QoS-aware Service Location in Mobile Ad-Hoc Networks

Jinshan Liu, Valérie Issarny
INRIA - Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France
Jinshan.Liu, Valerie.Issarny@inria.fr

Abstract

The advent of light-weight terminals (e.g., PDA) with integrated communication capabilities facilitates service access and hosting anytime, anywhere. However, effective service access requires dealing with the service's QoS, including related resource consumption. This paper introduces a comprehensive framework for QoS-aware service location in ad hoc networks. Our framework divides into: (i) QoS specification that is expressive enough to capture most significant QoS-related properties, and concise enough to have minimal computation cost associated with QoS management, and (ii) a benefit function for evaluating the overall benefit of service instances available in the network from the standpoint of QoS regarding both the user's perspective and resource consumption on hosts. Application of the proposed QoS framework is addressed in the context of the mobile AdHocWS system, which supports QoS-aware location of Web services deployed in mobile ad hoc networks.

1. Introduction

The vision of *ambient intelligence* (also referred to as *pervasive computing*) is among the most challenging topics for next generation computing systems. Realizing the vision means that consumers will be provided with universal and immediate access to available content and services anytime, anywhere, together with a way of effectively exploiting them [1, 9, 11, 13]. In particular, content and services should be made available to mobile users. Such a requirement further requires addressing deployment of services on possibly thin, wireless devices for increased availability [19, 12, 25]. A key part of effectiveness then lies in the specification of mobile service properties and the dynamic composition of services, in a way that both minimizes resource consumption on mobile terminals and satisfies users' request with respect to the offered Quality of Service (QoS). The resource limitation of thin wireless computing devices makes QoS-aware resource management even more important.

QoS in mobile environments relates to characterizing capabilities of mobile hosts. This is addressed by the CC/PP[1] W3C standard, which describes devices' capabilities and preferences, with a focus on wireless devices. Such a specification is sent along with any request from the device so that the server can filter and adapt to the device's requirements. CC/PP can be utilized to describe richer context information, including location and QoS [16]. However, using CC/PP for supporting QoS management is an open issue and needs to be coupled with network-related QoS specification.

Work within the Internet Engineering Task Force (IETF) addresses the issue of providing seamless handover with support for the exchange of QoS information within the SeaMoby working group[2], and the one of QoS provisioning over mobile IP [7]. The IST BRAIN project investigates service adaptation, scalability, QoS handling and management on mobility-enabled IP based networks[3]. However, the above work only addresses QoS at the level of the wireless network, and thus does not deal with QoS related to terminals (e.g., CPU load).

In general, supporting QoS in mobile environments remains an open issue, considering in particular the necessary exploitation of ad hoc wireless networks in addition to infrastructure-based ones for enforcing ubiquitous networking [22]. This paper contributes to the above issue by introducing a comprehensive framework for QoS-aware service location in ad hoc networks. By ad hoc networks, we mean networks of light-weight terminals (e.g., PDA), which are formed in an

1 Composite Capabilities/Preference Profiles; see http://www.w3c.org/Mobile/CCPP/

2 The Context and Micro-Mobility Routing Working Group, http://www.ietf.org/html.charters/seamoby-charter.html

3 BRAIN: Broadband Radio Access for IP based Networks. http://www.ist-brain.org/

ad hoc manner (i.e., without any infrastructure); terminals further have varying and scarce resources (e.g., CPU, network, battery) and may all act as service hosts.

The work presented in this paper is part of the Ozone project[4]. This paper is organized as follows. Section 2 introduces QoS specification, which is expressive enough to capture most significant QoS-related properties, and concise enough to have minimal computation cost associated with QoS management. Section 3 further details a solution to the measurement and prediction of QoS values, specifically targeted at thin devices. Section 4 then elaborates the benefit function for assessing the overall QoS of service instances, which takes into account both users' benefit and hosts' resource consumption. It is followed by the application of the QoS framework to the mobile AdHocWS system in Section 5; AdHocWS supports QoS-aware location of Web services deployed in mobile ad hoc networks. Finally, conclusions and future work are discussed in Section 6.

2. QoS specification

QoS specification associated with services is concerned with capturing the services' QoS requirements and policies. QoS specification should: (1) allow description of both quantitative (e.g., service latency) and qualitative (e.g., CPU scheduling mechanism) QoS attributes, as well as adaption rules [24]; (2) be declarative in nature, that is, specify only what is required, but not how the requirements are implemented [2].

In the following, QoS *category* refers to a specific non-functional property of a service that we are interested in (e.g., *performance*). Every category consists of one or more *dimensions*, each representing one attribute of the category. For instance, *latency* defines a dimension of the performance category. Quantitative dimensions in QoS specification, also referred to as *metrics*, measure specific quantifiable attributes of the service. Qualitative dimensions, referred to as *policies*, dictate the behavior of the services. Sabata et. al. further classify the metrics into categories of *performance*, *security levels*, and *relative importance*. Policies are divided into categories of *management* and *level of service* [24].

Based on the aforementioned work, we introduce a solution to the QoS specification of services that is adapted to mobile ad-hoc environments. In particular, we notice that although more QoS parameters yield more detailed description, the gain has to be put up against the increased overhead. Usually a small number of parameters (i.e., ≤ 5) is sufficient to capture the dominant QoS properties of a system [10]. Along with the factor of limited resources on mobile devices, we only take into account the most dominant and descriptive dimensions in our base QoS specification instead of trying to incorporate every applicable dimension. More dimensions can further be added, if required by specific services, by supporting the new dimensions in a way similar to the one discussed in the following.

