Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 75 (2005) 193–204
The Journal of Systems and Software
www.elsevier.com/locate/jss

Enhancing Service Location Protocol for efficiency, scalability and advanced discovery

Weibin Zhao *, Henning Schulzrinne

Department of Computer Science, Columbia University, 1214 Amsterdam Avenue, Mailcode 0401, New York, NY 10027, USA

Received 18 March 2004; received in revised form 21 March 2004; accepted 3 April 2004
Available online 7 June 2004

Abstract

This paper presents three new mechanisms for the Service Location Protocol (SLP): mesh enhancement, preference filters and global attributes. The mesh enhancement simplifies Service Agent (SA) registrations and improves consistency among Directory Agents (DAs) by defining an interaction scheme for DAs and supporting automatic registration distribution among peer DAs. Preference filters facilitate processing of search results (e.g., finding the best match) in SLP servers (DAs and SAs) to reduce the amount of data transferred to the client for saving network bandwidth. Global attributes allow using a single query to search services across multiple types. These mechanisms can improve SLP efficiency and scalability and support advanced discovery such as discovering multi-access-point services and multi-function devices. We expect that these techniques can also be applied to other service discovery systems.

Keywords: Service discovery; Service Location Protocol; Peer relationship; Full mesh; Registration distribution; Preference filters; Global attributes

1. Introduction

As computing continues moving towards a network-centric model, finding and making use of available services such as printing, display and file sharing in the network becomes increasingly important. To use a service, a device such as a computer or personal digital assistant needs to know the access point [1] of the service, which traditionally depends on a priori knowledge or manual configuration. As more devices are network enabled and more services are available on networks, properly configuring devices for better utilizing available services involves non-trivial administrative overhead. Moreover, administrative configuration becomes difficult or even impossible when devices move from a fixed managed network to a constantly changing or unmanaged network. Consider the following three application scenarios. First, as a mobile device often relies on services provided by other devices, it needs to discover available services when it moves to a new network. Secondly, in an ad-hoc network (e.g., a disaster rescue setting), administrative configuration is unlikely to be possible and effective since devices need to learn about each other dynamically and cooperate. Lastly, for home networks, low cost and ease of use are dominant design considerations, making administrative configuration unsuitable. In recognizing the need to reduce administrative configuration as much as possible and enable automated service discovery, many companies, standards bodies and consortia are actively developing service discovery technology. As a result, various service discovery systems and protocols are emerging in recent years, such as the Service Location Protocol (SLP) (Guttman et al., 1999b), Jini (Waldo, 1999), Universal Plug and Play (UPnP, 2004), Rendezvous (Apple, 2004), Salutation (Salutation, 2004), Universal Description Discovery and Integration (UDDI, 2004), and the Bluetooth Service Discovery Protocol (SDP) (Bluetooth,

* Corresponding author. Tel.: +1-2127490908; fax: +1-2126660140.
E-mail addresses: zwb@cs.columbia.edu (W. Zhao), hgs@cs.columbia.edu (H. Schulzrinne).

[1] In IP networks, a service access point is specified by a tuple: IP address, port number and access protocol (such as FTP or HTTP), which can often be encoded into a URL.

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2004.04.011

2004). Although different systems address the service discovery problem at various levels in various ways, they all support the same basic functionality, namely mapping service descriptions or specifications to service access points. By using service discovery technology, a device no longer needs to know its service access points via a priori knowledge, instead it can just specify the characteristics of its desired services, which will be automatically mapped into available service access points in any network that supports service discovery.

One of the widely used service discovery protocols is SLP, which is an IETF (Internet Engineering Task Force) proposed standard for service discovery in IP networks. As more applications (Kempf and Montenegro, 2001; Naugle et al., 2001; Bakke et al., 2003; Zhao and Schulzrinne, 2004c; Poynor, 2001) employ SLP for various discovery purposes, we saw a need to improve SLP efficiency and scalability, and support new discovery scenarios such as discovering multi-access-point services and multi-function devices. In this paper, we present three new mechanisms for SLP: mesh enhancement, preference filters and global attributes. The mesh enhancement simplifies Service Agent (SA) registrations and improves consistency among Directory Agents (DAs) by defining an interaction scheme for DAs and supporting automatic registration distribution among peer DAs. Preference filters facilitate processing of search results (e.g., finding the best match) in SLP servers (DAs and SAs) to reduce the amount of data transferred to the client for saving network bandwidth. Global attributes allow using a single query to search services across multiple types.

