since it involves message transmission between nodes. Figure VII.17 shows the *RET* with recommenders at increasing network distances. The service location latency is also presented for the purpose of comparison. It can be seen

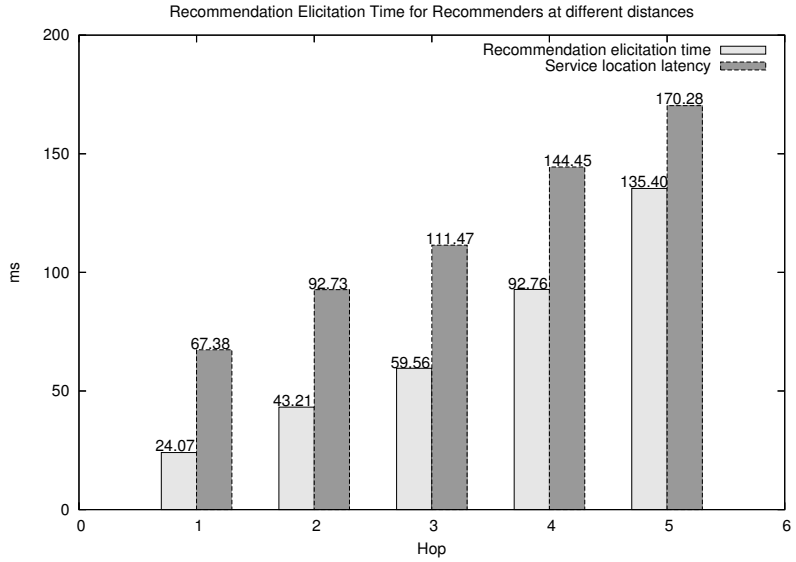


Figure VII.17: Recommendation elicitation time

that if a service is located within 1 hop, the recommendation within one hop increases the latency by 36%, 46% for 2 hops and 88% for 3 hops. It suggests that recommendation elicitation poses a considerable overhead and should be carried out within a small range (e.g., within 1 or 2 hops) to avoid too large overhead. Meanwhile, if a service is located within larger range (e.g., more than 1 hop), the range for recommendation elicitation can be accordingly expanded.

Service Selection Time. With the service replies from honest service providers (i.e., after reputation checking by the reputation manager), the service selector needs to choose the best one among them. Recall that a service reply includes a WSDL document and service path. Therefore, the service selector needs to (1) carry out QoS parsing to extract QoS values and (2) evaluate all the service instances using the utility function. Thus, the service selection time, the time taken by the service selector to choose the best instance, includes the time spent for QoS parsing and QoS computing (i.e., normalization and utility calculation). Table VII.7 shows the service selection time (in ms) with different number of service instances and QoS dimensions. It can be seen that the selection time is generally very small compared to service location latency, since both QoS parsing and computing do not incur any message transmission or computing expensive operations.

Number of dimensions	3	6	9	12	15
5 instances	0.16	0.34	0.6	0.93	1.33
10 instances	0.19	0.44	0.78	1.24	1.81
15 instances	0.22	0.53	0.97	2.01	2.28

Table VII.7: Service selection time in QoWSD

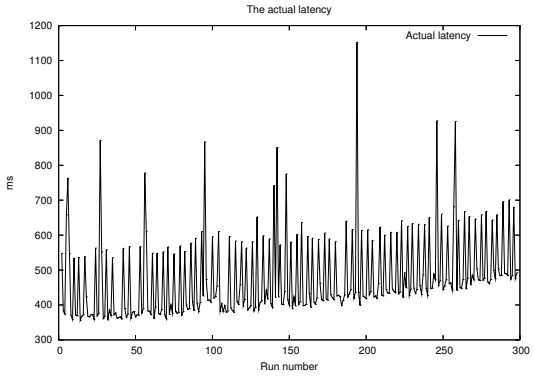


Figure VII.18: Actual service latency for different runs

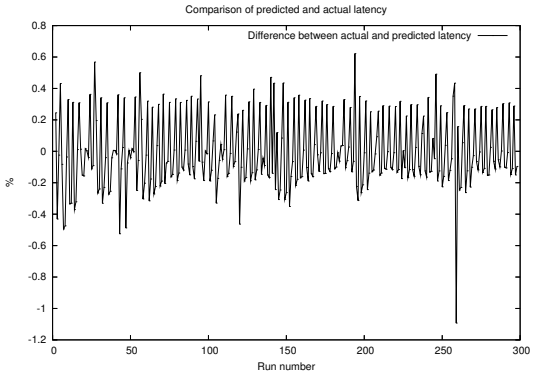


Figure VII.19: Service latency prediction errors

QoS Prediction. After each service invocation, the QoS values are recorded by the the service invoker in order to predict new values for QoS description. We implement a `FormatConvert` service, which converts a zipped postscript file to a zipped PDF file. Figure VII.18 shows the actual service latency for 300 runs for converting a file with size of about 50 KB. Figure VII.19 shows the prediction error, which is defined as (actual latency - predicted latency)/actual latency. It can be observed that with latency fluctuating in a large range, it is possible to estimate the service latency with certain degree of accuracy (less than 20%). It presents an example of estimating QoS values by the QoS predictor.

Summary. In the above, a prototype implementing *QoWSD* middleware is evaluated with respect to the overhead introduced by QoS-awareness on the service discovery latency. The results suggest that the introduction of QoS description and beacons brings a reasonable overhead (increase by less than 15%); the QoS-aware service selection incurs very small overhead; the recommendations are better elicited from nearby nodes (e.g., less than 3 hops) to avoid large increase of service discovery latency.

VII.4 Concluding Remarks

We have presented in this chapter a QoS-aware Web Service Discovery (*QoWSD*) middleware targeting ubiquitous computing environments, integrating the solutions of the previous three chapters, including signal strength based service location, QoS-aware service selection using Vickrey auction and a robust and incentive compatible reputation mechanism. It allows service providers to register their services and clients to discover and invoke services. Using *QoWSD*, a client is able to find a service that (1) has more robustness against device mobility; (2) is best in matching its needs among the service instances that satisfy its requirements; (3) is hosted by honest service providers. A prototype implementing *QoWSD* has been deployed on wireless multi-hop ad hoc networks. We have measured the impact of QoS awareness in terms of increased service discovery latency due to QoS description and beacon, service selection and recommendation elicitation. In general, the overhead appears reasonable for enhancing service discovery with QoS awareness in the ubiquitous computing environments. In summary, *QoWSD* demonstrates as an effective overall solution to QoS-aware service discovery in ubiquitous computing environments. It thus validates our solutions addressing three different aspects of service discovery as presented previously.

