Service-oriented Network Sockets

Umar Saif and Justin Mazzola Paluska
M.I.T. Laboratory for Computer Science
{umar, jmp}@mit.edu

Abstract

This paper presents the design and implementation of service-oriented network sockets (SoNS) for accessing services in a dynamically changing networked environment. A service-oriented network socket takes a high-level description of a service and opportunistically connects to the best provider of that service in the changing characteristics of a mobile system. An application states its high-level service requirements as a set of constraints on the properties required in a suitable resource and SoNS continuously monitors, evaluates and compares the available resources and (re-)connects to the resource that best satisfies the specified constraints.

Unlike content-based routing systems, SoNS is an end-host system, interposed at the session-binding layer, and offers connection-oriented semantics. SoNS' interface allows an application to tailor the planning policy used to establish and rebind a network session. SoNS is based on an extensible architecture to leverage the wide-range of emerging technologies for discovering and locating resources in a mobile system.

SoNS integrates a service-oriented abstraction with the traditional operating system interface for accessing network services, making it simpler to develop pervasive, mobile applications. We present an implementation for a mobile handheld device, analyze the performance of our system and describe an application to demonstrate the utility of our system.

1 Introduction

Advances in digital electronics over the last decade have made computers faster, cheaper and smaller. This coupled with the revolution in communication technology has led to the development and rapid market growth of embedded devices equipped with network interfaces. It has also promoted the development and widespread use of battery-operated portable computers, allowing users to carry their computation resources and tasks with them.

These advances have led to the recent activity in pervasive systems [1][2]. MIT's project Oxygen [22], and related pervasive computing projects elsewhere, aim to define computational environments that would allow users to carry their mobile handheld devices from one networked environment to another, possibly disconnected, environment while providing personalized ubiquitous access to services in the environment of the user.

Such a system must be able to continuously adapt to changes in user locations and needs, respond both to component failures and newly available resources, and maintain continuity of service as the set of available resources change. This requires more than service discovery [3] or simple content-based routing [4]; it necessitates a certain degree of planning involving continuous reevaluation of available alternatives, as well as heuristic compromises to best address the application's requirement using imperfect resources in the changing environment of the application [5].

Such opportunistic access to system resources is contrary to what is offered by traditional mobile systems [6] that aim to *preserve* access to a mobile host as the characteristics of the system change. Such systems do not cater to *context-aware* applications [5][1][2] that desire to access the best provider of a service (henceforth referred to as a *resource*) in their environment, rather than maintaining access to a particular host.

Traditionally, such a context-aware application must itself provide the planning involved in accessing the best available service-provider in its environment. Such applications typically contain a planning component that continuously reevaluates the available alternatives and provides access to the best available service-provider. These planning components often employ a resource discovery system to find the available alternatives and use the operating system socket interface to establish and rebind network connections as better alternatives become available. Most context-aware and adaptive applications layered on top of traditional operating systems and network routing architectures are examples of this model [7].

Where the above-mentioned model has the virtue that the application is free to use any arbitrarily complex planning policy befitting its requirements, allowing the underlying system to be policy-neutral, it requires every application to be capable of discovering, monitoring, evaluating and comparing the available alternatives in order to utilize the best available service-provider in its environment. In a pervasive computing environment, where such *opportunistic* access to service-providers is a *norm*, it is clearly desirable to separate this complexity in a *re-usable planning layer* that can be employed by different applications to

opportunistically access resources in a dynamically changing networked environment.

Among the existing systems, the Intentional Naming System (INS) [4] comes closest to achieving this goal. The late binding architecture of INS allows an application to send *intentional datagrams* that carry a description of the properties of the required service, instead of the network address of a host, and an overlay of INS resolvers route these datagrams to the hosts that match the service description. Where this scheme of integrating service location and message routing alleviates an application from the task of continuously monitoring its environment and rebinding its network connections when a better alternative becomes available, INS provides limited planning for choosing the closest match to application requirements when more than one resource matches a service description. In this case INS simply relies on an application-level anycast to all the matching resources.

Even though it is conceivable that a more elaborate scheme could lead to more informed routing decisions, this approach of handling the dynamism of the system at the routing level inherently suffers from the following problems.

- The planning policy, used to select the best match to application requirements, is hidden from the application in the routing infrastructure and, worse, distributed in the network. Therefore, it cannot be tailored to suit the requirements of the various different applications found in a pervasive mobile system.
- Such content-based routing systems [4][8] only provide connection-less datagram semantics; every datagram carries the required service description which is resolved by, often an overlay of, network resolvers to deliver the message to an appropriate host. Therefore, such systems lack application-level session semantics, in that there is no concept of an application-level connection; two successive datagrams generated by an application can be routed to two different hosts, transparently to the application. This coupled with the characteristic fluctuations in the performance of wireless links and mobile hosts, means that an application has little control over which resource gets accessed, precluding applications with inherently connection-oriented semantics e.g. multi-media streaming applications. Such a system is also prone to *thrashing* between service-providers in the presence of frequent performance fluctuations and node failures.
- From a performance point of view, content-based routing, performed by resolving complex service descriptions at every hop in an overlay network, is considerably slower than traditional address-based network routing [4] since it introduces the cost of resolving a service description to a network address in the critical path of message delivery. Furthermore, including a full service description of the required service with every network message is wasteful of the scarce bandwidth available in a wireless network.
- Finally, content-based routing systems introduce a new API for network communication [4][21], which is often different from the traditional operating system interface, for accessing services in the system.

We propose Service-oriented Network Sockets (SoNS) to access services in a highly dynamic networked environment. A service-oriented network socket takes a high-level description of a service and opportunistically connects to the best provider of that service in the changing characteristics of a mobile system. An application states its requirements as a set of constraints on the properties required in a suitable resource and SoNS continuously monitors, evaluates and compares the available resources and (re-)connects to the resource that best satisfies the specified constraints.

Unlike content-based routing systems, SoNS is an end-host system, interposed at the session-binding layer, and offers connection-oriented semantics. Most importantly, SoNS allows an application to configure, and even replace, the planning policy used to evaluate and compare available alternatives and the semantics used for rebinding a network connection when a better alternative becomes available. SoNS integrates a service-oriented abstraction with the traditional operating system interface for accessing network services, making it simpler to develop pervasive mobile applications.

We favor this approach over a content-based routing scheme as it handles the dynamism of a mobile system at the stage of binding a network connection at an end-host, and hence 1) offers connection-oriented semantics 2) does not introduce the overhead of resolving a service description in the critical path of network communication, 3) does not require a service description to be carried with every network message, and 4) does not require any changes to the network routing architecture.

The rest of the paper is organized as follows. Section 2 identifies the design goals for SoNS and Section 3 describes the architecture of SoNS. Section 4 describes the operation of the SoNS constraint parser, section 5 describes the SoNS resource discovery framework, section 6 describes the architecture of the module used to evaluate resources and section 7 presents the support for network connection migration. In section 8 we describe the API exported by a service-oriented network socket and present a representative context-aware application built using SoNS. Section 9 describes the implementation of SoNS for a mobile handheld device, and section 10 presents performance analysis and evaluation. Section 11 describes related work and, finally, in section 12 we conclude the paper and outline future directions of our research.

2 Design Goals

In order to identify the goals for a system designed to provide opportunistic access to services in a dynamically changing system, we consider a simple example application of such a system.