The rest of this paper is organized as follows. We first give some background for service discovery and SLP in Section 2. Then we describe three proposed SLP mechanisms: mesh enhancement, preference filters and global attributes in Section 3–5, respectively. Finally, we discuss our implementation in Section 6, give experimental results and their evaluation in Section 7, list related work in Section 8, and conclude in Section 9.

2. Background

2.1. Service discovery

In a service discovery system, a common service description framework is needed for service providers, referred to as servers, and service users, referred to as clients, to describe service characteristics so that they can understand each other properly. In general, each service can be described using a set of attribute-value pairs, with each attribute-value pair specifying one property of the service. There are two ways to organize attributes: a flat structure where all attributes are at the same level, and a hierarchical structure where attributes can be at different levels. For examples, SLP simply puts attribute-value pairs into a list, whereas UPnP and UDDI use XML to describe a hierarchy of attributes. Although Resource Description Framework (RDF, 2004) has been proposed as the service description format for interoperability between service discovery systems (Reynolds, 2001), so far there is no service description standard yet.

While service advertisements from servers usually include all attributes of services, service search requests from clients only include attributes of interest, which may just specify a desired service type or class such as printer, or give additional desired service properties such as color printer and printing speed. In general, any service search request can be specified by a search filter such as those used in SLP and Lightweight Directory Access Protocol (LDAP) (Howes, 1997), which is a logical expression about attributes of interest and their desired values. The matching of a service search request with a service advertisement leads to a discovery. Although service discovery systems bear similarity to web search engines as they both provide matches for search requests, they differ in that service discovery uses attribute-based matching whereas web search uses keyword-based matching, and thus a match in the former has a specific meaning while a match in the latter may have different meanings in different contexts.

Multicast and directories are two widely used mechanisms for service discovery. When only multicast is used, services are discovered in a peer-to-peer fashion in two ways. In passive discovery, servers periodically multicast their service advertisements, and clients listen to these advertisements. Clients compare received service advertisements with their desired service requirements to determine matching services. In active discovery, clients multicast their service search requests, and servers listen to these requests. A server compares received service search requests with its service advertisement; if there is a match, the server unicasts its service advertisement to the client. Although multicast can enable a device to be fully auto-configured (IETF, 2004) in a network segment, it usually cannot scale to a large number of devices, and it is not generally supported in wide-area networks. In the directory-centric service discovery model, directory services accept service advertisements from servers, and answer service search requests from clients. Servers register their services with directories, and clients search services at directories, all using unicast. In order to discover directory services, multicast can be used for an intranet, such as in SLP and Jini, but well-known directories are often assumed for the Internet, such as in UDDI. In addition, Dynamic Host Configuration Protocol (DHCP) (Droms, 1997) can be used to get directory service information in local-area networks (Perkins and Guttman, 1999), and DNS SRV (Gulbrandsen et al., 2000) can be used to obtain

directory service information for given DNS domains (Zhao et al., 2004).

2.2. Service Location Protocol

The Service Location Protocol (SLP) (Guttman et al., 1999b) provides a flexible framework for service discovery in IP networks. It supports both directory-centric and peer-to-peer discovery models, and enables powerful service filtering and browsing. SLP uses general Universal Resource Locators (URLs) (Berners-Lee et al., 1998) or the "service:" URL scheme (Guttman et al., 1999a) to specify *service locations* or service access points. Each service has a *service type*. For examples, service:printer:lpr://mandolin.cs.columbia.edu is a printing service, and ftp://ftp.cs.columbia.edu is an FTP service. Service characteristics are described using a list of attribute-value pairs, such as "speed = 15 ppm, resolution = 1200 dpi" for a printing service. SLP uses *service scopes* to arrange services into groups, which can indicate geographic locations (e.g., "New York"), administrative groupings (e.g., "Law School"), or other categories (e.g., "Emergency"). Each service registration is valid only for its specified *service lifetime* (e.g., 2 h), and will be removed from directory services when it has expired. In other words, SLP service registrations are soft states, and need to be refreshed.