VIII

Conclusion

The vision of ubiquitous computing [Weiser, 1991] is becoming a reality thanks to the advent of portable devices (e.g., smartphones) and the advances in wireless networking technologies (e.g., WLAN, Bluetooth). It aims to facilitate user tasks through seamless utilization of various services in the surrounding environments. To realize the above goal, the handheld wireless device (e.g., a cellphone) carried by a user needs to dynamically discover services, whether embedded in the environment or hosted by other handheld devices belonging to other users. The services can be very much diverse in their QoS properties due to factors such as service provider's computing power. Besides, services also bear prices, which are charged by service providers on clients, in order to motivate service provisioning among autonomous entities. Moreover, to allow for flexible interaction, devices are preferably interconnected using MANETs instead of through network infrastructure. This is because the former is deployment free and realizes spontaneous networking, which supports impromptu interactions between entities, considered as a desirable feature for ubiquitous computing [Kindberg and Fox, 2002]. The characteristics of ubiquitous computing as described above necessitates the incorporation of QoS awareness during service discovery.

VIII.1 Contribution

This thesis focuses on supporting QoS-awareness during service discovery in ubiquitous computing environments. Our contribution lies in identifying and addressing the following three aspects related to service's QoS properties, which are not well handled by the current service discovery protocols.

Firstly, device mobility, which is inherent in ubiquitous computing, can disconnect wireless links and cause service failures, thus degrading service reliability. Routing protocols for ad hoc networks do not provide enough mobility support because their provided routes can tend to break. In light of this, we propose a signal strength based service location (S3L) that identifies the links that are likely to break due to mobility. By following service paths without these expiring links, S3L improves service reliability. Based on simulation-based evaluation, S3L improves considerably service reliability in most scenarios (i.e., when services last longer than 1 second and devices move at speeds less than 11m/s).

Secondly, the abundance of services in the environments makes it likely to locate multiple service instances satisfying the client's requirements. Service selection, which chooses the best one on behalf of the client, needs to comprehensively take into account service's non-functional properties, including service price and QoS properties, as well as the client's preferences among them. We present a utility function for doing so, which is complemented by a Vickrey auction based pricing model to motivate service providers' truthful revelation of service prices. Such a pricing model ensures that a client's service selection is based on the truthful service cost information from providers.

Thirdly, in order to avoid dishonest service providers, we also devise a distributed reputation mechanism that evaluates an entity's trustworthiness based on direct experiences and others' (honest) recommendations. Due to the featured openness of the ubiquitous computing environments, it is very commonplace for an entity to interact with others that it has little knowledge of, increasing the importance of recommendations. However, recommendations can be deviated from the truth (e.g., due to collusion) or withheld due to lack of incentives. Therefore, in our proposed reputation mechanism, recommenders are evaluated depending on their honesty and activeness in providing recommendations. Only recommendations from honest recommenders are considered helpful and taken into account. Moreover, by treating recommenders differently depending on their honesty and activeness, it is ensured that an entity benefits from others' (helpful) recommendations in proportion to its contribution. More specifically, an honest and active recommender can elicit honest (i.e., helpful) recommendations more easily than others, for the purpose of stimulating truthful recommendations.

Finally, the above solutions addressing different aspects are incorporated into a middleware that supports QoS-aware Web service discovery in ubiquitous computing environments. The middleware enables a client to discover Web services that (1) have improved reliability, (2) are best in matching the client's preferences among non-functional properties and (3) are trustworthy

(i.e., hosted by honest service providers). A prototype implementing the middleware is further deployed on a multihop ad hoc network and evaluated, especially, regarding the overhead of introducing QoS-awareness on service discovery latency. The overhead seems reasonable according to the results of performance evaluation.

VIII.2 Perspective

Besides the contribution as stated above, this thesis can be further extended to incorporate other related functionalities. For example, the selective multicast in S3L can also be based on service functionality (e.g., GSD [Chakraborty et al., 2006]), besides link stability. Specifically, a node announces not only its services, but also its "seen" services (e.g., services advertised by others). Although the latter is generally described with less details, it makes it possible to identify those one-hop neighbors that have more probability of knowing where the requested service is located. Thus, service requests can be sent along links that not only are considered stable by S3L, but also lead to higher possibility of finding the requested services. Moreover, service discovery in ubiquitous computing environments can be extended with group support that organizes related nodes as groups (e.g., participants of a conference). Such support can facilitate QoS awareness during service discovery. For example, if group members move with similar speeds and directions, service location can derive the mobility of an individual from that of its group. Signal strength tendency of beacons (e.g., from border nodes of a group which are connected to nodes from other groups) can then be used to detect the moving tendency of a group. Moreover, since nodes can join and leave groups freely, it necessitates detection of group member departure, which can also be based on beacons' SS tendency. Similarly, group members can share their reputation. This can ease trustworthiness recognition since it is more likely to have direct experiences with group members than with an individual group member. Meanwhile, group reputation requires strong degree of mutual trust between group members, which can also be dynamic and evolving and be enforced by group membership management.

Bibliography

[Abdul-Rahman and Hailes, 2000] Abdul-Rahman, A. and Hailes, S. (2000). Supporting trust in virtual communities. In *Proc. Hawaii Int'l Conf. System Science HICSS-33*. *Cited in page(s):* 38 , 86

[Aberer and Despotovic, 2001] Aberer, K. and Despotovic, Z. (2001). Managing trust in a peer-2-peer information system. In Paques, H., Liu, L., and Grossman, D., editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM01)*, pages 310–317. ACM Press. *Cited in page(s):* 37

[ACPI, 2004] ACPI (2004). Advanced configuration and power interface specification. `http://www.acpi.info/`. revision 3.0. *Cited in page(s):* 72

[Adams and Farrell, 1999] Adams, C. and Farrell, S. (1999). *RFC2510 - Internet X.509 Public Key Infrastructure Certificate Management Protocols*. *Cited in page(s):* 16

[Adjie-Winoto et al., 1999] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., and Lilley, J. (1999). The design and implementation of an intentional naming system. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 186–201, New York, NY, USA. ACM Press. *Cited in page(s):* 21 , 31

[Agarwal et al., 2000] Agarwal, S., Ahuja, A., Singh, J. P., and Shorey, R. (2000). Route-lifetime assessment based routing (RABR) protocol for mobile ad-hoc networks. In *Proc. of IEEE International Conference on Communications (ICC)*. *Cited in page(s):* 26 , 27

[Al-Ali et al., 2003] Al-Ali, R. J., ShaikhAli, A., Rana, O. F., and Walker, D. W. (2003). Supporting QoS-Based discovery in Service-Oriented Grids. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. *Cited in page(s):* 23

[Aurrecoechea et al., 1998] Aurrecoechea, C., Campell, A. T., and Hauw, L. (1998). A survey of QoS architectures. *ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture*, 3(6). *Cited in page(s):* 70

[Avizienis et al., 2001] Avizienis, A., Laprie, J. C., and Randell, B. (2001). Fundamental concepts of computer system dependability. In *Proc. of IARP/IEEE Workshop on Robot Dependability: Technological Challenges of Dependable Robots in Human Environments*. *Cited in page(s):* 66 , 67 , 68