In our example, a video-stream played by a user's handheld device is automatically redirected to the nearest display as she moves in an environment populated with displays, possibly from different vendors and conforming to different standards. In order to provide this *follow-me-video* functionality, the application requires opportunistic access to the nearest display of a decent size, located in the same subnet as the user. Furthermore, though the application requires access to a better display as soon as one becomes available, it would not like the video-stream to be switched between displays due to transient fluctuations in their access latency or when a display device is quickly carried past it by another user. Finally, the application must be notified before a session is migrated to a new resource so that, for instance, it can transfer some application-specific state to the new resource to resume access to the service or to even decline the rebinding suggestion all together.

In order to support such applications, our system must meet the following goals.

- **Resource Discovery and Selection**: The system must be able to discover resources based on a high-level service specification. Additionally, the system must define a planning framework capable of evaluating and comparing the properties of available alternatives in order to find the closest match to application requirements.
- **Expressiveness:** An application must be able to state its requirements such that they can be used for both discovering and, subsequently, comparing the suitability of available alternatives. An application must be able to state the attributes required in a suitable resource, the range of acceptable values for each attribute, the preferred values for an attribute and the relative importance of each attribute to the application.
- **Extensibility:** In order to support a diverse set of applications in a variety of network characteristics and standards, the system must not enforce any fixed policies that could limit the use or efficacy of the system. Instead, the system must define an architecture that may be extended to handle different application requirements, network characteristics and standards.
- **Connection Rebinding Semantics:** It must be possible for an application to configure the semantics of rebinding a network session when a better alternative becomes available. Based on our target applications, we identify the following parameters to provide an application with the flexibility to configure the semantics of session rebinding.
    - **Context** It must be possible for an application to configure the context within which it wants to find the best resource for its requirements e.g. current subnet, current room.
    - **Agility**: It must be possible for an application to configure the agility with which it wants the system to react to *valid* changes in its context.
    - **Hysteresis**: It must be possible for an application to configure the hysteresis of the system, indicating how long the system should wait before reacting to a change, in order to avoid reacting to transient fluctuations that are not of interest to an application, and to protect against *thrashing*.
    - **Application-notification:** It must be possible for an application to register a call-back method, which is invoked by the system to notify the application about the availability of a better alternative. This notification can be used by the application to prepare for the rebinding of the network session. It must also be possible for the application to decline the suggestion of rebinding the session to the new resource.
- **Performance:** Where the system must include a planning function capable of evaluating and comparing a set of resources competing against application requirements, this planning task must be fast enough to quickly respond to changes in the system. Furthermore, as our system is interposed at the operating system socket level, it must be comparable in performance with the traditional socket-based communication. Finally, it must not introduce an overhead for applications that do not require service-oriented communication.

2.1 Service-oriented Network Sockets

Our service-oriented network session layer includes an attribute-based discovery framework for discovering resources in the system, as well as an *evaluator* module for computing the suitability of available alternatives against application requirements.

Since a network socket provides a portal between an application and the network communication support of an operating system, it presents a natural interface for incorporating application-level policies for establishing a service-oriented network connection by discovering and evaluating the available alternatives.

Service-oriented Network Sockets offer an additional socket domain that takes a high-level service specification as the destination name, instead of a network address, and defines additional socket options to configure the rebinding semantics for the service-oriented session. Using this interface, applications configure a network socket with an appropriate context, agility and hysteresis, and *connect* the socket by

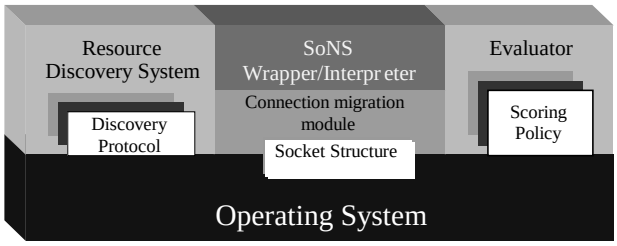

Figure 1: SoNS System Architecture

providing a service description to open a service-oriented network session. Using these application-level semantics, SoNS locates the most appropriate resource in the given context and establishes a network connection. If any subsequent changes in the system render another resource more suitable for application requirements, in accordance with the agility and hysteresis semantics of the application, SoNS notifies the application and migrates the session to the better alternative.

A service description is expressed as a set of constraints on the properties of an acceptable resource. As opposed to the resource discovery systems that find a resource by performing an exact pattern-match on its attribute-value pairs [3][4], the use of a constraint language in SoNS, for stating an evaluation criteria, offers the flexibility to evaluate and compare the alternatives available in a given context in order to find the closest match to the requirements of an application.

The design of SoNS handles the heterogeneity of discovery standards and application requirements by using a modular and extensible architecture for resource discovery and evaluation. Protocols for discovering resources and the policy for evaluating available choices can be tailored according to the application requirements and discovery standards used by different resources.

By handling the dynamism of the system at an end-node, SoNS does not require any changes to the network routing infrastructure. Therefore, as opposed to systems that employ application-level content-based routing [4] to address the dynamism of the system, SoNS architecture does not introduce extra routing complexity in the participating nodes, achieves better performance, and leverages the underlying network support for quality-of-service.

3 System Architecture

Figure 1 shows the architecture of a Service-oriented Network Sockets system. In order to facilitate application-specific extensibility, portability, accounting and fault-isolation, Service-oriented Network Sockets are implemented as a user-space wrapper around a traditional socket interface, instead of as a kernel module.

The SoNS architecture has four components: a resource discovery module, an evaluator module, a connection migration module, and a socket-wrapper module. Below we describe these modules in detail.

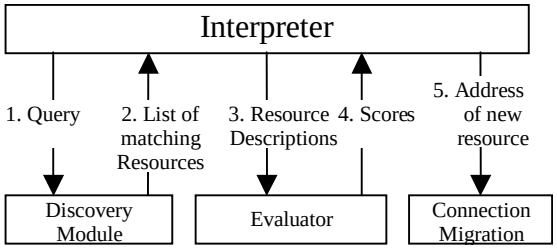

Figure 2: The SoNS Interpreter drives the different components in the system

3.1 SoNS Interpreter

The SoNS Interpreter, shown in figure 2, lies at the heart of the system and drives the different modules of the SoNS architecture; it parses the constraints specified by an application, discovers matching resources by invoking the resource discovery module, invokes the evaluator module to evaluate the suitability of any matching resources, and finally, in the case when a new resource becomes a better choice for the application, notifies the application and requests the connection migration module to migrate the connection to the new resource.

In order to allow this processing to be accounted on a per-connection basis, SoNS system forks a new Interpreter for every service-oriented network socket created by an application.

3.2 SoNS Interface

SoNS is designed as an extension of the operating system socket interface; it implements all the methods and options of a traditional AF_INET Unix socket, with additional options for establishing service-oriented network connections.