SLP has three types of entities: User Agents (UAs), Service Agents (SAs), and Directory Agents (DAs). Fig. 1 illustrates their relationships. UAs initiate service discovery on behalf of clients by querying all SAs via multicast or a DA, if available, via unicast. UAs use three types of SLP messages: a service type request (`SrvTypeRqst`) message to get a list of available service types in a service type reply (`SrvTypeRply`) message, an attribute request (`AttrRqst`) message to get a list of attributes for a given service type or service instance in an attribute reply (`AttrRply`) message, and a service request (`SrvRqst`) message with a search filter (or attribute predicate) specifying characteristics of the desired service to get a list of URLs giving the locations of matching services in a service reply (`SrvRply`) message. `SrvTypeRqst`, `SrvTypeRply`, `AttrRqst` and `AttrRply` messages allow a client to browse available service types and their attributes, which can be used to construct service queries in `SrvRqst` messages. Given the desired service type, and a set of attributes describing the service, SLP derives the service access points (URLs) for clients. SAs work on behalf of services by responding directly to UA queries, and registering with DAs, if they exist, via service registration (`SrvReg`) messages. SAs can also deregister services from DAs using service deregistration (`SrvDeReg`) messages. DAs serve as centralized information repositories by accepting SA registrations and answering UA queries. DAs can be discovered in two ways. For passive DA discovery, UAs and SAs simply listen for unsolicited DA advertisement (`DAAdvert`) messages sent periodically by DAs to an administratively scoped multicast address (Meyer, 1998). UAs and SAs can actively discover DAs by multicasting a DA discovery `SrvRqst` message whose service type is "service:directory-agent". DAs answer each DA discovery request with a unicast `DAAdvert` message.

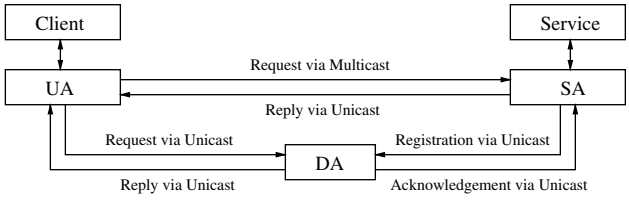

Fig. 1. SLP system architecture.

SLP achieves scalability by using DAs and service scopes, and thus efficiently supports service discovery in systems of different scales. In small SLP deployments, DAs are usually not needed. UAs multicast requests to all SAs, and SAs respond via unicast. Since this multicast-based discovery cannot scale to a large number of SAs and UAs, DAs are introduced in mid-size SLP deployments, where SAs register services with DAs, and UAs search services at DAs, all using unicast. In large SLP deployments, DAs are arranged into different scopes to provide further scalability, e.g., services in the Law School and Business School of Columbia University can be assigned to different scopes.

3. Mesh enhancement

3.1. Motivation

DAs allow SLP to scale to large deployments that may span large geographic regions. To avoid a single point of failure, each scope needs to have multiple DAs. However, SLP DAs do not interact with each other, thus SAs are required to register services with all DAs in their scopes. This simple approach has two disadvantages. First, it places too heavy a burden on SAs since they not only need to discover and register with all existing DAs, but also need to re-register when new DAs are discovered or old DAs are found to have rebooted. In other words, an SA needs to constantly monitor all DAs in its scope. This burden becomes an issue of efficiency and scalability when many devices provide services and each of them uses an SA. Secondly, in large deployments it is hard to guarantee that all SAs can discover all DAs in their scopes, leading DAs in the same scope to have inconsistent registrations. To remedy this situation, we designed the SLP mesh enhancement (mSLP). The rationale behind mSLP is that distributing registrations to multiple DAs should be

taken care of by DAs instead of by SAs because there are far fewer DAs than SAs. It is more efficient and scalable for SLP to have a smaller number of powerful DAs but many lightweight SAs.