[Bahl and Padmanabhan, 2000] Bahl, P. and Padmanabhan, V. N. (2000). RADAR: An in-building rf-based user location and tracking system. In *Proc. of IEEE INFOCOM*. *Cited in page(s):* 26

[Baker and Dobson, 2005] Baker, S. and Dobson, S. (2005). Comparing service-oriented and distributed object architectures. In *Proceedings of the International Symposium on Distributed Objects and Applications*. *Cited in page(s):* 12

[Baudron and Stern, 2001] Baudron, O. and Stern, J. (2001). Non-interactive private auctions. In *Proc. of Fifth Int'l Conf. on Financial Cryptography(FC)*. *Cited in page(s):* 82

[Beckman et al., 2002] Beckman, W., Crowcroft, J., Gevros, P., and Oleneva, M. (2002). TAPAS Deliverable D1. `http://tapas.sourceforge.net/deliverables/index.html`. *Cited in page(s):* 66

[Bhagwat et al., 1994] Bhagwat, P., Perkins, C. E., and Tripathi, S. K. (1994). Transparent resources discovery for mobile computers. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US. *Cited in page(s):* 14

[Bluetooth SDP, 2004] Bluetooth SDP (2004). Service discovery protocol, bluetooth specification 2.0 part b. `https://www.bluetooth.org/spec/`. *Cited in page(s):* 21

[Boulkenafed and Issarny, 2003] Boulkenafed, M. and Issarny, V. (2003). A middleware service for mobile ad hoc data sharing, enhancing data availability. In *Proceedings of the 4th ACM/IFIP/USENIX International Middleware Conference*. *Cited in page(s):* 10

[Bowers et al., 2000] Bowers, S., Delcambre, L., Maier, D., Cowan, C., Wagle, P., McNamee, D., Meur, A. F. L., and Hinton, H. (2000). Applying adaptation spaces to support quality of service and survivability. In *DARPA Information Survivability Conference and Exposition*. *Cited in page(s):* 70

[Brandt and Weiβ, 2001] Brandt, F. and Weiβ, G. (2001). Vicious strategies for Vickrey auctions. In *Proc. of the Fifth Int'l Conf. on Autonomous Agents*. ACM Press. *Cited in page(s):* 82

[Buchegger and Boudec, 2002] Buchegger, S. and Boudec, J. Y. L. (2002). Performance analysis of the CONFIDANT protocol. In *Proc. of MobiHOC*. *Cited in page(s):* 18 , 38 , 39 , 103

[Buchegger and Boudec, 2003] Buchegger, S. and Boudec, J.-Y. L. (2003). The effect of rumor spreading in reputation systems for mobile ad-hoc networks. In *Proc. workshop on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt03)*. *Cited in page(s):* 39 , 83

[Buchegger and Boudec, 2004] Buchegger, S. and Boudec, J.-Y. L. (2004). A robust reputation system for P2P and mobile ad-hoc networks. In *Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems*. *Cited in page(s):* 39 , 85

[Buchegger and Boudec, 2005] Buchegger, S. and Boudec, J.-Y. L. (2005). Self-policing mobile ad hoc networks by reputation systems. *IEEE Communications Magazine*, 43(7):101– 107. *Cited in page(s):* 39

[Buttyan and Hubaux, 2003] Buttyan, L. and Hubaux, J. P. (2003). Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM/Kluwer Mobile Networks and Applications*, 8(5). *Cited in page(s):* 9

[Caffery and Stuber, 1998] Caffery, J. J. and Stuber, G. L. (1998). Overview of radiolocation in CDMA cellular systems. *IEEE Communications Magazine*. *Cited in page(s):* 26

[Cahill et al., 2003] Cahill, V., Gray, E., Seigneur, J.-M., et al. (2003). Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 2(3). *Cited in page(s):* 9 , 16

[Camp et al., 2002] Camp, T., Boleng, J., and Davies, V. (2002). A survey of mobility models for ad hoc network research. *Wireless Communication and Mobile Computing (WCMC)*, 2(5). *Cited in page(s):* 58

[Campbell et al., 2002] Campbell, R. H., Al-Muhtadi, J., Naldurg, P., Sampemane, G., and Mickunas, M. D. (2002). Towards security and privacy for pervasive computing. In *Proc. of international Symposium of Software Security – Theories and Systems, Mext-NSF-JSPS*, pages 1–15. *Cited in page(s):* 11

[Capra et al., 2003] Capra, L., Emmerich, W., and Mascolo, C. (2003). CARISMA: Context-Aware REflective mIddleware System for Mobile Applications. *IEEE Transactions of Software Engineering*, 29(10). *Cited in page(s):* 35

[Capra et al., 2005] Capra, L., Zachariadis, S., and Mascolo, C. (2005). Q-CAD: QoS and Context Aware Discovery Framework for Mobile Systems. In *Proc. of International Conference on Pervasive Services (ICPS'05)*. *Cited in page(s):* 12

[Cardoso et al., 2004] Cardoso, J., Sheth, A., Millerb, J., Arnoldc, J., and Kochutb, K. (2004). Quality of Service for workflows and Web service processes. *journal of Web Semantics*, 1(3). *Cited in page(s):* 23 , 66 , 68

[Casella and Berger, 2002] Casella, G. and Berger, R. L. (2002). *Statistical Inference*. Duxbury Press. *Cited in page(s):* 84

[Chakraborty et al., 2006] Chakraborty, D., Joshi, A., Yesha, Y., and Finin, T. (2006). Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112. *Cited in page(s):* 11 , 21 , 22 , 23 , 131

[Chakraborty et al., 2001] Chakraborty, D., Perich, F., Avancha, S., and Joshi, A. (2001). Dreggie: Semantic service discovery for m-commerce applications. In *Proc. of Workshop on Reliable and Secure Applications in Mobile Environment*. *Cited in page(s):* 23

[Chakravorty et al., 2005] Chakravorty, R., Agarwal, S., Banerjee, S., and Pratt, I. (2005). Mob: a mobile bazaar for wide-area wireless services. In *Proceedings of the 11th annual international conference on Mobile computing and networking (MobiCom)*, pages 228–242. *Cited in page(s):* 37

[Chalmers and Sloman, 1999] Chalmers, D. and Sloman, M. (1999). A survey of quality of service in mobile computing environments. *IEEE communications surveys*. *Cited in page(s):* 66 , 68

[Chen et al., 2005] Chen, L.-J., Sun, T., Yang, G., Sanadidi, M. Y., and Gerla, M. (2005). Adhoc probe: Path capacity probing in wireless ad hoc networks. In *Proc. of the First International Conference on Wireless Internet (WICON)*. *Cited in page(s):* 72

[Chen et al., 2003] Chen, Z., Liang-Tien, C., Silverajan, B., and Bu-Sung, L. (2003). UX - An Architecture Providing QoS-Aware and Federated Support for UDDI. In *proceeding of the first International Conference on Web Services(ICWS)*. *Cited in page(s):* 23