A service-oriented network socket extends a traditional network socket in the following ways:

1) The call to create an operating system socket accepts an additional domain, *AF_SONS*, for creating a service-oriented network socket. AF_SONS extends an AF_INET socket and allows an application to choose between (sock_stream) and UDP (sock_datagram) as the transport protocol for a service-oriented session, including support for the various options associated with these transport protocols e.g. TCP_NO_DELAY for TCP.
2) The *connect* method of a service-oriented network socket takes a high-level *service description*, instead of a network address, to establish a service-oriented network session. The service description is expressed in a simple constraint language, described in detail later in the section.
3) A service-oriented network socket can be configured with four additional options (as arguments to *setsockopt), context, agility, hysteresis and*

```
ConstSpec = Nested | Cmplx | Smpl
Nested = Cmplx (Cmplx+)
Cmplx = (Logical (Smpl  Smpl+))
Smpl =  (Relation Attribute)  |
    (Relation Attribute Range | Value) |
    (Relation Attribute Range | Value) Weight
Logical = AND | OR
Relation = < | > | =
Attribute = String
Weight = Integer
Range = Numeric Numeric
Value = String | Numeric
String = [a-z]+[a-z1-9]*
Numeric = Integer | Float
Integer = [1-9]+
Float = [1-9]+.[1-9]*
```

Figure 3: Constraints Language for SoNS

application-callback, to tailor the session rebinding semantics according to application requirements.

4) Finally, when configured with the optional application-callback, a service-oriented network socket invokes a callback method to notify (and seek permission of) the application before rebinding a network connection to a better alternative.

3.3 SoNS Constraint Language.

Though previous resource discovery systems offer varying degrees of sophistication for looking-up resources based on their attributes [9][4], these systems do not offer support for evaluating and comparing the suitability of matching resources against application requirements. SoNS, on the other hand, allows applications to specify the criteria for discovering, evaluating and comparing the available alternatives as a set of constraints expressed in a simple constraint language.

Though several sophisticated constraint languages have been proposed in other problem domains [10], the constraint language used to express a service-requirement in the SoNS system achieves a delicate balance between the expressiveness required for evaluating the suitability of available service-providers and the simplicity of design necessitated by the paucity of resources available in a mobile device.

The grammar for the SoNS constraint language is shown in figure 3. An expression in the SoNS constraint language lists the attributes that must be present in the selected resource, along with a range of acceptable values for each attribute. In order to define an evaluation and comparison criterion, a constraint also includes an operator, (less-than "**<",** or greater-than "**>**"), to indicate the preferred extreme in the range of acceptable values; resources with attribute values closer to the preferred extreme are favored over the resources with values further away towards the other extreme. This approach of allowing an application to express its requirement as a range of acceptable values, instead of a single scalar value, has the following merits. 1) It provides the flexibility to satisfy the requirements of an application with imperfect resources in its environment 2) It provides the system with a yardstick to compare and evaluate the matching resources against application requirements. 3) It encourages an application to explicitly declare its *scale of tolerance* for an attribute value; a change k in a range L ↔ K is more significant than the same amount of change k in a larger range, L ↔ (K +P).

```
(and (= device display)
     (> (size 15 30)
     (= color yes)
     (or (> video-streams 1)
          (= load 0)))
```

Figure 4: An example constraint specification expressed in the SoNS constraint language

In the case where an application is interested in the *least* or the *greatest* value for an attribute, irrespective of the specific value of the attribute, the application can leave the range unspecified. This could be used by an application to, for example, connect to the least loaded server in its environment, expressed as "< load".

SoNS also allows open ended ranges in the case where the application is interested in having an attribute value to be greater than (or lower than) a certain threshold, but perceives no marginal gain as the value of the attribute moves further away from the specified threshold. SoNS handles this case by treating the unbounded end of a range as 0 or a large positive integer, depending on which side of the range is unspecified.

Not all attributes of a resource required by an application are of the same importance to the application. SoNS handles this requirement by allowing an application to specify the relative importance of the listed attributes by attaching a (integer) weight with every attribute; an attribute with a weight of 4 is twice as important to an application as an attribute with a weight of 2.

Attributes that are allowed to have only a single value, including the attributes with textual values, use an equality (**"="**) operator and do not specify a range or attach a weight to the attribute; a resource description that does not match an equality constraint is simply rejected. Attributes that must be present in a matching resource, but whose value is not of interest to the application, are specified as a don't care value, stated as ANY.

Finally, the constraint language includes two logical operators, *conjunction* and *disjunction*, to allow individual constraint-expressions to be combined into a *composite constraint specification*. A composite constraint specification can have a hierarchical structure; constraints can be grouped (associated) and nested using braces, and the logical operators are distributed over nested constraints when evaluating a constraint.

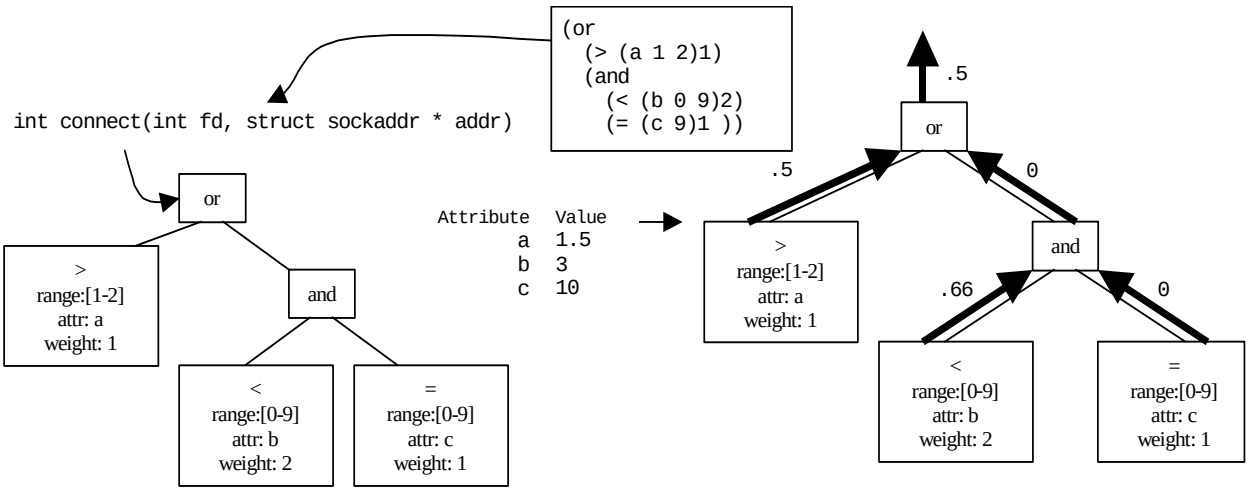

Figure 5: An illustration of constraint parsing and evaluation by the SoNS Interpreter

To illustrate the expressiveness of the SoNS constraint language, we show how the requirements of a follow-me-video application, presented in section 2, will be expressed in our language. Such an application can impose the following constraints on the display used by it. 1) The display must be more than 15 inches in size, for clear viewing, but less than 30 inches, due to the resolution limitations of the video-encoding scheme, 2) it must be capable of rendering colors, 3) and should be either capable of displaying more than one video-stream simultaneously or must not be in use. These requirements would be expressed in the SoNS constraint language as shown in figure 4. It is worth noting that the use of an open-ended range for the number of video-streams supported by the display device implies that the application is indifferent to the number of streams being displayed on the screen. If the application prefers to use a less cluttered screen, it will provide a closed range, and will use the "<"operator to indicate that a display capable of showing fewer streams is preferable. Therefore, the use of a range to express a constraint, in fact, encourages an application to be more precise in defining the, often assumed, precincts of context-awareness.

3.4 Semantics of Session Rebinding

Besides the constraints specified by an application to define the criteria for comparing available resources against application requirements, SoNS also allows an application to tailor the semantics of rebinding the network session by controlling the parameters for detecting and reacting to changes in the system. A service-oriented network socket takes four additional options as arguments to the *setsockopt* library call.