3.2. Design overview

mSLP defines a scope-based fully meshed peering DA architecture. For each scope, all DAs that serve the scope form a fully meshed peer relationship, similar to Internal Border Gateway Protocol (IBGP) (Rekhter and Li, 1995). In mSLP, DAs that share one or multiple scopes are peers. Each pair of peer DAs maintain a single peering connection [2] between them no matter how many scopes they share. Fig. 2 shows an example of this peering DA architecture for four DAs and three scopes. To keep consistent registrations for their shared scopes, two peer DAs exchange new registrations via their peering connection by using direct forwarding and anti-entropy—these two mechanisms will be described further in Section 3.4.

mSLP employs a full-mesh topology mainly for simplicity and reliability. We anticipate that each scope has a small number of DAs, thus mSLP should be sufficient for a mesh size on the order of tens or below. Moreover, large DA meshes can be avoided by splitting scopes. For example, if scope S has n DAs and n is too large, we can split S into two finer scopes S_1 and S_2, with n_1 DAs for S_1 only, n_2 DAs for S_2 only, and n_3 DAs for both S_1 and S_2, and $n_1 + n_2 + n_3 = n$. In this way, instead of having a large full mesh of size n, now we have two smaller full meshes of size $n_1 + n_3$ and $n_2 + n_3$, respectively. Accordingly, a service registration that previously targets for S now needs to be registered under both S_1 and S_2.

Another important mSLP design consideration is to be fully backward compatible with SLP. mSLP strives to be a lightweight enhancement to SLP by only defining a new `DAAdvert` attribute—"mesh-enhanced", a new message extension—mesh forwarding (`MeshFwd`), and a new message type—anti-entropy request (`AntiEtrpRqst`). An SLP DA can be mesh-enhanced by carrying the "mesh-enhanced" attribute keyword in its `DAAdvert` message and supporting the mSLP functionalities without affecting its old functionalities. Mesh-enhanced DAs can be deployed incrementally and coexist with legacy SLP DAs in the same system.

mSLP offers a number of advantages. First, SA registrations can be simplified because no matter how many DAs are present in a scope, an SA only needs to discover, monitor and register with any one of them for that scope. Registrations will then be propagated automatically to other DAs. Secondly, consistency among peer DAs can be improved as they periodically reconcile their inconsistent registrations. Furthermore, newly booted and rebooted DAs can catch up on all new registrations at once from their peers purely through DA interaction, without involving any SAs. Finally, fewer TCP connections are needed when SAs register with DAs via TCP. [3] Consider a scope that has n SAs and m DAs. In SLP, each SA needs to connect to each DA and register, thus $n * m$ TCP connections are needed. But in mSLP, each SA only needs to connect to one DA in the full mesh of m nodes and register, then registrations are propagated through the DA mesh, therefore only $n + m * (m-1)/2$ TCP connections are needed. Given any $n > m \geqslant 2$, we have $n * m > n + m * (m-1)/2$. For example, if $n = 100$ and $m = 10$, then 1000 TCP connections are needed in SLP, but only 145 such connections are needed in mSLP.

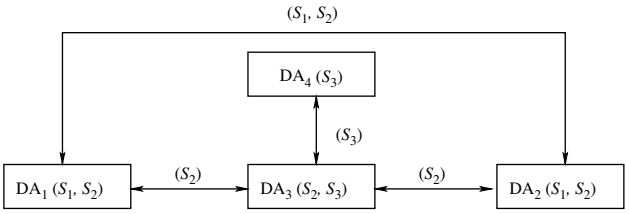

Fig. 2. An example of the mSLP scope-based fully meshed peering DA architecture for four DAs (DA_1 to DA_4) and three scopes (S_1 to S_3). An edge between two DAs means that they are peers.

3.3. Peer relationship management

In mSLP, a DA maintains a peer relationship to each of its peers. A DA can learn about its peers via configuration, DHCP (Perkins and Guttman, 1999), and `DAAdvert` multicast and unicast. A peer relationship has three stages: setting up, maintaining, and tearing down. To begin a peer relationship, one DA (say, DA_1) must first get the `DAAdvert` of another DA (say, DA_2), then DA_1 initiates a peering connection to DA_2, and sends its `DAAdvert` along this peering connection to ensure that DA_2 will have its `DAAdvert` and be able to identify this peering connection. Thus, only a single peering connection will be established between DA_1 and DA_2. Also in the setting up stage, two peer DAs exchange information about their existing peers, which enables both DAs to learn about new peers incrementally. After two DAs has set up a peer relationship, they start to propagate new registrations and periodically send a keep-alive message to each other. This keep-alive mechanism enables a DA to detect network partitions

[2] A peering connection is a persistent connection (e.g., TCP) that provides reliable and ordered transfers between two peers.