[Chin et al., 2002] Chin, K. W., Judge, J., Williams, A., and Kermode, R. (2002). Implementation experience with manet routing protocols. *ACM SIGCOMM Comp. Comm. Review*, 32(5). *Cited in page(s):* 26

[Clausen and Jacquet, 2003] Clausen, T. and Jacquet, P. (2003). Optimized link state routing protocol. IETF RFC 3626. *Cited in page(s):* 48 , 59

[Conti et al., 2004] Conti, M., Maselli, G., Turi, G., and Giordano, S. (2004). Cross layering in mobile ad hoc network design. *IEEE Computer*, pages 48–51. *Cited in page(s):* 48

[Couto et al., 2002] Couto, D. D., Aguayo, D., Cambers, B. A., and Morris, R. (2002). Performance of multihop wireless networks: Shortest path is not enough. In *Proc. of 1st workshop on Hot Topics in Networking*. *Cited in page(s):* 31

[Creese et al., 2004] Creese, S., Goldsmith, M., Roscoe, B., and Zakiuddin, I. (2004). Research directions for trust and security in human-centric computing. In *Proc. of the First Workshop on Security and Privacy at the Conference on Pervasive Computing*. *Cited in page(s):* 11

[Czerwinski et al., 1999] Czerwinski, S. E., Zhao, B. Y., Hodes, T. D., Joseph, A. D., and Katz, R. H. (1999). An architecture for a secure service discovery service. In *Proc. of ACM MobiCom*. *Cited in page(s):* 1

[Dawkins, 1989] Dawkins, R. (1989). *The Selfish Gene*. Oxford University Press. *Cited in page(s):* 95

[Day and Deters, 2004] Day, J. and Deters, R. (2004). Selecting the best web service. In *Proc. of Conf. of the center for advanced studies on collaborative research*. *Cited in page(s):* 32

[Dellarocas, 2003] Dellarocas, C. (2003). The digitization of word-of-mouth: promise and challenges of online feedback mechanisms. MIT Working Paper. *Cited in page(s):* 89

[Dey, 2001] Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7. *Cited in page(s):* 11

[Dokovski et al., 2004] Dokovski, N., Widya, I., and van Halteren, A. (2004). Paradigm: Service oriented computing. AWARENESS whitepaper, `https://doc.freeband.nl/dscgi/ds.py/Get/File-49216/D2.7b_-_Paradigm_-_Service_Oriented_Computing.pdf`. *Cited in page(s):* 12

[Douceur, 2002] Douceur, J. (2002). The sybil attack. In *Proc. of the IPTPS02 Workshop*. *Cited in page(s):* 103

[Dube et al., 1997] Dube, R., Rais, C. D., Wang, K. Y., and Tripathi, S. K. (1997). Signal stability based adaptive routing (SSA) for ad-hoc mobile networks. *IEEE Personal Communications*, 4(2). *Cited in page(s):* 26

[Dykes et al., 2000] Dykes, S. G., Robbins, K. A., and Jeffrey, C. L. (2000). An empirical evaluation of client-side server selection algorithms. In *Proc. of IEEE Infocom*. *Cited in page(s):* 31 , 73

[Ekenstam et al., 2001] Ekenstam, T., Matheny, C., Reiher, P., and Popek, G. (2001). The bengal database replication system. *Distributed and Parallel Databases*, 9(3). *Cited in page(s):* 10

[Feeney and Nilsson, 2001] Feeney, L. M. and Nilsson, M. (2001). Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proc. of IEEE INFOCOM*. *Cited in page(s):* 72

[Fei et al., 1998] Fei, Z., Bhattacharjee, S., Zegura, E. W., and Ammar, M. H. (1998). A novel server selection technique for improving the response time of a replicated service. In *Proc. of IEEE Infocom*. *Cited in page(s):* 31

[Feigenbaum and Shenker, 2002] Feigenbaum, J. and Shenker, S. (2002). Distributed algorithmic mechanism design: Recent results and future directions. In *Proc. 6th Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*. *Cited in page(s):* 34

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). HTTP/1.1. IETF, RFC 2616. *Cited in page(s):* 31

[Forman and Zahorjan, 1994] Forman, G. H. and Zahorjan, J. (1994). The challenges of mobile computing. *IEEE Computer*, 27(4). *Cited in page(s):* 7 , 8

[Friendman and Resnick, 2001] Friendman, E. and Resnick, P. (2001). The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2). *Cited in page(s):* 103

[Frolund and Koistinen, 1998] Frolund, S. and Koistinen, J. (1998). QML: A language for quality of service specification. Technical Report HPL-98–10, Hewlette Packard. *Cited in page(s):* 66

[Gambetta, 1990] Gambetta, D. (1990). Can we trust trust? In *Trust, Making and Breaking Cooperative Relations*, chapter 13, pages 213–237. basil blackwell. *Cited in page(s):* 17

[Gao and Steenkiste, 2002] Gao, J. and Steenkiste, P. (2002). Rendezvous points-based scalable content discovery with load balancing. In *Proc. of the Fourth International Workshop on Networked Group Communication*. *Cited in page(s):* 31

[Geihs, 2002] Geihs, K. (2002). Analysis of adaptation strategies for mobile QoS-Aware applications. In *Proc. of ACM MSWiM*. *Cited in page(s):* 29

[Goff et al., 2001] Goff, T., Abu-Ghuzaleh, N. B., Phatak, D. S., and Kahvecioglu, R. (2001). Preemptive routing in ad hoc networks. In *Proc. ACM MobiCom*. *Cited in page(s):* 26 , 27

[Goland et al., 1999] Goland, Y., Cai, T., Leach, P., Gu, Y., and Albright, S. (1999). Simple service discovery protocol. IETF Draft, `http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt`. *Cited in page(s):* 21 , 28

[Grandison and Sloman, 2000] Grandison, T. and Sloman, M. (2000). A survey of trust in Internet applications. *IEEE Communication Surveys*, 3(4). *Cited in page(s):* 17

[Gray and Reuter, 1993] Gray, J. and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers. *Cited in page(s):* 67

[Gurun et al., 2004] Gurun, S., Krintz, C., and Wolski, R. (2004). NWSLite: a light-weight prediction utility for mobile devices. In *Proc. of 4th ACM MobiSys*. *Cited in page(s):* 72 , 119

[Guttman et al., 1999] Guttman, E., Perkins, C., Veizades, J., and day, M. (1999). Service location protocol, version 2. RFC 2608. *Cited in page(s):* 21 , 22

[He et al., 2004] He, Q., Wu, O. D., and Khosla, P. (2004). SORI: A secure and objective reputation-based incentive scheme for ad-hoc networks. In *Proc. of IEEE Wireless Communications and Networking Conference*. *Cited in page(s):* 38

[Helal, 2002] Helal, S. (2002). Standards for service discovery and delivery. *IEEE Pervasive computing*, 1(3). *Cited in page(s):* 21