Context: An application can specify its *context* as a sub-net address, location of the looked-up resources, number of network hops traversed by a discovery message or any other metric meaningful for the discovery protocols part of the SoNS architecture. For example, the current implementation adjusts the *SCOPE* of an SLP [9] network query to limit the context of the discovery.

Agility: An application can specify the *agility* with which it reacts to changes in the system by adjusting the frequency to probe the system for changes. The agility is specified as the interval between successive probes, stated in seconds.

Hysteresis: An application can keep the system from reacting to transient changes, not of interest to the application, by specifying a value for *hysteresis*. The hysteresis is stated in terms of the number of probes for which an application requires the properties of the resources in its context to be consistent before SoNS (notifies an application and) switches the connection to a better alternative.

Application-Callback: Finally, an application can register a callback with the socket, which, if registered, is used to notify the application when a better alternative becomes available. This notification, parameterized with the description (including the network address) of the new resource, can be used by an application to prepare itself to switchover to the new resource or to reject the change by returning a false value from the callback. It is worth noting that since a connection migration can only happen when the application returns control from the call-back, the application can use the call-back to delay the migration to a "migration-safe" point in its control flow.

4 Constraint Parsing

The constraints specified by an application are used both for *discovering* and *evaluating* resources in the context of an application. To accomplish this, the constraints are parsed into a tree data-structure, which serves as an in-core representation of the application requirements for discovering and evaluating resource descriptions.

Constraints are read as a plain-text string from the sockaddr_sons structure passed by the application in a connect()socket call (refer to figure 5). The string is then parsed using a standard GNU Flex/Bison lexer/parser into a constraint tree. The parser makes a distinction between composite constraints and simple constraints. Simple constraint, specifying a range over a single attribute, are placed at the leaves of tree, while composite constraints, containing nested constraints composed by taking disjunctions (OR) and conjunctions (AND) of simpler constraints, are represented at the intermediate nodes of the tree (refer to figure 5).

The parser also fills-in any missing bounds, 0 for less than constraints and a large integer for greater than constraints, as well as missing weights with a default of 1.

5 Resource Discovery

After constructing a constraint tree, the SoNS interpreter invokes the discovery module with the list of attributes at the leaves of the constraint tree. The discovery module invokes the discovery protocols registered with it and returns the matching resource descriptions to the interpreter.

The interpreter then passes this list to the evaluator module, which assigns each resource a score by comparing the values of its attributes against the constraints stored in the constraint tree. The evaluator invalidates the resource descriptions with attribute values outside the range specified by the application, as well as the resources that fail to meet an equality constraint.

After the initial setup, this procedure is repeated every time the probe period specified by the application expires. An application can also force a probe/evaluate cycle, for instance on the command of a user. After receiving the score for each resource, the interpreter removes all the resource descriptions that were rejected and forms the "n-best-list" for the probe. If the application forced the probe (by invoking *connect* on an already connected socket), then the resource with the highest score is chosen from the n-best-list and the socket is migrated to its network address (just like the initial setup). However, if the probe was a normal periodic probe, the system enters the *hysteresis* phase. In the hysteresis phase the n-best-list from one probe/evaluate cycle is compared to the n-best-list stored from the previous cycle and the resources present in both new and old probes have their hysteresis value increased by one. Resource(s) with a hysteresis value greater than the hysteresis value specified by the application are separated and the connection is migrated to the network address of the resource with the highest score. In the case where an application has registered a call-back, SoNS invokes the callback method, with the description of the chosen resource, before performing the migration, and migrates only if the application-callback returns a true value (indicating application's approval of the connection migration). Upon migration of the network connection, the n-best-list is reset and the process is started anew.

5.1 SoNS Resource Discovery Framework

Our target network environment often comprises of resources conforming to different resource discovery protocols, e.g. IETF SLP [9], INS [4] and SSDP [11], due to both commercial and technical reasons. Therefore, a service discovery framework based on just a single discovery protocol is not sufficient to discover the various resources found in a pervasive mobile system.

SoNS handles this heterogeneity by defining an extensible resource discovery framework, capable of employing different discovery protocols to discover resources in the system. A discovery protocol is added to SoNS by registering a pointer to its *look-up* method, while SoNS performs resource discovery by invoking the look-up methods of all the discovery protocols registered with it.

However, various discovery protocols found in our target environment offer different degrees of expressiveness for looking-up resources in the system. Protocols like INS [4] and SSDP [11] simply take a list of attributes and match them with the attributes of the resources being looked-up, whereas more sophisticated protocols like SLP [9] and SSDS [3] can perform complex queries containing conjunctions and disjunctions on nested lists of attributes, as well as range comparisons for attributes with numerical values. In order to interoperate with such diverse protocols, SoNS translates a service specification to a very basic query format common to all discovery protocols.

SoNS resource discovery framework invokes a constituent discovery protocol with a simple list of ASCII-encoded attribute names, constructed by taking the attribute names from the leaves of the constraint tree created by the SoNS parser. Upon invocation, a discovery protocol finds the resources containing the specified attributes, and returns their descriptions in a list of *feature-sets*: sets of attribute-value pairs. The matching resource descriptions, encoded as feature-sets, are passed on to the evaluator module to evaluate their suitability against the constraints specified by an application.

It is worth noting that, in order to achieve compatibility with simpler protocols, this scheme does not require any filtering involving value comparisons to be performed by a discovery protocol. Rather, discovery protocols look-up resources by simply performing a pattern match on the specified attributes, and the suitability of a resource, based on the values of the looked-up attributes, is computed in the SoNS evaluator module.

Passing a query as a simple ASCII-encoded list of required attributes also has the virtue that it can be easily converted to a more ornate format, by a simple wrapper around the lookup interface, if required by a more sophisticated discovery protocol.

5.2 Context of Discovery

Along with a pointer to a look-up method, a discovery protocol can also register a pointer to a method for setting the scope of the network queries generated by the discovery protocol. This method is invoked by SoNS when an application specifies a context of interest as an option to a service-oriented network socket. For example, SLP and SSDS register a pointer to a method that sets the value of SCOPE of the discovery agent to configure the context of the network queries. Though some simpler protocols, e.g. SSDP, lack support for scoped queries, and hence, do not register this method, we believe that such support is the key to the scalability of a pervasive discovery protocol and will soon find its way in mainstream discovery protocols.

5.3 Probing vs. Advertising

A mobile device wishing to discover resources in its environment can either *passively* listen to advertisements by other resources in the system or can *actively* probe the network with periodic discovery messages.

SoNS uses active probing as it makes it simpler to support application-level semantics for session-rebinding. Applications configure the session rebinding semantics by setting 1) the frequency of probing, to adjust the agility with which resources are discovered, 2) the scope of a probe message, to adjust the discovery context and 3) the number of probes for which the properties of a resource must be consistent, to set the hysteresis of the system.

We favor probing over advertisements because in an advertisement based system the scope and frequency of the messages generated by a resource to advertise itself to the system cannot be adjusted to suit the requirements of any single application. Furthermore, with resource advertisements arriving asynchronously at different frequencies from various resources, there is no clean way to specify the hysteresis of the system.

From a design point of view, in an advertisement-based system, where resources are required to continuously advertise themselves to the system in the hope that some application might be interested, introduces a continuous overhead of network messages and processing of advertisement messages even when there is no application listening to the advertisements.