Our specification, similar to [24], incorporates *metrics* and *policies*. Moreover, we consider that in mobile and resource-constrained environments, the service location process not only needs to consider the users' benefit (i.e., satisfying the user's request on QoS' aspects), but also needs to take into account the resulting resource consumption on hosts. Hence, our metrics cover both service-level (§ 2.1) and resource-level (§ 2.2) metrics. In addition, we consider that it is important to describe the service's adaption and level of service with respect to offered QoS, for increased flexibility in the service selection process in mobile environments. Hence, we include both of them in our policies (§ 2.3). Moreover, we use *relative importance* to characterize both the users' preferences among the various QoS dimensions and the criticality of the hosts' resources (§ 2.4).

2.1. Service-level Metrics

Service-level metrics capture the users' QoS-related requirements for the target service. Based on [24], our specification includes the *performance* and *security* categories. We further include an important category, i.e., *reliability*, which basically must be taken into account systematically for any system [21]. Also, as mobile systems feature "continuous change is the only constant", it is necessary to emphasize the reliability of services in the unpredictable environments. In addition, we include the *transaction* category since many real-life services are transactional.

Intuitively, people refer to performance when talking about QoS. That is because performance is the most influential system parameter in the observed behavior of applications. Common performance dimensions include latency (response-time), jitter and throughput. In most cases, users care most about getting their answers for their requests quickly and accurately. It is closely related to user-friendliness due to the human's nature of limited patience. Hence *latency* stands itself as the most dominant dimension in the performance category. Throughput is more server and network-oriented. We will thus talk about network's bandwidth, which is a

4 http://www. extra.research.philips.com/europrojects/ozone

form of throughput, in the resource-related specification. Jitter, the difference between the maximal and minimal latencies, can be used to predict the service time more accurately since the latency is given as a form of mean response time in most occasions. However, we only consider latency, which is expressed as the time between initiation of the request and arrival of the result, as the basic performance dimension.

Reliability of software systems is becoming increasingly important as the cost of down-time increases [8]. Reliability can be characterized by a number of dimensions (e.g., availability, continuous availability, mean time to failure, number of service failures) [21]. They each depict one aspect of reliability. However, not all of them are required by every service since reliability requirements vary significantly according to the application domain. In general, *availability*, which characterizes the probability that a service is available when accessed at any time, is the most commonly used dimension for characterizing reliability. We thus consider only availability as the basic reliability dimension.

Security deals against unauthorized attempts to access information. It is concerned with confidentiality, integrity and availability. The security taxonomy also includes encryption, authentication, access control and non-repudiation [17]. Encryption, authentication and access control are security functionalities to enforce security properties such as confidentiality, integrity and non-repudiation. Since we already have availability in the reliability category, we do not consider it in the security category. We thus include the dimensions of *confidentiality*, *integrity* and *non-repudiation* in our QoS specification. They are all boolean.

The transactional category relates to provisioning fault tolerance measures and thus to the reliability category. However, since many real-world services are transactional (e.g., ticket booking, stock trading, online banking, etc), we list it as an individual category for its prevalence and significance in the real world. The transactional category refers to whether a service offers ACID (Atomicity, Consistency, Isolation and Durability) properties or a subset of them [14], leading to the definition of the corresponding dimensions, which are all boolean.

2.2. Resource-related Metrics

Different QoSs lead to the consumption of different amounts of resources. Due to the limited resources of wireless service hosts in mobile environments, it is natural to prefer resource-richer hosts to resource-poorer ones. For example, a plugged PC is certainly preferred over a PDA when both host instances of the same service that offers the same quality, especially if the service consumes a fair amount of battery. More importantly, without consideration of associated resource consumption, the user will always request the best service and shared resources will be misused [29]. We thus introduce the *resource* category to depict the service's resource consumption on hosts. The following four dimensions are commonly used to characterize terminals' resources: *CPU load*, *memory*, *bandwidth* and *battery*. Thus, we include the consumption of the above four resources in the resource-related specification of services.

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:

$$rc_{s,r} = \frac{ac_{s,r}}{tar_r} \quad (1)$$

The reason that $rc_{s,r}$ is defined as a percentage instead of absolute value is that the same amount of resource is 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 left than on a host with 10MB left). The following characterizes the values of ac and tar for the above four dimensions of the resource category.

The *CPU load* describes the work load on the CPU(s) of a host. The CPU load of a host is either defined as the number of running processes during some time, e.g., 5 minutes, or the utilization percentage of the CPU. They are closely related, but far from identical. The system may have plenty of processes running (heavily-loaded) but a low utilization percentage when most of the processes are I/O bound instead of CPU-bound. Hence, we define the available CPU load (i.e., tar_{CPU}) of a host as the utilizable percentage of the CPU (i.e., 1 - utilization percentage). The CPU load introduced by a service s (i.e., $ac_{s,CPU}$) is further defined as the CPU time of the service divided by the total service time[5].

Memory refers to the size of the primary memory of a host. The available memory of a host (i.e., tar_{memory}) is thus defined as the host's available primary memory, and the *memory consumption* of a service s (i.e., $ac_{s,memory}$) is defined as the total amount of physical memory used by the service. Available memory of a host is a key factor affecting service performance. The more available memory is, the less possibility the system needs to read data from disk into memory, and the less service latency is.

5 Note that total service time refers to the time the host executes the service, i.e., it does not include the time waiting for network or I/O-related events.

The available *bandwidth* for a host (i.e., $tar_{bandwidth}$) represents the actual capacity of the connection and the *bandwidth consumption* of a service s (i.e., $ac_{s,bandwidth}$) refers to the volume of data the service sends and receives per some time units. Strictly speaking, bandwidth is not a property of the host, but of the network a host is attached to. Since it also reflects the physical conditions of the environments a host is resident in, we consider bandwidth consumption as a dimension depicting resource consumption of a service instance.