[Helal et al., 2003] Helal, S., Desai, N., Verma, V., and Lee, C. (2003). Konark - a service discovery and delivery protocol for ad hoc networks. In *Proc. of 3rd IEEE Conf. on WCNC*. *Cited in page(s):* 22 , 28

[Hightower et al., 2002] Hightower, J., Brumitt, B., and Borriello, G. (2002). The location stack: A layered model for location in ubiquitous computing. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*. *Cited in page(s):* 11

[Hille et al., 2000] Hille, S., Jonkers, H., Tokmakoff, A., and Wibbels, M. (2000). State-of-the-art in electronic accounting, billing and payment. Telematica Institut GigaABP D1.1. *Cited in page(s):* 33

[Hodes et al., 2002] Hodes, T. D., Czerwinski, S. E., Zhao, B. Y., Joseph, A. D., and Katz, R. H. (2002). An architecture for secure wide-area service discovery. *Wireless Networks*, 8(2-3):213–230. *Cited in page(s):* 12 , 21 , 28

[Huhns and Singh, 2005] Huhns, M. N. and Singh, M. P. (2005). Service-oriented computing: key concepts and principles. *IEEE Internet Computing*, 9(1):75–81. *Cited in page(s):* 12

[Hung and Li, 2003] Hung, P. and Li, H. (2003). Web services discovery based on the trade-off between quality and cost of service : a token-based approach. *ACM SIGecom Exchanges*, 4(2):21–31. *Cited in page(s):* 15

[Huynh et al., 2005] Huynh, T. D., Jennings, N. R., and Shadbolt, N. (2005). On handling inaccurate witness reports. In *Proc. 8th International Workshop on Trust in Agent Societies*, pages 63–77, Utrecht, The Netherlands. *Cited in page(s):* 39 , 83

[IEEE, 1999] IEEE (1999). Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification. IEEE Standard 802.11. *Cited in page(s):* 25

[ISO, 2002] ISO (2002). Quality management systems – fundamentals and vocabulary(iso 9000:2000). *Cited in page(s):* 13

[Issarny et al., 2005] Issarny, V., Sacchetti, D., Tartanoglu, F., Sailhan, F., Chibout, R., Levy, N., and Talamona, A. (2005). Developing Ambient Intelligence Systems: A Solution based on Web Services. *Journal of Automated Software Engineering*. *Cited in page(s):* 105

[Issarny et al., 2004] Issarny, V., Tartanoglu, F., Liu, J., and Sailhan, F. (2004). Software architecture for mobile distributed computing. In *Proceedings of*

4th Working IEEE/IFIP Conference on Software Architecture (WICSA 2004), pages 201–210. *Cited in page(s):* 10

[Jiang et al., 2001] Jiang, S., He, D., and Rao, J. (2001). A prediction-based link availability estimation for mobile ad hoc networks. In *Proc. of IEEE INFOCOM*. *Cited in page(s):* 26

[Johnson et al., 2004] Johnson, D. B., Maltz, D. A., and Hu, Y.-C. (2004). The dynamic source routing protocol for mobile ad hoc networks (DSR). IETF Draft, `http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt`. *Cited in page(s):* 54

[Josang et al., 2005] Josang, A., Ismail, R., and Boyd, C. (2005). A survey of trust and reputation systems for online service provision. *Decision Support Systems (to appear)*. *Cited in page(s):* 37

[Jurca and Faltings, 2003] Jurca, R. and Faltings, B. (2003). An incentive compatible reputation mechanism. In *Proceedings of IEEE International Conference on E-Commerce*, CA, USA. *Cited in page(s):* 18 , 40

[Jøsang and Ismail, 2002] Jøsang, A. and Ismail, R. (2002). The beta reputation system. In *Proc. of 15th Bled Conf. on Electronic Commerce*. *Cited in page(s):* 85 , 89

[Kagal et al., 2002] Kagal, L., Korolev, V., Avancha, S., Joshi, A., Finin, T., and Yesha, Y. (2002). Centaurus: an infrastructure for service management in ubiquitous computing environments. *Wireless Networks*, 8(6):619–635. *Cited in page(s):* 10

[Kamvar et al., 2003] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. (2003). The EigenTrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International World Wide Web Conference*. *Cited in page(s):* 37 , 38

[Karlins and Abelson, 1970] Karlins, M. and Abelson, H. I. (1970). *Persuasion, how opinion and attitudes are changed*. Crosby Lockwood & Son. *Cited in page(s):* 89

[Kindberg and Fox, 2002] Kindberg, T. and Fox, A. (2002). System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70–81. *Cited in page(s):* 7 , 16 , 129

[Klemm et al., 2005] Klemm, F., Krishnamurthy, S. V., and Tripathi, S. K. (2005). Improving TCP performance in ad hoc networks using signal

strength based link management. *Ad Hoc Networks*, 3(2). *Cited in page(s):* 26 , 27

[Koistinen, 1997] Koistinen, J. (1997). Dimensions for reliability contracts in distributed objects systems. Technical Report HPL-97-119, Hewlette Packard. *Cited in page(s):* 67

[Kollock, 1994] Kollock, P. (1994). The emergence of exchange structures: an experimented study of uncertainty, commitment, and trust. *American Journal of sociology*, 100(2). *Cited in page(s):* 88

[Kommerling and Kuhn, 1999] Kommerling, O. and Kuhn, M. G. (1999). Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology*. *Cited in page(s):* 9

[Kotz et al., 2003] Kotz, D., Newport, C., and Elliott, C. (2003). The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth College, Computer Science, Hanover, NH. *Cited in page(s):* 28

[LBNL, 2001] LBNL (2001). Network simulator ns-2. `http://www.isi.edu/nsnam/ns`. *Cited in page(s):* 57 , 96

[Lee and Helal, 2003] Lee, C. and Helal, S. (2003). A multi-tier ubiquitous service discovery protocol for mobile clients. In *Proc. of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECT)*. *Cited in page(s):* 31

[Lee et al., 2004] Lee, G., Faratin, P., Bauer, S., and Wroclawski, J. (2004). A user-guided cognitive agent for network service selection in pervasive computing environments. In *Proc. of 2nd IEEE Annual Conference on Pervasive Computing and Communications*. *Cited in page(s):* 30 , 73

[Liao and Campbell, 2001] Liao, R. R.-F. and Campbell, A. T. (2001). A utility-based approach for quantitative adaptation in wireless packet networks. *Wireless Networks*, 7(5):541–557. *Cited in page(s):* 29

[Liu and Issarny, 2004a] Liu, J. and Issarny, V. (2004a). Enhanced reputation mechanism for mobile ad hoc networks. In *Proc. of Int'l Conf. on Trust Management (iTrust 2004)*. LNCS 2995. *Cited in page(s):* 89

[Liu and Issarny, 2004b] Liu, J. and Issarny, V. (2004b). QoS-aware service location in mobile ad hoc networks. In *Proc. of the 5th IEEE Int'l Conf. on Mobile Data Management (MDM)*. *Cited in page(s):* 74