Finally, probing is supported by all the resource discovery protocols found in our target environment (though some protocols can also be configured to operate in an advertisement-based mode).

5.4 Directory-based versus Peer-to-peer Discovery

Resources can either respond to queries directly, in a peer-to-peer setup, or could register their descriptions with a directory service which could be searched to locate resources.

SoNS' extensible design does not impose a restriction on which of the two methods is employed by a constituent discovery protocol to discover resources in the system. However, we believe that a peer-to-peer model is more suitable for supporting application-specific session rebinding.

Though a directory-based setup avoids query broadcasts, and, hence, presents a more scalable design, it suffers from the following limitations in a dynamically changing system. 1) A directory-based architecture depends on the availability of host(s) in the system that are capable and willing to answer queries on behalf of other resources. 2) A directory-based scheme introduces the overhead of keeping the directory state consistent with the (oft-changing) properties of resources in the system. 3) The directory service can itself cause a bottleneck in the system. Since in a peer-to-peer setup resources themselves report their, up-to-date, properties, the rate of probing provides an accurate mapping for the rate of adaptation expected by the application; this can only be guaranteed in a directory based system when the directory service is always consistent with the changes in resource properties.

6 SoNS Evaluator Module

The discovery framework returns all the matching resource descriptions returned by the various discovery protocols to the interpeter, which passes these resource descriptions to the evaluator module. The SoNS evaluator module performs the planning required to select the resource, among the available alternatives, that comes closest to satisfying the service requirements of an application.

To motivate the evaluation strategy used by the SoNS evaluator module, consider a situation where the follow-me-video application mentioned above moves into an environment with two displays: one closer to the handheld device but the other larger in size and with better resolution. In this situation, there is no clear winner (that is better than all the other available alternatives in every aspect). A naïve solution could be to count the number of attributes for which a resource "beats" other alternatives and pick the resource with the maximum number of "wins". However, such a solution not only leads to a combinatorial explosion but also requires every sample of attribute values to be kept for later comparisons according to application's hysteresis requirements.

SoNS evaluator is designed to be simple and responsive to changes and does not require the attribute values of every resource to be preserved across multiple probes. SoNS achieves this by using a simple scoring scheme which sums-up the suitability of a resource in a single scalar value for efficient comparisons.

SoNS evaluator takes a list of feature-sets, along with the application's constraint tree, and returns a corresponding list of positive integer scores reflecting the suitability of each resource. A resource with an attribute that fails to meet an equality constraint or has a numerical value outside the range specified by an application is assigned a score of zero.

Figure 5 shows the operation of the default SoNS evaluator. SoNS' default evaluator performs a depth-first search of the constraint tree. On reaching a simple constraint at a leaf node, it extracts the value of the corresponding attribute from the resource's feature-set and compares the value with the range specified in the constraint. If the value satisfies the constraint, then a score between 0 and 1 is calculated based on where the value falls in the valid range. If the constraint specifies that smaller is better, then a value equal to the lower bound is assigned score 1 and a value equal to the upper bound is assigned score 0, with all other values being assigned linearly within that range. The reverse occurs for constraints indicating that larger values are better. If the constraint specifies only equality or the ANY keyword, then any value fitting the constraint is given a score of 1. Finally, the score is multiplied by the constraint's weight and returned as the value of that leaf node.

After assigning scores to the leaf nodes, scores for the intermediate nodes, containing conjunctions and disjunctions, are calculated using the following algorithm. An OR node acquires the score of a child node with the highest score in its sub-tree, while a score of zero is assigned if all of its children nodes have a score of zero. An AND node is evaluated in a complimentary way: the score of an AND node is calculated by adding the scores assigned to its child nodes, while any child node with a score of zero causes the AND node to be assigned a score of zero. The overall score of a resource is the score calculated for the root of the constraint tree using the attribute values in the resource's feature-set.

We have found this simple evaluation strategy to be sufficient for our purposes for the following two reasons. 1) It keeps the design of SoNS simple enough to be hosted in resource constrained mobile devices and 2) the simplicity of the algorithm used for evaluating and comparing the available alternatives incurs minimal penalty in terms of the responsiveness of the system; where a more elaborate scheme could be used for comparing the suitability of available alternatives, it would increase the time spent in evaluating a resource, resulting in an increased latency between the time a viable resource become available and when the system recognizes its superiority.

As described earlier, the scores returned at each probe are compared by the Interpreter according to the hysteresis semantics of the application and a winner is chosen if a resource consistently scores better than other resources.

The extensible design of SoNS also allows the default evaluation policy to be replaced by more efficient or specialized algorithms better suited to individual application requirements. An application can replace the default evaluation policy by registering a pointer to an application-specific evaluator with the SoNS evaluation module. This allows more involved constraint satisfaction engines, for example as proposed in [12], to be employed for calculating the relative utility of available resources. Such planning and constraint satisfaction systems are a topic of our current research.

7 Connection Migration Module

Once a better resource has been selected, the SoNS Interpreter requests the connection migration module to migrate the network connection to the new resource.

The semantics of migrating the network connection from one resource to another depend on both the stateful-ness of the service being accessed and the reliability guarantees offered by the underlying message transport protocol [13]. Migration of an unreliable network connection to a stateless service is accomplished by simply closing the old network connection and opening a fresh connection to the new resource. However, additional support is required for migrating reliable connections and for managing stateful services [13]. Migration of a reliable connection requires support for preserving the sequence of messages across migration, while a connection to a stateful service can be migrated transparently across resources only when the state accessed at the old resource is also available at the new resource in the form that the access to the service can be resumed at the new host from where it left-off at the old resource.

The former requires a reliable transport protocol with support for migrating an active connection, while the later also requires a system for distributing and maintaining consistent state across replicated instances of a stateful service.

This paper focuses on enabling a client to utilize the best provider of a service in its changing context; the subject of replicating and synchronizing stateful services has been extensively researched by others [14] and is not covered in this paper.

SoNS uses the Migrate system [15] for migrating network connections between resources. We chose the Migrate system as it provides support for securely migrating both reliable and unreliable network connections, as well as a lightweight, soft-state based

consistency management system to support connection migration across stateful servers. Unlike other connection migration systems, like SCTP[16], that require the network addresses of all the potential servers to be known at connection setup time, Migrate allows a connection to be migrated to a newly available server using the TCP migrate options. Having said this, the modular design of SoNS allows other connection-migration systems to be used as well, though we have not integrated other such systems with SoNS as yet.

8 Applications

This section describes the API of SoNS and a simple, yet representative, application we have developed to demonstrate the utility of service-oriented network connections offered by the SoNS architecture.

Our test applications were developed for a Compaq iPAQ, fitted with a backPaQ and running familiar Linux. Our backPAQ is fitted with an 802.11b wireless card, video-camera, accelerometer and the Cricket Location detection system [17].

8.1 Follow-me-video

We have used SoNS to develop a *follow-me-video* application. A follow-me-video application running in a handheld device carried by a user re-directs the video stream to the display closest to the user as she moves in the system. In our test environment, all resources (server devices) are also fixed with Cricket Beacons to measure their distance relative to other Cricket-enabled devices (including our handheld device).