[3] Service registrations in SLP can be performed via either TCP or UDP, but a registration needs to use TCP if it is too large to fit into a UDP packet.

and peer crashes. A DA will tear down a peer relationship when its peer's keep-alive message has timed out, when it has received its peer's shutdown message, or when its peer has closed the peering connection.

3.4. Registration propagation control

mSLP propagates registrations among peer DAs under two constraints. First, two peer DAs exchange registrations only in their shared scopes. For example, as DA_1 and DA_3 in Fig. 2 share one scope S_2, they exchange registrations only in S_2. Note that a multi-scoped registration needs to be propagated properly in all targeting scopes. For example, consider that DA_3 in Fig. 2 receives a multi-scoped registration in scopes S_2 and S_3, then this registration needs to be propagated to DA_1, DA_2 and DA_4 since all of them serve part of the targeting scopes. The second constraint is that only new registrations are exchanged between two peer DAs. To achieve this, each registration uses a `MeshFwd` extension to carry two pieces of control information: a *version-timestamp* assigned by its SA, and an *accept-id* assigned by its accepting DA. The accepting DA for a registration is the first DA that accepts the registration. An accept-id has two components: accept-da (a unique identifier for the accepting DA) and accept-timestamp. All accept-timestamps assigned by the same DA must be monotonically increasing. Therefore, all accept-ids are unique; they define a total order for all registrations accepted by the same DA and a partial order for all registrations accepted by all DAs. mSLP uses accept-ids for registration propagation control to ensure that any registration accepted by any DA is distributed to all targeting DAs exactly once. Specifically, a DA propagates registrations in the increasing order of their accept-ids, i.e., registrations accepted by the same DA are propagated in the increasing order of their accept-timestamps, and registration accepted by different DAs may be propagated in any order. Similarly, all version-timestamps assigned by the same SA must be monotonically increasing. Version-timestamps are used to resolve different versions of the same registration: [4] a new version will overwrite an old one, but not vise versa. Since an SLP registration is updated only by one SA, using version-timestamps is sufficient to identify the most recent version for any registration.

mSLP distributes registrations among peer DAs in two ways: anti-entropy and direct forwarding. Anti-entropy is used for exchanging initial registrations when two peer DAs find out about each other for the first time, and for catching up on new registrations after failures. A DA initiates an anti-entropy session by sending an `AntiEtrpRqst` message to a peer. Then the peer replies with all requested new registrations in the increasing order of their accept-ids, and sends a service acknowledgment (`SrvAck`) message at the end of the batch of new registrations to indicate the processing of the corresponding `AntiEtrpRqst` message has been completed. While in anti-entropy, new registrations are pulled by the receiving DA via an `AntiEtrpRqst` message and are sent in a batch using a `SrvAck` message to signal the end of the batch, in direct forwarding, new registrations are pushed by the sending DA and are sent individually. More specifically, after a DA has sent all new registrations accepted by itself to a peer via anti-entropy, the DA starts to forward any further incoming registrations accepted by itself directly to the peer. This direct forwarding continues as long as the peer is alive and there is no failure. Note that the direct forwarding of a registration only goes one hop from its accepting DA to all targeting DAs.