Battery longevity is an important part of the mobile world and battery consumption is an important resource-related dimension of a service. The available battery (i.e., $tar_{battery}$) for host represents the power level a host has (an AC plugged-in host is considered to have infinite battery). The *battery consumption* of a service s (i.e., $ac_{s,battery}$) refers to the power a host consumes for executing the service.

2.3. Policy

Compared to metrics-based categories, policies are qualitative and behavior specific. We introduce the category of *service behavior* to describe the service-specific QoS-related policies. More specifically, our QoS specification includes the dimensions of *service adaption* and *level of service*, which are the dominant ones in unpredictable mobile environments.

The service adaption dimension captures the degree of QoS adaption that the service can tolerate and scaling actions to be taken in the event of violation in the contracted QoS [2]. Bowers et. al. introduce *adaption space* for specifying when various implementation alternatives of a software component are feasible and when they should be selected, based on current conditions of the application environment and the user preferences [4]. Each implementation alternative meets the basic goals but differs in the level of service that it provides. Narayanan et. al. use *fidelity* to represent an application-specific notion of the "goodness" of application result. In general, lower fidelity consumes less resources, but at the cost of providing a degraded service to the user [23]. In our framework, we use the concept of *fidelity* to represent an implementation alternative of the service, and to represent an acceptable level of service to the users. The user then lists in his/her service request, his/her acceptable fidelities and the associated service-level QoS specification (e.g., fidelity excellent: availability $> 90\%$ & latency $<$ 0.01ms; fidelity fair: availability $> 60\%$ & latency $<$ 0.1ms). In the same way, the service lists in its advertisement, its provided fidelities and associated service-level and resource-level specifications (e.g., fidelity excellent: availability $> 90\%$, latency $<$ 0.01ms, memory $> 20\%$; fidelity fair: availability $> 60\%$, latency $<$ 0.1ms, memory $> 10\%$). In the initial negotiation, the system tries to satisfy the request from the highest fidelity to the lowest. During runtime, when the availability of resources varies significantly, the user can either enable renegotiation to be able to continue or improve the service; or simply disable renegotiation. Thus, possible service adaption policies are: *renegotiation required when resources improve*; *renegotiation allowed when resources deteriorate*; *no renegotiation when resources improve*; *no renegotiation when resources deteriorate*, or meaningful combination of any of them.

The *level of service* dimension specifies the required degree of end-to-end resource commitment [2]. It allows the qualitative distinction between hard and soft performance guarantees. Optional values of level of service include *deterministic*, *predictive* and *best effort* [2]:

- Deterministic, which is also called *guaranteed* or *hard guarantee*, refers to services whose QoS will be surely met as long as hardware is functioning and the service conforms to its resource consumption characterization. It is typically based on static resource reservation and used for hard real-time performance applications.
- Predictive, or *probabilistic*, is usually based on shared resource allocation to utilize resources efficiently. It allows for a certain percentage of violations and the service may suffer from degradation from time to time. It is particularly suitable for continuous media applications.
- Best-Effort is the lowest priority commitment. No resources are allocated or monitored because service providers are not obliged to any level of service. This commitment level is only able to receive whatever resources are left after the other levels are serviced.

2.4. Relative Importance

The respective importance of both quantitative and qualitative QoS dimensions depends on the user and environment. Some users may want to get results as quickly as possible, while others may consider security as their first priority. We use *relative importance* to characterize both the user's preference regarding each QoS dimension in the user's request, and the criticality of the various resources for hosts. Relative importance is introduced as a weight over dimensions, which applies to both quantitative and qualitative dimensions.

Hence, every time a service client looks up some service instance (e.g., look for a restaurant) matching cer-

Importance Type	Weight (Priority)
Don't care	0
Trivial	1
Slightly Important	2
Important	3
Very Important	5
Ultra Important	7
Deadly Important	9

Table 1. Importance table

tain functional (e.g., Chinese Restaurant in Versailles) and non-functional properties (latency < 30 seconds), the service request not only includes the target properties, but also the user's preferences (importance) to each non-functional property. In the same way, service advertisements from a service host not only include the estimated resource consumption, but also the criticality of the resources to the host (e.g., battery is very very important to the host since it has some important tasks to execute in the future; while CPU load is not so important to it since its CPU is now idle most of the time).

Mapping from the user's preferences and the resources criticalities (mostly some text) to some importance value may seem too casual at first glance. But, as long as the users/hosts realize the order of importance that the text represents, it is reasonable to map it into some specific value. Also there is some practice before, e.g., Gray and Reuter define availability classes according to the system availability [14]. Similarly, we define an importance table (Table 1) stating the mapping between importance and weight. The reason why weights have 2 spaced in between instead of 1 from *important* is because we consider that it is necessary to spread the weight space to make the difference more expressive (e.g., *deadly important* is much more important than *important*, and values $-$ 9 and 3 for this case $-$ need to reflect the difference).

Table 2 summarizes base QoS categories and related dimensions that we have introduced for the QoS specification of services. Such a specification allows for the selection of mobile services according to offered functional *and* non-functional properties among the service instances that can be retrieved in the environment using base capabilities of existing service location protocols. However, some services may need to be selected according to more dimensions (e.g., a distant surgery service requires the *continuous availability* dimension). Addition of dimensions only requires users to add new dimensions within the service request. Then, if a service instance fails to provide those new dimensions, which must be further managed by the host, this service instance is not considered qualified for the user's request.