The relevant code snippet from our example application, mentioned in section 3.3, is shown in figure 6. Our example application generates an MPEG-1 encoded stream and is interested in the nearest display with 1) Resolution: 640x800 – 1280x1600 (with preference for displays with higher resolution), 2) Size: larger than 15 inches to allow viewing from a distance, but less than 30 inches due to the limitation of the encoding resolution (with preference for a larger display)

The application creates an AF_SONS domain socket, and specifies the following options: Context: 6th-Floor, Agility: 5 (seconds between probes), Hysteresis: 3 (number of probes), Call-back: pointer to method that forced a base frame to be transmitted. The application then connects the socket by giving it a composite constraint specification for the properties of the display device.

SoNS sets the SCOPE of the SLP user agent to 6th-Floor, probes the network for display devices, and connects to a display that scores the highest points among those present on the 6th floor. The distance to a display device is measured by invoking the Cricket location system (added to the system just like another discovery protocol). After making the initial connection, the network is probed, using SLP and

```
int main(int argc, char ** argv) {
  int sockfd;
  int intopt;
  char * charopt;
  size_t opt_sz;
  sons_callback_t cbsons;
  struct sockaddr_sons sasons;
  sockfd = socket(AF_SONS, SOCK_STREAM, 0);
  intopt = 5;
  opt_sz = sizeof(int);
  setsockopt(sockfd, SOL_SOCKET,
          SO_PROBE_PERIOD, &intopt, opt_sz);
  intopt = 3;
  opt_sz = sizeof(int);
  setsockopt(sockfd, SOL_SOCKET,
          SO_HYSTERESIS, &intopt, opt_sz);
  charopt = "floor-6";
  opt_sz = strlen(charopt) + 1;
  setsockopt(sockfd, SOL_SOCKET,
          SO_SERVICE_SCOPE, charopt, opt_sz);
  cbsons = force_b_frame();
  opt_sz = sizeof(sons_callback_t);
  setsockopt(sockfd, SOL_SOCKET,
          SO_CALLBACK, &cbsons, opt_sz);
  charopt =
          "(and (= service display)\n"
              "(= media mpeg1) \n"
              "(> xresolution 800 1600) \n"
              "(> yresolution 640 1280)) \n"
              "(> displaysize 15 30) \n"
              "(< distance))";
  sasons.sin_family = AF_SONS;
  memcpy(sasons.query, charopt,
        strlen(charopt));
  connect(sockfd, (struct sockaddr *) &sasons,
         sizeof(struct sockaddr_sons));
  //...read and write using standard socket
  // calls...
  close(sockfd);
  return 0;
}
```

Figure 6: Code snippet from the follow-me-video application developed using SoNS.

Cricket, every 5 sec and available resources are compared according to the hysteresis. If a better display becomes available, SoNS invokes the application callback, which forces a base MPEG frame to be transmitted upon migration, so that the playing of video at the new display can be resumed without jitter.

Our example application also monitors the accelerometer embedded in the backPAQ to find out if the device is moving and with what speed. If the application discovers that the handheld device is mobile, the application can increase the rate of probing and reduce the hystersis value, according to the degree of movement reported by the accelerometer, in order to take advantage of the displays that become available for a short time when, for example, a user walks down a hallway.

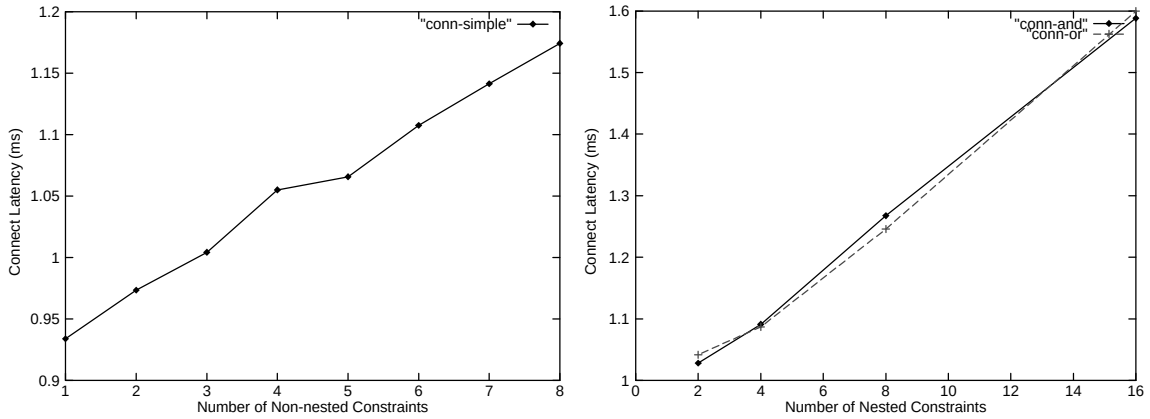

Figure 7: The cost of a connect() call (averaged over 100 tests) increases only linearly with the number of constraints.

9 Implementation

SoNS was implemented in GNU C/C++ on GNU/Linux. The code is divided into four modules: the socket interface library, SoNS interpreter, wrappers for the resource discovery protocols, and the evaluator module.

9.1 Socket Library

The socket library code overrides the socket interface to offer the extended SoNS interface, and spawns an interpreter daemon to periodically discover and evaluate available resources.

In order to provide an extended interface, we either needed to modify the underlying Linux libc or use a package that captures system calls and redirects them through other functions. We chose the latter route and specifically chose to use TESLA [18]. TESLA allows arbitrary handlers to be inserted between an application's socket call and the underlying socket kernel calls, precisely matching our needs. Furthermore, the Migrate architecture, which we use for connection migration, also uses TESLA, so the overhead of using TESLA would be present in our system anyway.

We wrote TESLA handlers to override the calls to socket(), connect(), getsockopt(), setsockopt(), and close() functions. The most interesting overridden call is connect(). The connect call exercises the entire system since it sends a message to the interpreter daemon telling it to force a network probe, then picks the best service, and finally calls connect on the underlying socket structure. getsockopt() and setsockopt() simply update socket-related data in the daemon. The interpreter daemon itself merely sits in a loop waiting either for an event from the TESLA handler or an alarm signal indicating that it should perform a periodic poll/evaluate cycle.

9.2 Resource Discovery Protocols

The interpreter invokes the *run_query* method of all the discovery protocols registered with it. The run query method takes an array of required attribute names and returns a list of attribute-value bindings. This method is the sole interface between the interpreter and the discovery protocols so that discovery protocols can be easily added/replaced.

The current implementation employs two discovery protocols to find resources in the system: IETF Service Location Protocol [9] and the Cricket Location System [17] (for estimating distance to available resources). We use the OpenSLP implementation of SLP, with the User Agent configured to perform discovery in a peer-to-peer fashion, by multicasting the query on the SLP multicast channel.

10 Performance Analysis and Evaluation

Unlike content-based routing systems, the session-oriented approach of SoNS moves the cost of resolving service descriptions from the critical path of a network message delivery to the stage of establishing and, subsequently, rebinding a network session. Therefore, we evaluate the performance of SoNS by measuring 1) how quickly it can setup a service-oriented network session and 2) how quickly it can rebind the network session when a better alternative becomes available.

All tests were performed on a Pentium III with 256MB of RAM running Linux 2.4. Since we wanted to isolate our system from network latency, we used an in-memory stub SLP rather than the OpenSLP SLP. We expect most constraint specifications to have between 1 and 15 elements and be a combination of both simple constraint specifications and composite constraint specifications. Our tests span this space: we vary the number of attributes in a straight-line set of simple constraint specifications and also vary the height of the tree of composite constraint specifications.