[Liu and Issarny, 2004c] Liu, J. and Issarny, V. (2004c). Service allocation in selfish mobile ad hoc networks using Vickrey auction. In *Current Trends in Database Technology - EDBT 2004 Workshops, EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14-18, 2004, Revised Selected Papers*, pages 385–394. *Cited in page(s):* 82

[Liu and Issarny, 2005] Liu, J. and Issarny, V. (2005). Signal Strength based Service Discovery (S3D) in Mobile Ad Hoc Networks. In *Proc. of the 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*. *Cited in page(s):* 11

[Liu and Issarny, 2006] Liu, J. and Issarny, V. (2006). An incentive compatible reputation mechanism for ubiquitous computing environments. In *Proc. of 4th Int'l Conf. on Privacy, Security and Trust (to appear)*. *Cited in page(s):* 83

[Liu et al., 2005] Liu, J., Sailhan, F., Sacchetti, D., and Issarny, V. (2005). Group management for mobile ad hoc networks: Design, implementation and experiment. In *Proc. of the 6th International Conference on Mobile Data Management (MDM05)*. *Cited in page(s):* 102

[Liu et al., 2003] Liu, J., Sohraby, K., Zhang, Q., Li, B., and Zhu, W. (2003). Resource discovery in mobile ad hoc networks. *The handbook of ad hoc wireless networks*. *Cited in page(s):* 24

[Liu et al., 2004] Liu, Y., Ngu, A. H. H., and Zeng, L. (2004). QoS computation and policing in dynamic Web service selection. In *Proc. of WWW conference*. *Cited in page(s):* 30 , 75

[Lyytinen and Yoo, 2002] Lyytinen, K. and Yoo, Y. (2002). Issues and challenges in ubiquitous computing, introduction. *Communication of ACM*, 45(12):62–65. *Cited in page(s):* 5

[MacKie-Mason and Varian, 1994] MacKie-Mason, J. K. and Varian, H. R. (1994). Pricing the internet. Technical report, Economics Working Paper. *Cited in page(s):* 33

[MANET, 2005] MANET (2005). IETF working group: Mobile ad hoc networks (manet). `http://www.ietf.org/html.charters/manet-charter.html`. *Cited in page(s):* 6

[Marsh, 1994] Marsh, S. P. (1994). *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling. *Cited in page(s):* 86 , 87

[Marti et al., 2000] Marti, S., Giuli, T. J., Lai, K., and Baker, M. (2000). Mitigating routing misbehavior in mobile ad hoc networks. In *Proc. of the 6th ACM International Conf. on Mobile Computing and Networking*. *Cited in page(s):* 9 , 38

[Mascolo et al., 2001] Mascolo, C., Capra, L., Zachariadis, S., and Emmerich, W. (2001). XMIDDLE: A data-sharing middleware for mobile computing. *Wireless Personal Communications*, 21(1). *Cited in page(s):* 10

[Maximilien and Singh, 2004] Maximilien, E. M. and Singh, M. P. (2004). A framework and ontology for dynamic Web services selection. *IEEE Internet Computing*. *Cited in page(s):* 23

[McAfee and McMillan, 1987] McAfee, R. P. and McMillan, J. (1987). Auctions and bidding. *Journal of Economic Literature*, 25. *Cited in page(s):* 35

[McMillan, 1994] McMillan, M. (1994). Selling spectrum rights. *Journal of Economic Perspectives*. *Cited in page(s):* 35

[Michiardi and Molva, 2002] Michiardi, P. and Molva, R. (2002). CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *CMS'2002*. *Cited in page(s):* 18 , 37 , 38

[Miller and Pascoe, 2000] Miller, B. A. and Pascoe, R. A. (2000). Salutation service discovery in pervasive computing environments. IBM White Paper. *Cited in page(s):* 5 , 13

[Miller et al., 2002] Miller, N., Resnick, P., and Zeckhauser, R. (2002). Eliciting honest feedback in electronic markets. Working Paper. *Cited in page(s):* 17 , 18

[Mokhtar et al., 2005] Mokhtar, S. B., Liu, J., Georgantas, N., and Issarny, V. (2005). QoS-aware dynamic service composition in ambient intelligence environments. In *Proc. of ACM/IEEE International Conference on Automatic Software Engineering*. *Cited in page(s):* 18

[Mui et al., 2001] Mui, L., Mohtashemi, M., Ang, C., Szolovits, P., and Halberstadt, A. (2001). Ratings in distributed systems: A bayesian approach. In *Proc. of 11th Workshop on Information Technologies and Systems (WITS)*. *Cited in page(s):* 85

[Mui et al., 2002] Mui, L., Mohtashemi, M., and Halberstadt, A. (2002). A computational model of trust and reputation. In *Proceedings of the 35th HICSS*. *Cited in page(s):* 17 , 87 , 94 , 102

[Naor et al., 1999] Naor, M., Pinkas, B., and Sumner, R. (1999). Privacy preserving auctions and mechanism design. In *Proc. of ACM Conference on Electronic Commerce*. *Cited in page(s):* 82

[Narayanan et al., 2000] Narayanan, D., Flinn, J., and Satyanarayanan, M. (2000). Using history to improve mobile application adaptation. In *Proc. of 3rd IEEE Workshop on Mobile Computing Systems and Applications*. *Cited in page(s):* 70

[Nash et al., 2005] Nash, D. C., Martin, T. L., Ha, D. S., and Hsiao, M. S. (2005). Towards an intrusion detection for battery exhaustion attacks on mobile computing devices. In *Proc. of 3rd Int'l Conf. on Pervasive Computing and Communications Workshops*. *Cited in page(s):* 72

[Niculeascu and Nath, 2001] Niculeascu, D. and Nath, B. (2001). Ad hoc positioning system (APS). In *Proc. of IEEE GlobeCOM*. *Cited in page(s):* 26

[Niculeascu and Nath, 2003] Niculeascu, D. and Nath, B. (2003). Ad hoc positioning system (APS) using AOA. In *Proc. of IEEE INFOCOM*. *Cited in page(s):* 26

[Nidd, 2001] Nidd, M. (2001). Service discovery in DEAPspace. *IEEE Personal Communications*, 8(4). *Cited in page(s):* 21 , 28

[Nisan and Ronen, 2001] Nisan, N. and Ronen, A. (2001). Algorithmic mechanism design. In *31st ACM Symp. on Theory of Computing*, pages 129–140. *Cited in page(s):* 34 , 35

[Noble and Satyanarayanan, 1999] Noble, B. D. and Satyanarayanan, M. (1999). Experience with adaptive mobile applications in odyssey. *Mob. Netw. Appl.*, 4(4):245–254. *Cited in page(s):* 10 , 29

[Noble et al., 1997] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. (1997). Agile application-aware adaptation for mobility. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 276–287, New York, NY, USA. ACM Press. *Cited in page(s):* 70