Category	Dimension	Definition
Reliability	Availability	Probability=[0..1]
Performance	Latency	Service Time (in ms)
Security	Confidentiality	Boolean
	Integrity	Boolean
	Non-repudiation	Boolean
Transaction	Atomicity	Boolean
	Consistency	Boolean
	Isolation	Boolean
	Durability	boolean
Resource	CPU Load	Percentage=[0..1]
	Memory	Percentage=[0..1]
	Bandwidth	Percentage=[0..1]
	Battery	Percentage=[0..1]
Service Behavior	Service Adaption	adaption policy
	Level of Service	service level

Table 2. QoS dimensions in our base specification

3. QoS Measurement

The specification of quantitative dimensions in service requests and advertisements requires providing as accurate measures as possible. Service-level dimensions can be measured easily (e.g., off-line measurements using available quality analysis tools). Resource-related measures for the services are also easy to obtain *after* service execution, using available utilities (e.g., pathchar[6] for bandwidth measurement). However, providing accurate resource consumption measures (i.e., tar_r, $ac_{s,r}$) for the selection of service instances *prior* to their execution requires special care, since this relates to predicting the service's resource consumption.

Existing work about resource consumption prediction undertakes two approaches: analytical and statistical. The analytical approach is application-specific, and administrators or developers are required to analyze the algorithm complexity and produce an expression. Its applicability is limited to those applications whose complexity expression is not hard to come up with. The statistical approach utilizes cumulative statistics data from previous runs and tries to find some rules for future prediction-making. It applies to those situations

6 http://www.caida.org/tools/utilities/others/pathchar

when resource consumptions do not vary much from run to run.

Narayanan et. al. offer a hybrid approach to predict resource consumption of applications [23]. It is defined as a function of fidelity, as defined in Section 2.3, (e.g., $CPU\ Consumption = c_0 + c_1 \times p \times r \times log(p \times r) + c_2(p \times r)^2$, where p and r are input parameters or fidelities). More specifically, the prediction function is obtained through combination of logs about resource consumption after running applications at different fidelities in off-line training, with application-specific hints from the analysis of the application complexity. After each run, new results are logged and coefficients (e.g. c_i in our example) are recomputed. However, in [23], the application's developer has to offer the estimation of computation complexity after off-line training. Although provided prediction can be refined after new runs, the computation complexity, which can be predicted wrongly at the early stage, cannot be changed. Using cumulative statistics, one can predict resource consumption associated with a request according to similar requests in the history, under the assumption that similar requests lead to similar results [20, 28]. Similarity is defined with respect to the input (e.g., same in application name, user name, executable arguments, etc) . Kapadia et. al. go further by proposing a prediction approach, named two-level knowledge-based organization, which exploits the principle of temporal and spatial localities [20]. However, it is mostly applicable to academic environments in which students tend to work on the same homework at the same time. When a host may have requests from any terminal as in a mobile environment, this organization is of limited usage.

To solve the aforementioned issue, our approach exploits the excellent properties of normal distribution. Normal distributions are symmetric with scores more concentrated in the middle than in the tails (Fig 1). Applied statisticians have confirmed by observation that the normal distribution is often adequate as a description of many data sets that appear in the "real" (i.e., non-mathematical) world [27]. Many kinds of behavioral data are approximated well by normal distribution. Furthermore, most real phenomena generate distributions that are close to normal, and long-tailed distributions and modal distributions can be successfully approximated by normal distributions [26].

Hence, we consider that services' resource consumption under one fidelity is regular and not random, and that the resource consumption distribution is very close to normal distribution. Our experiments have further proved it. We further use off-line training as a starting point, as in [23]. However, the difference between our solution and the one of [23] is that we do not require the service's developer to come up with the predicted linear regression at the beginning. Our approach is thus much simpler and computationally cheaper.

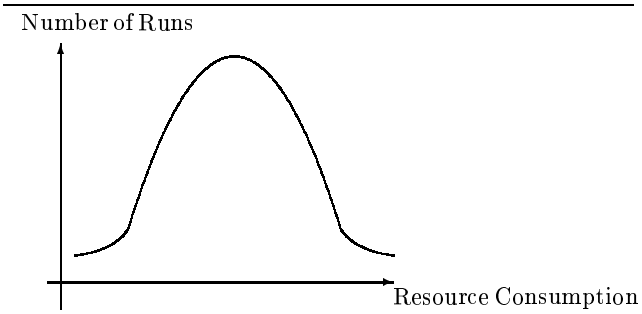

Figure 1. Normal Distribution

Amount	Probability of less than amount
$m + 1.282 \times \delta$	90%
$m + 0.53 \times \delta$	70%
m	50%
$m - 0.53 \times \delta$	30%
$m - 1.282 \times \delta$	10%

Table 3. Probability that a standard normal random variable is less than a value

From the property of normal distribution, we have a table listing the probability that a standard normal random variable is less than a value for mean value m and standard deviation δ (Table 3). Therefore, if we want to have conservative prediction (i.e., there is heavy penalty for misprediction), we give the prediction of $m + 1.282 \times \delta$. For more aggressive prediction (i.e., there is fair penalty for misprediction), we give the prediction of m. For even more aggressive (i.e., there is little penalty for misprediction), $m - 0.53 \times \delta$ is chosen.

4. Overall QoS

Given the QoS specification of a service, related dimensions must be combined to evaluate the service's overall QoS. This is realized using a benefit function. Benefit function is used to find the *Operational Point* that maximizes the users' benefit under current conditions. It is very useful in enabling the service's adaptation policy by supporting graceful adaption when resource conditions suddenly deteriorate or improve. It further serves selecting the best service instance from a pool of eligible ones (i.e., instances offering the target functional properties), as identified using base functionalities of exisiting service location protocols.