10.1 Session-setup Latency

The SoNS system adds latency in two places, at a socket() call where we fork a daemon and initialize all the discovery protocols installed in the system, and on a connect() call where we must decide which device is the best device available.

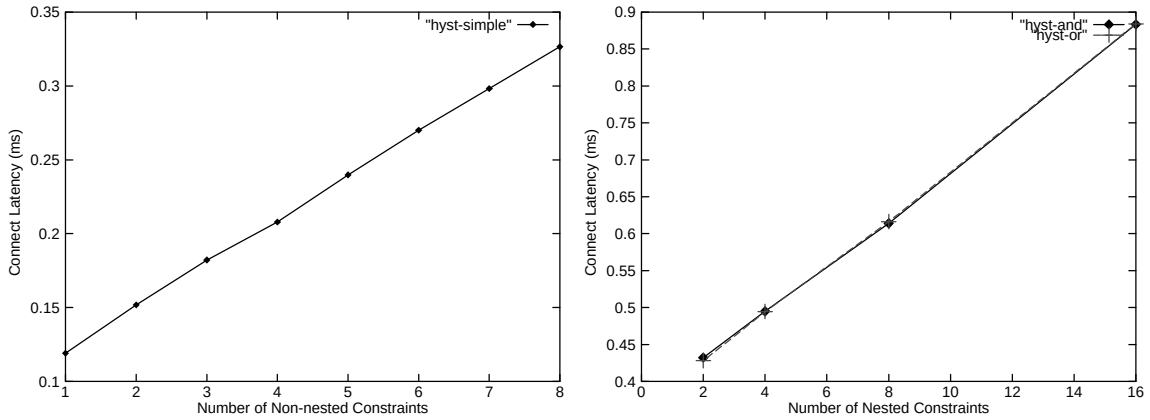

Figure 8: Latency (averaged over 100 tests) between the time a better alternative becomes available and the time SoNS realizes its presence (and signals the application about its presence).

In all of our tests, the *socket* call took between 2ms and 5ms. This is fairly high, but represents the cost of forking a process and all inter-process communication between the daemon and the SoNS TESLA handler.

Figures 7 summarizes the cost of a connect call as we varied the number of constraints. The cost of the connect call increases only linearly as the number of constraints, both nested and non-nested, are increased. The connect latencies in our system hover around 1ms, which, though higher than expected, is acceptable when amortized over the life of the connection.

10.2 Session-rebinding Latency

Another form of latency shows itself as the time between the discovery of a resource and signaling the application that the SoNS system has found a new best resource. Ideally, this should be zero, but we must evaluate the services returned, which has a non-zero cost. Figure 8 show the time elapsed from when a call to run_query() method returns — with matching resources on the network— and when SoNS invokes the application-callback to notify the presence of a better alternative (given the hysteresis semantics). This latency again increases only linearly with the number of constraints (both for nested and non-nested constraints), and more importantly, hovers only around 200-500µs of latency.

In order to achieve compatibility with simpler protocols, SoNS does not require any filtering involving value comparisons to be performed by a discovery protocol. Instead, discovery protocols return all those resources that contain the required attributes and the evaluator module performs the value comparisons to compute the suitability of matching resources. However, since all the value comparisons in this scheme are performed by the evaluator module, the evaluator must be able to efficiently compare a moderately large number of matching resource descriptions. Figure 9 shows the time spent in the evaluator module as we increased the number of resource descriptions processed by the evaluator. This cost increases only linearly and hovered only between 0.8 – 1.0 msec in our tests.

The simplicity of our system makes it suitable for mobile handheld device. The memory footprint of our system varied between 0.8 MB to 1 MB during our experiments.

10.3 Evaluation

Where our system achieves acceptable performance, we found that the following implementation choices incurred unwanted overhead:

- Per-socket interpreter daemon *processes*,
- Use of standard IPC between the socket wrapper and the per process interpreter deamon, and
- fixed-point arithmetic.

Our implementation would be faster if we were able to communicate with the daemon without copying through interprocess communication channels. This could be accomplished by using a thread library at the cost of making our code less portable. Secondly, we maintain a separate daemon process for each socket to allow for fine-grained accounting. However, this is inefficient compared to an implementation in which a single deamon process handles the discover/evaluate cycles for all the sockets, since such an implementation would save the cost of spawning a daemon every time a socket is created, and might allow for optimizations by batching queries by different applications.

Finally, our system is slowed down by the use of the fixed-point math system we wrote for computing resource scores. This is because the iPAQs we include in our target platforms do not have floating point units and incur an order of magnitude performance hit on floating point performance. Since scoring and weighting is an inherently floating-point process, we were forced to write our own, non-optimized, implementation of fixed-point arithmetic. We are currently working on a more efficient implementation in the light of these observations.

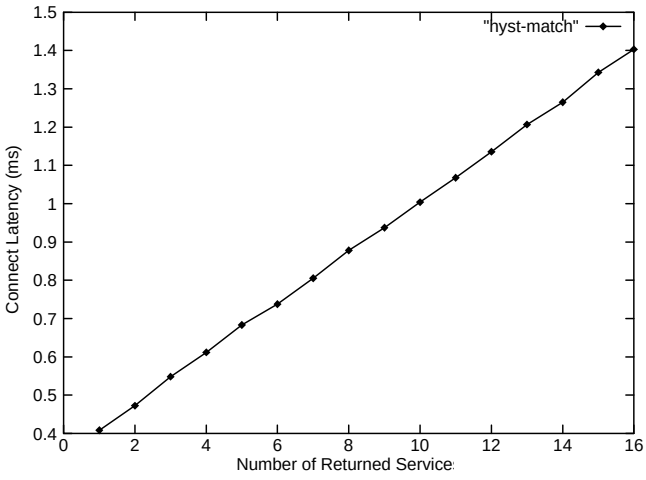

Figure 9: The hysteresis latency increases linearly with the number of services returned by the SoNS discovery framework.

11 Related Work

SoNS integrates a service-oriented abstraction with a traditional operating system communication interface. Using SoNS, applications open a network connection with an abstract service specification, instead of a network address, and the system automatically connects the application to the most suitable server in its changing environment. SoNS combines resource discovery and evaluation with a connection migration system to provide application-specific opportunistic access to service providers. SoNS' modular architecture can be extended with various resource discovery, location-detection, and connection migration protocols, and its evaluation policy can be customized to individual application preferences.

Therefore, SoNS is designed to complement and leverage recent research in resource discovery, location-detection and connection migration protocols, not to replace such systems. In fact, SoNS was motivated by the desire to combine pervasive mobile computing technologies developed for MIT's Project Oxygen, e.g. INS, Migrate and Cricket, to leverage context-aware applications in a mobile handheld device.

The recent interest in pervasive computing environments has given rise to a proliferation of systems that allow resources to be dynamically discovered based on their attributes. As opposed to the white-pages style lookup offered by systems like DNS that simply resolve a resource name to its network address, such systems do not require a priori knowledge of some unique identifier of the resource, like its network address, and hence can be used to dynamically discover and utilize resources as they become available in a pervasive system.