[OASIS, 2005] OASIS (2005). Service oriented architecture reference model. Working Draft 05, `http://xml.coverpages.org/SOA-RM-WD05.pdf`. *Cited in page(s):* 12

[Obreiter et al., 2003] Obreiter, P., König-Ries, B., and Klein, M. (2003). Stimulating cooperative behavior of autonomous devices: an analysis of requirements and existing approaches. In *Proc. of the 2nd Int'l Workshop on Wireless Information Systems*. *Cited in page(s):* 9

[Odlyzko, 1999] Odlyzko, A. (1999). Paris metro pricing for the internet. In *Proc. of ACM conference on Electronic Commerce*. *Cited in page(s):* 33

[Papazoglou, 2003] Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In *Proc. of Fourth International Conference on Web Information Systems Engineering (WISE'03)*. *Cited in page(s):* 12

[Papazoglou and Georgakopoulos, 2003] Papazoglou, M. P. and Georgakopoulos, D. (2003). Service-oriented computing. *Communications of the ACM*, 46(10). *Cited in page(s):* 1 , 12

[Paradiso and Starner, 2005] Paradiso, J. A. and Starner, T. (2005). Energy scavenging for mobile and wireless electronics. *IEEE Pervasivce Computing*. *Cited in page(s):* 8

[Patel et al., 2005] Patel, J., Teacy, W. L., Jennings, N. R., and Luck, M. (2005). A probabilistic trust model for handling inaccurate reputation sources. In *Proc. of third international conference on Trust Management (iTrust 2005) LNCS Volume 3477*, pages 193 – 209. *Cited in page(s):* 40

[Pedersen, 1991] Pedersen, T. (1991). Non-interactive and information-theoretic secure verifiable secret sharing. *Advances in Cryptology - CRYPTO 1991*, 576:129–140. *Cited in page(s):* 82

[Perkins, 2001] Perkins, C. E. (2001). *Ad hoc networking*. Addison Wesley. *Cited in page(s):* 6

[Perkins et al., 1998] Perkins, C. E., Alpert, S. R., and Woolf, B. (1998). *Mobile IP: Design Principles and Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. *Cited in page(s):* 14

[Pfleeger, 1997] Pfleeger, C. (1997). *Security in Computing*. Prentice Hall PTR. *Cited in page(s):* 67

[Punnoose et al., 2000] Punnoose, R. J., Nikitin, P. V., and Stancil, D. D. (2000). Efficient simulation of Ricean fading with a packet simulator. In *Proc. of IEEE VTC*. *Cited in page(s):* 57

[Qin and Kunz, 2002] Qin, L. and Kunz, T. (2002). Proactive routing maintenance in DSR. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(3). *Cited in page(s):* 26 , 27

[Raman and McCanne, 1999] Raman, S. and McCanne, S. (1999). A model, analysis, and protocol framework for soft state-based communication. In

SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, pages 15–25, New York, NY, USA. ACM Press. *Cited in page(s):* 28

[Ran, 2003] Ran, S. (2003). A model for Web services discovery with QoS. *ACM SIGecom Exchanges*, 4(1). *Cited in page(s):* 23 , 24 , 66

[Ranganathan and Campbell, 2003] Ranganathan, A. and Campbell, R. H. (2003). A middleware for context-aware agents in ubiquitous computing environments. In *Proc. of Middleware, Lecture Notes in Computer Science, Volume 2672*. *Cited in page(s):* 11

[Rappaport, 2002] Rappaport, T. S. (2002). *Wireless Communications: Principles and Practice*. Prentice Hall. *Cited in page(s):* 8 , 26 , 45 , 46 , 48 , 57

[Ratsimor et al., 2002] Ratsimor, O., Chakraborty, D., Joshi, A., and Finin, F. (2002). Allia: Alliance-based service discovery for ad-hoc environments. In *Proc. of ACM mobile Commerce Workshop*. *Cited in page(s):* 22

[Raverdy et al., 2006] Raverdy, P.-G., Riva, O., de La Chapelle, A., Chibout, R., and Issarny, V. (2006). Efficient context-aware service discovery in multi-protocol pervasive environments. In *Proc. of Int'l Conf. on Mobile Data Management (MDM)*. *Cited in page(s):* 12

[Reibman and Veeraraghavan, 1991] Reibman, A. L. and Veeraraghavan, M. (1991). Reliability modeling: An overview for system designers. *IEEE computer*. *Cited in page(s):* 14 , 67

[Resnick et al., 2000] Resnick, P., Zeckhauser, R., Friedman, E., and Kuwabara, K. (2000). Reputation systems. *Communications of the ACM*, 43(12):45–48. *Cited in page(s):* 17 , 18 , 37

[Roman et al., 2000] Roman, G.-C., Picco, G. P., and Murphy, A. L. (2000). Software engineering for mobility: A roadmap. In *Proc. of 22nd ICSE*. *Cited in page(s):* 8

[Roman et al., 2002] Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K. (2002). A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 01(4):74–83. *Cited in page(s):* 10

[Roscoe and Bowen, 2000] Roscoe, T. and Bowen, G. (2000). Script-driven packet marking for quality of service support in legacy applications. In *Proc. of SPIE Conf. on Multimedia Computing and Networking*. *Cited in page(s):* 68

[Rothkopf et al., 1990] Rothkopf, M. H., Teisberg, T. J., and Kahn, E. P. (1990). Why are Vickrey auctions rare? *Journal of Political Economy*, 98(1):94–109. *Cited in page(s):* 82

[Royer and Perkins, 1999] Royer, E. M. and Perkins, C. E. (1999). Multicast operation of the ad hoc on-demand distance vector routing protocol. In *Proceedings of MobiCom*, pages 207–218. *Cited in page(s):* 53

[Sabata et al., 1997] Sabata, B., Chatterjee, S., Davis, M., Sydir, J. J., and Lawrence, T. F. (1997). Taxonomy for QoS specification. In *Proc. of Workshop on Object-oriented Real-time Dependable Systems (WORDS)*. *Cited in page(s):* 66

[Sabater and Sierra, 2001] Sabater, J. and Sierra, C. (2001). Regret: A reputation model for gregarious societies. In *Proc. 4th Workshop Deception, Fraud, and Trust in Agent Societies*. *Cited in page(s):* 102

[Saha and Mukherjee, 2003] Saha, D. and Mukherjee, A. (2003). Pervasive computing: A paradigm for the 21st century. *IEEE Computer*, 36(3):25–31. *Cited in page(s):* 5 , 6 , 11

[Sandholm, 1996] Sandholm, T. W. (1996). Limitations of the Vickrey auction in computational multiagent systems. In *Proc. of the 2nd Int'l Conf. on Multi-Agent Systems*. *Cited in page(s):* 82

[Sassone, 1988] Sassone, P. G. (1988). Cost benefit analysis of information systems: a survey of methodologies. In *Proc. of Conference on Supporting Group Work*, pages 126–133. *Cited in page(s):* 77