4.1. Benefit Function

A benefit function is a model proposed to depict the benefit the user receives for a certain level of QoS [24]. It is widely used in resource management or job scheduling. Levin and Irvine develop a metric for evaluating resource-usage efficiency by incorporating into its benefit function: user's preference, job's priority, resource consumption and the efficiency of job scheduling [18]. Nahrstedt et. al. explore an aggregated approach by constructing a benefit function, which is a resource cost/user satisfaction ratio; user satisfaction is decided by the performance of video presentation (e.g., delay, skew) and resource consumption is proportional to the consumption percentage of the bottleneck resource (i.e., the resource that the service consumes relatively most) [29].

In a way similar to [29], we account for user's benefit and resource consumption for the definition of our benefit function. The difference between [29] and our approach is that we take into account the consumption of all the resources instead of only the bottleneck resource. Hence, we have the following formula:

$$\text{Overall Benefit} = \text{User's benefit / Service Cost} \quad (2)$$

Let us denote the set of QoS dimensions about which the user cares (i.e., the user may only care about part of the available QoS dimensions) D, and the number of properties D as n. Each property d_i of D further has an associated weight w_i (i.e., relative importance). We then define the user' benefit as follows:

$$\text{User's Benefit} = \sum_{i=1}^{n}(d_i * w_i) \quad (3)$$

Note that if d_i is of a dimension whose value space is either name literals (e.g., the value space of *level of service* is *deterministic*, *predictive*, *best-effort*) or boolean, d_i equals to 1. In addition, if d_is are of different dimensions, data normalization is needed, which is elaborated in Section 4.2.

Assuming further a service s that consumes m kinds of resource and that for each resource i of weight w_i, the service has resource consumption $rc_{s,i}$. We evaluate the service cost (resource consumption) by:

$$\text{Service Cost}_s = \sum_{i=1}^{m}(rc_{s,i} * w_i) \quad (4)$$

4.2. Data Normalization

In order to combine the various QoS dimensions of a service using our benefit function, data normalization is necessary. There are many ways of normalizing data. Two simple and effective normalization techniques are: (i) Decimal scaling, (ii) Standard deviation normalization.

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

Table 4. QoS values before and after decimal scaling

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:

$$d'(i) = d(i)/(10^k) \quad \text{for the smallest k such that } max(|d'(i)|) < 1 \quad (5)$$

Decimal scaling preserves most of the original character of the value, and a typical scale maintains the values in the range of -1 to 1. In our framework, we need to make the effect between two values during normalization as small as possible to keep the comparison as fair as possible. For example, consider 3 service instances, 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 weight (e.g., both are *very important*). After decimal scaling (d_1 becomes d'_1 and d_2 becomes d'_2, as shown in Table 4), c is obviously the best instance considering the sum of the two dimension values. But, we do not want to select c, since c does not have balanced properties (i.e., it is strong in d_1, but too weak in d_2). When data values range over a wide range of magnitudes, it may be difficult to work with them, comprehend them and properly compare them to others without the aid of transformation. This is because the large values dominate the situation, which makes it difficult to see other details in the rest of data [27]. Hence, we use standard deviation normalization in our work.

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

$$d'(i) = [d(i) - m(d)]/\delta(d) \quad (6)$$

where $d(i)$ is the value of dimension d for the service instance i, and $m(d)$ and $\delta(d)$ are the mean value and standard deviation for dimension d, respectively. In contrast to the resource estimation described in Sec-

	d_1	d_2	d'_1	d'_2
a	10	50	0.71	1.39
b	30	40	0.72	1.19
c	1000	1	1.58	0.44

Table 5. QoS values before and after standard normalization

tion 3, we cannot assume that the values of the service's QoS metrics (i.e., values of both service-level and resource-related dimensions) are normally distributed since they do not bear any relationship between each other. However, we have:

> **Chebyshev's theorem**: The portion of data that lies within k standard deviations to either side of the mean is at least (1-1/k^2) for any data set, where k is a number greater than 1.

We further consider that a 75% confidence interval (i.e., by considering the other 25% as extremities) is approximate enough. We thus let $k = 2$, leading to divide the space into $(-\infty..m - 2 \times \delta]$, $(m - 2 \times \delta..m + 2 \times \delta]$ and $(m + 2 \times \delta.. + \infty)$. Hence, we normalize the data according to the following:

$$d'(i) = \begin{cases} 2 & if(d(i) - m(d) > 2 * \delta(d)) \\ 0 & if(d(i) - m(d) < -2 * \delta(d)) \\ \frac{d(i)-m(d)}{2*\delta(d)} + 1 & otherwise \end{cases} \quad (7)$$

Note that for QoS parameters that are stronger with smaller values (e.g., latency), d'_i is further transformed by $d''_i = 2 - d'_i$, so that smaller values contribute more to the benefit. Table 5 lists the values after transformation for the same example as above. Obviously, a is the best instance in this case. Standard deviation normalization works better in picking up the most balanced instance instead of one that is really strong in one category while too weak in another category.

We thus apply the above normalization to the user's benefit function (Equation 3). The resulting normalization of d_is, leads to $0 < d_i < 2$, and user's benefit > 0. We do not apply normalization to the service cost function (Equation 4) because $rc_{s,i}$ is defined as a percentage and its values for the various resources are already comparable to each other. Thus, the value of the overall benefit function (Equation 2) falls between $(0..+\infty)$. In the special situation where the user does not have any preference for any service-level QoS dimension (i.e., he/she does not really care which QoS property is more important to him/her), the user's benefit is assigned to 1. Thus, the instance with the least service cost have the highest overall benefit.