Such attribute-based resource discovery systems differ in the format used by them to describe resource properties, expressiveness offered by their look-up interfaces, whether they offer push-based or pull-based discovery and whether queries are mediated by a directory service or resolved in a peer-to-peer fashion in the system. In addition to the classical examples like Grapevine, GNS [19] and X.500 [20], a range of industrial standards like Microsoft's UPnP resource discovery protocol (SSDP) [11], IBM's T-Spaces, and IETF's Service Location Protocol [9], and experimental systems like MIT's INS [4] and Berkeley's SSDS [3] have emerged over the last few years. For example, where SLP offers a rich LDAP-based [20] query interface, systems like INS and SSDP define simple attribute-based resolvers that can be hosted in resource constrained mobile devices.

SoNS is designed such that different discovery protocols can be added to its resource discovery module, possibly via a simple wrapper function to covert the SoNS attribute list to the specific format used by a discovery protocol, e.g. XML (used by Berkeley's SSDS).

Unlike existing resource discovery protocols that simply match queries against resource descriptions, SoNS uses an applications-specific evaluation framework that continuously monitors, evaluates and compares the available alternatives in order to pick the closest match to application requirements. Indeed, the problem of satisfying high-level requirements with imperfect resources has been extensively researched in the AI domain [12]. However, where systems like MetaGlue [12] propose to use general-purpose constraint satisfaction engines over complex utility functions, SoNS default evaluator is designed to be simple and responsive to changes in the system.

Content-based routing systems like INS's late binding architecture [4] and Information Bus [21], as well as application-level anycast routing systems like [8], allow applications to send messages without specifying the network address of the recipient, and route the messages to the appropriate server by looking at the content of each network message and matching that with the properties of the available servers. Where such systems offer an alternative to our approach, they inherently lack application-level session semantics, do not offer a clean interface for configuring the application-specific policy for resource comparison and session-rebinding, introduce the overhead of resolving service descriptions within the critical path of every message delivery, and, by defining their own routing framework, do not leverage the support for QoS offered by the underlying network.

12 Conclusions and Future Work

This paper establishes the need for *service-oriented* network connection, and presents the design and implementation of the SoNS system. SoNS presents an application with an extended socket interface to open a

service-oriented network connection by providing a high-level service-specification. When asked to establish a connection, SoNS discovers the available resources and connects the application to the resource that best provides that service in its context. Once connected, SoNS continuously monitors, compares, and evaluates available alternatives, and reconnects the application to a better alternative if one becomes available.

As opposed to content-routing systems, SoNS moves the cost of discovering and evaluating resources against application requirements at the connection set-up time, and allows the application to exercise control at the level of a network session. Since the cost of discovery and selection in SoNS is amortized over the life of the network session, it allows SoNS to be significantly more sophisticated in terms of expressiveness, evaluation and selection of available resources, as compared to systems that perform message-level service-selection-and-routing.

As SoNS integrates support for context-awareness with a traditional operating system communication interface, we have found it much simpler to use than other systems that require the use of additional, and often several different [7], APIs to build a context-aware application. Though we believe that SoNS has the potential to become an integral part of future operating systems in a pervasive computing environment, it relies on the wide-spread deployment of network devices embedded with service advertisement protocols, as well as the availability of location detecting mechanisms to estimate the distance of a user with the devices embedded in her context.

The design of SoNS pays special attention to extensibility in order to take advantage of the wide range of emerging technologies for resource discovery, location detection and network connection migration.

The current design of SoNS does not include a security framework. Security in such a system is required at several levels: to protect resources against illegitimate access, to protect the SoNS system against malicious extensions, and to protect the connection migration system against connection hijacking. Though some discovery protocols, like SSDS and SLP, and connection migration schemes, like Migrate, define their own security models, we are currently investigating an extensible security framework that would allow security policies to be defined independently of the constituent modules.

SoNS makes it simpler to develop context-aware applications. Our experience with SoNS has shown us that unlike message-based routing systems that are better suited to *command-based applications* e.g. “sending a document to the nearest printer”, SoNS is equally useful for *connection-oriented applications* as well, e.g. follow-me-video/audio. We are currently developing more applications to demonstrate the utility of SoNS in pervasive mobile environments.

References

1. Christopher K. Hess et al. *Building Applications for Ubiquitous Computing Environments*, International Conference on Pervasive Computing (Pervasive 2002), pp. 16-29, Zurich, Switzerland, August 26-28, 2002.
2. Esler, M. et al. G. *Next Century Challenges: Data-Centric Networking for Invisible Computing: The Portolano Project at the University of Washington* Mobicom 99
3. S. Czerwinski et al. *An architecture for a secure service discovery service.* In Proc. of MobiCom-99, pages 24-35, N.Y., August 1999
4. William Adjie-Winoto et al. *The design and implementation of an intentional naming system*, Proc. 17th ACM SOSP, Kiawah Island, SC, Dec. 1999.
5. David Garlan et al. *Project Aura: Towards Distraction-Free Pervasive Computing* IEEE Pervasive Computing, special issue on "Integrated Pervasive Computing Environments", Volume 1, Number 2, April-June 2002, pages 22-31.
6. C. Perkins et al *A Mobile Networking System Based on Internet Protocol*, IEEE Personal Communications, Vol. 1, No. 1, pp. 32-41, March 1994.
7. Harter, A. et al.: *The anatomy of a context-aware application.*, Mobile Computing and Networking. (1999) 59-68
8. S. Bhattacharjee et al. *Application Layer Anycasting.* In Proc. IEEE INFOCOM'97, 1997
9. Erik Guttman. *Service Location Protocol: Automatic Discovery of IP Network Services*. IEEE Internet Computing Journal, 3(4), 1999.
10. J. Jaffar et al. *Constraint logic programming: A survey*. The Journal of Logic Programming, 19/20:503--582, May/July 1994.
11. *Universal Plug and Play*, http://www.upnp.org
12. Krzysztof Gajos. *Rascal - a Resource Manager for Multi Agent Systems in Smart spaces*. In Proceedings of CEEMAS 2001.
13. Alex C. Snoeren et al. *Fine-Grained Failover Using Connection Migration*, Proc. 3rd USENIX USITS, March 2001.
14. R. Golding. *A Weak-Consistency Architecture for Distributed Information Services*. Computing Systems, 5(4):379--405, 1992.
15. Alex C. Snoeren et al. *An End-to-End Approach to Host Mobility*, Proc. 6th ACM MobiCom, August 2000
16. R. R. Stewart, et al. *Stream Control Transmission Protocol*. RFC 2960, IETF, Oct. 2000.
17. Nissanka B. Priyantha et al. *The Cricket Compass for Context-Aware Mobile Applications* Proc. ACM MOBICOM Conf., Rome, Italy, July 2001.
18. Jon Salz, SM thesis, MIT. 2002, *The Transparent Extensible Session-Layer Architecture for End-to-End Network Services*.
19. A. Birrell et al. *Grapevine: An exercise in distributed computing*. Comm. Of the ACM, 25(4):260–274, April 1982.
20. *CCITT. The Directory—Overview of Concepts, Models and Services*, December 1988. X.500 series recommendations, Geneva, Switzerland.
21. B. Oki et al. The Information Bus (R) – *An Architecture for Extensible Distributed Systems.* In *Proc. ACM SOSP*, pages 58–78, 1993.
22. MIT Project Oxygen, http://www.oxygen.lcs.mit.edu

Acknowledgements

We would like to thank Steve Ward for his valuable comments that helped us improve the paper significantly. The title of the paper was also suggested by Steve Ward.

We would like to thank Marvin Theimer, our “shepherd”, and anonymous reviewers for their helpful comments and suggestions.