mSLP supports two types of anti-entropy sessions: complete and selective. Complete anti-entropy (Petersen et al., 1997) is the traditional way to perform anti-entropy, in which all registrations that have an accept-id greater than any specified accept-id in the `AntiEtrpRqst` or have an accept-da not specified in the `AntiEtrpRqst` are solicited. Selective anti-entropy (Zhao and Schulzrinne, 2002) is our proposed new way to perform anti-entropy, in which only registrations that have an accept-id greater than any specified accept-id in the `AntiEtrpRqst` are solicited. Selective anti-entropy enables two DAs to perform partial anti-entropy in the granularity of one accept-da, i.e., all registrations accepted by the same DA. Selective anti-entropy is a generalization of traditional anti-entropy, and is more flexible to support scope-based replication in mSLP and support complex partial replication in general. Next, we use an example to show how selective anti-entropy differs from complete anti-entropy. Consider a scope that has three DAs: DA_1, DA_2 and DA_3. DA_2 has registrations accepted by DA_1, DA_2 and DA_3. If DA_1 sends a selective `AntiEtrpRqst` to DA_2 using an accept-id list as $\{(DA_2, T_2)\}$, then DA_1 only requests registrations that are accepted by DA_2 and have an accept-timestamp greater than T_2. If DA_1 sends a complete `AntiEtrpRqst` to DA_2 using the same accept-id list as before, then DA_1 requests all registrations accepted by DA_1 and DA_3, in addition to those registrations accepted by DA_2 and having an accept-timestamp greater than T_2.

4. Preference filters

4.1. Motivation

Because an SLP server, whether a DA or SA, does not perform any processing on search results, all

[4] Different versions of the same registration have the same service URL.

matching service entries are returned from the server to the client in no particular order. This works fine in small SLP deployments, but may not scale to large deployments. Consider the following scenarios. First, if too many services match a search request, the search results may exhaust client network and storage resources. Secondly, a client may just want to find a few services that satisfies its requirements rather than all of them. Sending unneeded results to the client will waste network and server resources. Finally, a client may want to weigh the relative suitability of matching services based on some criteria, which calls for sorting search results. Sorting at the server is more efficient than sorting at the client since the former does not need to pass the attributes of matching services to the client just for sorting purposes. Reducing the amount of data transferred to the client is useful when the client uses a low bandwidth channel, such as a wireless channel. A good example showing the need for processing search results at SLP servers is the best-match search, such as finding a printer that has the shortest queue. For this discovery, an SLP UA needs to get information for all printers, sort them based on the queue length attribute, and choose the one with the shortest queue. This procedure is inefficient when there are many printers.

4.2. Design

Preference filters are designed to facilitate flexible processing of search results, and are specified via SLP extensions attached to `SrvRqst` messages. Fig. 3 shows the processing of a `SrvRqst` message that has a search filter and a preference filter: first the search filter is applied to the service registration database generating a set of matching services, then the preference filter is applied to the matching services generating a set of preferred services.

Although the format of preference filters could be designed in a way similar to SLP and LDAP search filters (Howes, 1997), we employ a simpler approach based on composition. We choose *select* and *sort* as two basic preference filters and design the corresponding SLP `Select` and `Sort` extensions (Zhao et al., 2002) for specifying them, then we use these two basic filters to compose generic preference filters.

The `Select` extension is used by a UA in the `SrvRqst` message to limit the maximum number (say, n) of results to be returned, and is used by an SLP server in the corresponding `SrvRply` message to indicate the total number (say, m) of search results. If $n < m$, then only the first n search results are returned, otherwise all m search results are returned. As a special case, a UA may set n to 0 to obtain the number of search results without retrieving the results themselves.

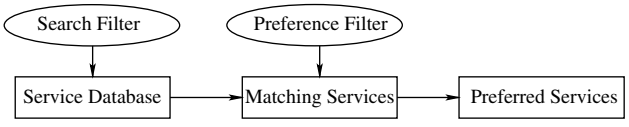

Fig. 3. The processing of a `SrvRqst` that has a search filter and a preference filter.

The `Sort` extension carries a sort key list. Each sort key has a key name (i.e., an attribute name), a type specifier ("s" for string and "i" for integer), an ordering specifier ("+" for increasing and "−" for decreasing), and an optional reference value. Although SLP has five attribute types (integer, string, boolean, opaque and keyword), we only consider integer sort and string sort since keyword attributes [5] never need to be sorted, and boolean and opaque attributes can be sorted as strings if needed. Integer keys may have a reference value, as in *speed:i:+:12*, causing the sort to be based on the distance to the reference value, 12.