5. Experiment and Evaluation

In order to assess our QoS framework, we integrate it in the mobile AdHocWS system that supports sharing of Web services running on wireless nodes of an (one hop) ad hoc WLAN. We select AdHocWS because of the Web technology's pervasiveness, which supports availability of services in most situations [19], and the increasing interest for making Web services available on thin devices (e.g. [12, 25]).

5.1. AdHocWS System

AdHocWS implements the following two functionalities relevant to QoS-aware service location: (a) service location by creating and maintaining ad hoc dynamic groups of mobile nodes, which is based on [3], (b) service advertisement, selection and update.

Ad hoc group management in AdHocWS builds upon an existing service discovery protocol, i.e., the IETF Service Location Protocol (SLP) [15]. We further use an SLP configuration that does not use any directory agent, i.e., each node acts as an SLP *Service Agent*. In our ad hoc network, peer nodes are grouped into *security domains* in which peers trust each other and share domain-specific keys [3]. The peers in a security domain form a community. Upon initialization, each node registers itself for each security domain of AdHocWS, using the format: `service: AdHocWS://IP_address:port_number/domain`. Periodically, each mobile node looks for peers by issuing the following service request: `service: AdHocWS`. Every mobile node in the communication range sends back as many security domains as it belongs to. Each reply message contains the URL associated with the given security domain, which embeds the terminal's IP address, port number, domain, unique identifier (UID). After receiving URLs, nodes are able to join the ad hoc group they belong to. Initially, all peers are isolated. The ad hoc group is then created by one of the peers, named *leader*, which is the node that has the smallest UID.

The leader requests information about hosted services from the peers before propagating them along with group information (e.g., group members) to the peers in the community. Thus, each peer is able to cache the information (i.e., functional and QoS properties) about all the service instances available within the group, along with group member and leader information. Periodically, the peer with the next higher UID

than *leader* becomes the new leader. It checks the local peers with SLP as described before, and compares the list with former member list. If the two are not identical, it indicates that either new members join in or old members leave or both. The new leader then communicates with the peers which newly join in and forms a new community.

5.2. QoS-aware Service Location

During group initialization, when the leader requires from every peer its hosted services, each peer submits services advertisement that not only contains the services' functionality and endpoint information, but also the services' QoS properties according to QoS specification detailed Section 2. All service advertisements are then broadcasted by the leader to every peer in the community after the leader finishes collecting the advertisements from peers. Hence, upon the initialization of a group, each peer caches on its own terminal the specification, including the QoS-related one, of all the service instances hosted by all the peers in the community.

When one peer is seeking a service, it looks up the pool of service instances that is locally cached. For every instance, it first carries out conformance checking with respect to both functional and QoS properties (i.e., conformance of the instance's interface with the target service's interface, and conformance of respective QoS values). If there is no qualified instance, the user can either adapt to lower fidelity (if there is an instance offering the target functional properties), which requires less resources, or may try later. If there is only one qualified instance, the instance is selected. If there are multiple qualified instances, the selection process proceeds to decide which of the qualified instances is the best in terms of both user's benefit and server's cost. The overall benefit for each service instance is computed for each instance using the benefit function defined in Section 4, and the instance with the highest overall benefit is selected.

Once a service instance is selected, the client sends a service request to the host of the selected instance. If the host accepts the request (it may possibly decline the request if resources suddenly deteriorate due to, e.g., concurent requests), it sends an acknowledgment to the user and updates its profile of available resources. If the service is expected to last longer than some threshold value (e.g., 1 minute), the host sends the update to the group leader, which forwards it to every other peer. After the service is completed, the host returns the result to the requester and records the QoS properties of this run for more accurate future prediction. Table 6 lists the specific methods that are used in AdHocWS for updating the various dimensions of our QoS specification. In particular, the prediction of resource consumption relies on a statistical approach, as detailed in Section 3. After computing the new prediction, the host compares it to the old prediction. If the new prediction deviates from the old prediction over a threshold value (e.g., 5%), the host informs the leader for the update. Otherwise, the host simply discards the new prediction. Thus, we save communication volume by exploiting the fact that small fluctuation is quite common.

Dimension	Update Method
CPU Load, memory, bandwidth and battery	statistical approach
Latency	statistical approach
Availability	fixed at development time
Security/Transactional dimensions	fixed at development time

Table 6. Updating QoS dimensions

5.3. Experiment

The first prototype of AdHocWS has been implemented with JWSDP1.1[7] and JVMPI[8] in Java SDK 1.4. Terminals of the platform are 6 laptops with a 500MHz Pentium III CPU, 256KB of cache, 192MB of memory running under Linux 2.4.18 (Red Hat Linux 7.3). The wireless interface is 2.4 GHz DS lucent IEEE 802.11 WaveLan PC "Silver" 11Mbps.

We implemented a service of image distillation, which converts images to JPEG and degrade them according to the specified quality using the ImageMagick package[9]. The service serves the users' requests for images, which also specify JPEG quality factor [6]. Smaller quality factors lead to smaller size and less network volume, but at the cost of worse perceived quality, and conversely. The quality factor values is from 0 (worst) to 100 (best) with a default value of 75. Table 7 shows the size of images associated with different quality factors.