[Satyanarayanan, 1996] Satyanarayanan, M. (1996). Fundamental challenges in mobile computing. In *Proc. of Symposium on Principle of Distributed Computing (PODC)*. *Cited in page(s):* 7 , 10

[Satyanarayanan, 2001] Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *IEEE Personal Communications*. *Cited in page(s):* 5 , 7 , 11

[Seigel, 1988] Seigel, A. F. (1988). *Statistics and Data Analysis: An introduction*. John Wesley & Sons. *Cited in page(s):* 75

[Sen et al., 2005] Sen, R., Handorean, R., Roman, G.-C., and Gill, C. (2005). *Service Oriented Computing Imperatives in Ad Hoc Wireless Settings (Book Chapter)*, pages 247–269. Idea Group Publishing. *Cited in page(s):* 12 , 13

[Shakkottai et al., 2003] Shakkottai, S., Rappaport, T. S., and Karlsson, P. C. (2003). Cross-layer design for wireless networks. *IEEE Commun. Mag.*, 41(10). *Cited in page(s):* 48

[Su et al., 2001] Su, W., Lee, S.-J., and Gerla, M. (2001). Mobility prediction and routing in ad hoc wireless networks. *International Journal of Network Management*, 10. *Cited in page(s):* 26

[Toh, 1997] Toh, C. K. (1997). Associativity based routing for ad hoc mobile networks. *Wireless Personal Communications Journal, Special Issue on Mobile Networking and Computing Systems*, 4(2). *Cited in page(s):* 26

[Varshavsky et al., 2005] Varshavsky, A., Reid, B., and Lara, E. D. (2005). A cross-layer approach to service discovery and selection in MANETs. In *Proc. of 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*. *Cited in page(s):* 31

[Venkatasubramanian and Nahrstedt, 1997] Venkatasubramanian, N. and Nahrstedt, K. (1997). An integrated metric for video QoS. In *ACM International Multimedia Conference*. *Cited in page(s):* 29 , 77

[Vickrey, 1961] Vickrey, W. (1961). Counter speculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37. *Cited in page(s):* 16 , 36 , 65

[Vu et al., 2005] Vu, L.-H., Hauswirth, M., and Aberer, K. (2005). QoS-based service selection and ranking with trust and reputation management. In *Proc. of OTM conferences CoopIS/DOA/ODBASE*. *Cited in page(s):* 32

[Vulkan and Jennings, 2000] Vulkan, N. and Jennings, N. R. (2000). Efficient mechanisms for the supply of services in multi-agent environments. *International Journal of Decision Support Systems*. *Cited in page(s):* 35

[W3C, 2001] W3C (2001). Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`. *Cited in page(s):* 106

[W3C, 2002] W3C (2002). Web services activity. `http://www.w3.org/2002/ws/`. *Cited in page(s):* 106

[W3C, 2003] W3C (2003). SOAP version 1.2. `http://www.w3.org/TR/soap12-part1/`. *Cited in page(s):* 106

[W3C, 2004a] W3C (2004a). OWL Web Ontology Language Overview. `http://www.w3.org/TR/owl-features/`. *Cited in page(s):* 23

[W3C, 2004b] W3C (2004b). Web Services Architecture. `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`. *Cited in page(s):* 106

[Want and Pering, 2005] Want, R. and Pering, T. (2005). System challenges for ubiquitous & pervasive computing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 9–14, New York, NY, USA. ACM Press. *Cited in page(s):* 6

[Want et al., 2002] Want, R., Pering, T., Borriello, G., and Farkas, K. I. (2002). Disappearing hardware. *IEEE Pervasive Computing*. *Cited in page(s):* 8

[Weiser, 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific American*. *Cited in page(s):* 5 , 15 , 129

[Weiser and Brown, 1996] Weiser, M. and Brown, J. S. (1996). The coming age of calm technolgy. `http://www.ubiq.com/hypertext/weiser/acmfuture2endnote.htm`. *Cited in page(s):* 1 , 6

[Whitby et al., 2004] Whitby, A., Josang, A., and Indulska, J. (2004). Filtering out unfair ratings in bayesian reputation systems. In *Proceedings of the 7th Int'l Workshop on Trust in Agent Societies*. *Cited in page(s):* 39

[Williams and Camp, 2002] Williams, B. and Camp, T. (2002). Comparision of broadcasting techniques for mobile ad hoc networks. In *Proc. of ACM MobiHoC*. *Cited in page(s):* 51

[Wolski, 1998] Wolski, R. (1998). Dynamically forecasting network performance using the network weather service. *Journal of Cluster Computing*, 1(1). *Cited in page(s):* 72

[Wolski et al., 1997] Wolski, R., Spring, N., and Peterson, C. (1997). Implementing a performance forecasting system for metacomputing: the network weather service. In *Proc. of SuperComputing*. *Cited in page(s):* 71 , 119

[Wolski et al., 1999] Wolski, R., Spring, N. T., and Hayes, T. (1999). The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of future generation Computing Systems*, 15(5-6). *Cited in page(s):* 71

[Xiong and Liu, 2004] Xiong, L. and Liu, L. (2004). PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857. *Cited in page(s):* 37

[Xu et al., 2001] Xu, D., Nahrstedt, K., and Wichadakul, D. (2001). QoS-aware discovery of wide-area distributed services. In *Proc. of first IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*. *Cited in page(s):* 24

[Yu and Singh, 2002] Yu, B. and Singh, M. P. (2002). An evidential model of distributed reputation management. In *Proceedings of ACM AAMAS*. *Cited in page(s):* 17 , 39 , 94

[Yu and Singh, 2003] Yu, B. and Singh, M. P. (2003). Detecting deception in reputation management. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pages 73–80. *Cited in page(s):* 39

[Zacharia and Maes, 2000] Zacharia, G. and Maes, P. (2000). Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14:881–907. *Cited in page(s):* 17 , 38 , 102

[Zeng et al., 2003] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. Z. (2003). Quality driven web services composition. In *Proc. of WWW conference*. *Cited in page(s):* 66

[Zeng et al., 2004] Zeng, L., Benatallah, B., Ngu, A. H. H., Dumas, M., Kalagnanam, J., and Chang, H. (2004). QoS-aware middleware for Web services composition. *IEEE Transactions on Software Engineering*, 30(5). *Cited in page(s):* 72

[Zhong et al., 2003] Zhong, S., Chen, J., and Yang, Y. R. (2003). Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proceedings of IEEE Infocom*. *Cited in page(s):* 9

[Zhu et al., 2004] Zhu, F., Mutka, M., and Ni, L. (2004). PrudentExposure: a private and user-centric service discovery protocol. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*. *Cited in page(s):* 12

[Zhu et al., 2005] Zhu, F., Mutka, M. W., and Ni, L. M. (2005). Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, 4(4):81–90. *Cited in page(s):* 13 , 15

[Zimmermann, 1995] Zimmermann, P. R. (1995). *The Official PGP User's Guide*. MIT press. *Cited in page(s):* 16