Upon request for some image from a client, the service host first checks whether the requested image with the specified quality factor is resident on the host, possibly leading to produce the image of specified quality. Regarding QoS measures associated with the service,

7 Java Web Service Developer Package 1.1 (TomCat 4 included)
8 Java Virtual Machine Profiler Interface
9 http://www.imagemagick.org

Quality	0	30	50	75	90	100
Image size (in bytes)	686	2446	3501	4538	6767	16988

Table 7. Image sizes and related JPEG qualities

Run	1	2	3	4	5	6	7	8	9	10
Latency (in ms)	89	116	94	97	93	93	102	108	87	101

Table 8. The latency of 10 runs of the Image Service (quality factor = 80, file size = 4761 bytes)

Table 8 gives the service latency of 10 runs for a quality factor of 80, and Table 9 gives the mean value and standard deviation of the latency and CPU load dimensions for each quality factor. From the table, we can see that the distribution closely resembles normal distribution and standard deviation is very small.

Assume a service request that requires $availability \geq 0.65$ and $service\ latency < 120\ sec.$, with relative importance of 3 and 7, respectively. Other dimensions are considered as not relevant for the service (i.e., their relative importance equals 0). Assume further that 4 qualified instances are available within the ad hoc group. The QoS properties offered by those instances are given in Table 10 where: a denotes availability, l denotes latency, a' denotes normalized availability, l' denotes normalized latency, CPU denotes CPU load, mem denotes memory, $band$ denotes bandwidth and $batt$ denotes battery, b denotes the user's benefit, c denotes the service cost and b/c denotes the overall service benefit. The resource-related dimensions (i.e., CPU load, memory, bandwidth and battery) are further accompanied by their relative importance. We can see from the table that although instance 4 provides the best service-level QoS (b = 14.23) but at the most expensive cost (c = 4.20). Instance 2 has the highest overall benefit (i.e., b/c) and is thus selected because it not only provides a decent benefit to the user ($b = 10.4$), but also has a comparatively low cost ($c = 2.08$).

Fidelity (quality factor)		0	30	50	80	100
Image size (in bytes)		686	2446	3501	4761	16988
Latency (in ms)	m	88.5	90.4	92.1	98.0	118.8
	δ	7.1	10.8	12.5	8.9	12.7
CPU Load (in 10^{-4}%)	m	1000.7	1000.5	998.8	1000.8	998.9
	δ	3.1	2.8	4.7	2.7	3.4

Table 9. Mean value and standard deviation of the ImageService

Instance	a	l	a'	l'	b	CPU	mem	$band$	$batt$	c	b/c
1	0.9	110.5	1.51	0.49	7.96	20/2	30/3	20/3	N/A	1.90	4.19
2	0.7	60.5	0.34	1.34	10.4	5/1	20/3	1/3	20/7	2.08	5.00
3	0.8	100.0	0.93	0.67	7.48	50/5	20/5	10/2	N/A	3.70	2.02
4	0.85	50.5	1.22	1.51	14.23	10/3	10/5	10/7	30/9	4.20	3.39

Table 10. Four service instances and their QoS properties

Number of Peers	2	3	4	5	6
Time (in ms)	11.3	84.88	129.29	224.11	385.78

Table 11. Ad-hoc Group initialization time

The computation required for the aforementioned selection process is computationally cheap and may thus be easily accommodated by resource-constrained nodes. Assume that the total number of instances is m, that the number of service-level dimensions about which a user care is n, and that the number of resource-related dimensions about which a host cares is k. It can be easily analyzed that the overall required computation for the selection process is $4 \times m + 4 \times m \times n + m \times k$.

The overall overhead of our QoS framework includes the group initialization time, the logging time, the computation required for service selection and the necessary service advertisement updates. Table 11 gives the initialization time for an ad-hoc group of different size, once peers of the group are discovered using SLP. The logging time takes about 30 microseconds. It is very acceptable compared to the service latency – for a latency that takes 80 ms, the increase in latency is less than 0.04%. From Table 9, we can see that standard deviation is small compared to the mean value, which shows that the QoS values vary in a small range and thus saves us from frequent updates. Therefore, the overall overhead of our QoS framework is affordable.

6. Conclusion

This paper has introduced a QoS framework for service location in mobile ad-hoc networks, so as to support QoS-aware service selection among service instances available on, possibly resource-constrained, wireless devices. Such a feature is becoming crucial as new computing paradigms like ambient intelligence and pervasive computing, are putting forward anytime,

anywhere access to services, which further calls for making services available on mobile nodes.

Our QoS framework supports extensible QoS specification for services on mobile terminals, and allows for efficiently locating service instances that optimize both the user's benefit and resource consumption on the host, according to a service request specifying required functional and non-functional (QoS) properties. By predicting service consumption values with normal distribution, we are further able to cache service advertisements on mobile terminals, which minimizes the communication cost associated with service discovery.

We have integrated our QoS framework with the mobile AdHocWS system for experiment. The resulting system supports sharing of QoS-aware Web services running on mobile nodes over an ad hoc WLAN. The experiment demonstrates the effectiveness of our approach in terms of both selecting the service instance that optimizes the overall QoS benefit, and performance of the selection process. AdHocWS is based on the definition of communities of peer mobile nodes where every node cooperates and is willing to service requests. However, in selfish environments, nodes are competing for resources and the service hosts intend to get the highest profit for a certain provided QoS and resource consumption. We are examining extension of our QoS framework with a price model in order to support sharing of services in most environments, as already suggested for systems based on ad hoc routing protocols. In particular, Buttyan and Hubaux propose a model of *nuglet counter*, which decreases when nodes send their own packets and increases when forwarding others' packets to stimulate cooperation and discourage misbehavior [5].

In the provided QoS specification, multiple dimensions (e.g., security dimensions) have been simplified to save computation. However, finer grained specification can be presented to make the system more amenable to meet overall user demands and more flexible for graceful adaption to changes in resource availability. We further include only two factors in our benefit function, i.e., users' benefit and resource consumption. Additional preference rules may be incorporated into the benefit function. For instance, considering that the execution of a service may be dependent upon others in the most general case, self-contained services may be preferred over composite one. It is part of out future work to examine further extension of our framework to take into account such features of services, while still offering a framework that can be accommodated by thin devices.

Acknowledgements

This research is partially supported by the European IST OZONE (Offering an Open and Optimal roadmap towards Consumer Oriented Ambient Intelligence) project (IST-2000-30026).

References

[1] E. Aarts, R. Harwig, and M. Schuurmans. Ambient Intelligence. In *The invisible Future: The Seamless Integration of Technology into Everyday life*. McGraw-Hill Professional, 2001.

[2] C. Aurrecoechea, A. T. Campell, and L. Hauw. A survey of QoS architectures. *ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture*, 3(6):138–151, May 1998.

[3] M. Boulkenafed and V. Issarny. AdHocFS: Sharing files in WLANs. In *the IEEE International Symposium on Network Computing and Applications (NCA-03)*, Cambridge, MA, USA, April 2003.

[4] S. Bowers, L. Delcambre, D. Maier, C. Cowan, P. Wagle, D. McNamee, A. F. L. Meur, and H. Hinton. Applying adaptation spaces to support quality of service and survivability. In *DARPA Information Survivability Conference and Exposition*, January 2000.

[5] L. Buttyan and J. P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM/Kluwer Mobile Networks and Applications (MONET 2002)*, 2002.

[6] S. Chandra and C. S. Ellis. JPEG compression metric as a quality aware image transcoding. In *2nd Symposium on Internet Technologies and Systems*, Boulder, CO, USA, October 1999.

[7] H. Chaskar. Requirements of a QoS solution for mobile IP. Internet Draft of IETF Next Steps in Signaling (nsis) working group, June, 2003.

[8] T. C. K. Chou. Beyond fault tolerance. *IEEE Computer*, 30(4), April 1997.

[9] M. L. Dertouzos. The future of Computing. *Scientific American*, 1999.

[10] H. V. Dijk, K. Langendoen, and H. Sips. ARC: a bottom-up approach to negotiated QoS. In *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000)*, December, 2000.

[11] M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next century challenges: Data-centric networking for invisible computing. In *Proceedings of MOBICOM'99*, 1999.

[12] R. Friedman. Caching Web services in mobile ad hoc networks: Opportunities and challenges. In *Proceedings of the 2nd ACM Workshop on Principles of Mobile Computing (POMC)*, 2002.

[13] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2), 2002.

[14] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, California, USA, 1993.

[15] E. Gttman, C. Perkins, J. Veizades, and M. Day. Service location protocol version 2. RFC 2608, June 1999.

[16] J. Indulska, R. Robinson, A. Rakotonirainy, and K. Henricksen. Experience in using CC/PP in context-aware systems. In *Proceedings of the 4th International Conference on Mobile Data Management (MDM)*, Melbourne, Australia, January 2003. LNCS 2574.

[17] C. Irvine and T. Levin. Towards a taxonomy and costing method for security services. In *Proceedings of the 15th Computer Security Application Conference*, Phoenix, AZ, December 1999.

[18] C. Irvine and T. Levin. Toward quality of security service in a resource management system benefit function. In *Proceedings of the 2000 Heterogeneous Computing Workshop*, pages 133–139, Cancun, Mexico, May 2000.

[19] V. Issarny, D. Sacchetti, F. Tartanoglu, and F. Sailhan. Enabling Ambient Intelligence via the Web. In *Proceeding of DNAC'16 De Nouvelles Architectures pour les Communications - La génération Wi-Fi et l'Internet Ambiant*, Paris, France, December 2002.

[20] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley. Predictive application-performance modeling in a computational grid environment. In *Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC)*, Los Angeles, CA, August 1999.

[21] J. Koistinen. Dimensions for reliability contracts in distributed objects systems. Technical Report HPL-97-119, Hewlette Packard, October 1997.

[22] P. Mohapatra, J. Li, and C. Gui. QoS in mobile ad hoc networks. *Special Issue on QoS in Next-Generation Wireless Multimedia Communications Systems in IEEE Wireless Communications Magazine*, June 2003.

[23] D. Narayanan, J. Flinn, and M. Satyanarayanan. Using history to improve mobile application adaptation. In *Proceeding of 3rd IEEE Workshop on Mobile Computing Systems and Applications*, pages 30–41, Monterey, CA, Aug. 2000.

[24] B. Sabata, S. Chatterjee, M. Davis, J. J. Sydir, and T. F. Lawrence. Taxonomy for QoS specification. In *Proceedings of Workshop on Object-oriented Real-time Dependable Systems (WORDS 97)*, Newport Beach, California, USA, 1997.

[25] J. Sairamesh, S. Goh, I. Stanoi, C. Li, and S. Padmanabhan. Self-managing, disconnected processes and mechanisms for mobile e-business. In *Proceedings of the 2nd International Workshop on Mobile Commerce*, 2002.

[26] J. M. Schopf and F. Berman. Performance prediction in production environments. In *Proceedings of the First Merged Symposium: 12th International Parallel Processing symposium and 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP'98)*, Orlando, Florida, March 1998.

[27] A. F. Seigel. *Statistics and Data Analysis: An introduction*. John Wesley & Sons, New York, USA, 1988.

[28] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In *Proceedings of the IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[29] N. Venkatasubramanian and K. Nahrstedt. An integrated metric for video QoS. In *ACM International Multimedia Conference*, Seattle, WA, USA, Nov 1997.