A generic preference filter is a list of select and sort filters observing the following rules: (1) two basic filters of the same type, whether select or sort, cannot be adjacent to each other; (2) if the same number of sort and select filters are used, the last one must be a select filter; and (3) for two select filters s_1 and s_2, if s_1 appears earlier than s_2, then the selected number of results specified in s_1 must be greater than that in s_2.

Next we show some examples of preference filters, in which *select(n)* denotes a select filter and *sort(sort-key-list)* denotes a sort filter. Finding the best match is accomplished via a sort filter followed by a select filter, e.g., "*sort(load:i:+), select(1)*" for the least loaded service, "*sort(speed:i:−), select(1)*" for the fastest service, and "*sort(price:i:+:12), select(1)*" for the service with a price closest to 12 charging units. Other complex preference filterings include "*sort(speed:i:-), select(3)*" for the three fastest services, "*sort(speed:i:−,load:i:+), select(1)*" for the least loaded service among the fastest, and "*sort(speed:i:−), select(3), sort(load:i:+), select(1)*" for the least loaded service among the three fastest.

5. Global attributes

5.1. Motivation

A *local attribute* describes a service property specific to certain service type whereas a *global attribute* describes a service property common to all service types. Local attributes and global attributes differ in how they

[5] SLP keyword attributes have no values.

are defined, named and used. Currently, SLP only supports local attributes in that each service type defines its own attribute set via a service template (Guttman et al., 1999a); an attribute name is unique only within its service type (i.e., two different service types may use the same attribute name); and an attribute is always used along with its service type. As more service properties are identified as being common to all service types, such as transport protocol, we saw a need to introduce global attributes into SLP for efficiency and advanced discovery. Without such a mechanism, a UA needs three steps to find all services supporting Stream Control Transmission Protocol (SCTP) (Stewart et al., 2000): sending a `SrvTypeRqst` message to obtain a list of service types, then using a separate `SrvRqst` message to search services of each type, and finally combining the search results. As a `SrvRqst` message can only search services of a single type, $n + 1$ searches are needed for n service types, which is inefficient if n is large.

5.2. Basic design

To enable global attributes in SLP, we need to assign a separate namespace to global attributes, define them via attribute templates, and using them properly in searching services across multiple service types. Since a global attribute can be used with any service type, if it has the same name as a local attribute, then there will be a confusion on which is which. Therefore, a separate namespace is needed for global attributes. To follow the common practice of prefixing an attribute name with its service type, we use the "*service-*" prefix in global attribute naming. Note that XML (Bray et al., 2000) also uses prefixes to define its namespaces. We define a global attribute via an attribute template (Zhao and Schulzrinne, 2004b). Any service type that uses a global attribute imports the attribute's definition into its service template, similar to the C `include` and Java `import` mechanisms. In this way, a global attribute only has one definition, and can be used consistently for all service types. A global attribute can appear in any place where a local attribute is appropriate. In a `SrvRqst` message, when local attributes are used, exactly one service type must be specified; but when only global attributes are used, multiple service types or a service type wildcard can be specified. Thus, using a single `SrvRqst` message can search services across multiple or all service types. For example, to find all services supporting SCTP, we can use a `SrvRqst` message that has a service type wildcard, and a search filter (or attribute predicate) of "*service-transport-protocol* = *sctp*".

Using global attributes can improve SLP efficiency. First, global attributes only need to be defined once. Afterwards, they can be imported into any service template. This avoids defining the same attribute repeatedly in different service templates, and ensures a consistent definition. Secondly, by using global attributes, a single `SrvRqst` message can search services across multiple service types, which is more efficient than using multiple `SrvRqst` messages, one for each service type.

5.3. Advanced discovery

Using global attributes can accelerate the standardization of common service properties and support advanced discovery scenarios. As good examples, we can define service identifier and device identifier as global attributes. Service identifiers and device identifiers are URIs (Berners-Lee et al., 1998), e.g., UUIDs (2004). Each of them uniquely and persistently identifies a service or a device.

While service identifiers are used as service keys in Jini (Waldo, 1999) and UDDI (2004), SLP uses service URLs as service keys. Since a service may change its URLs (e.g., when the service moves), retrieving a service based on its service URLs may not always be feasible. To remedy this situation, we can define service identifier as a global attribute so that a client can always find a service based on its service identifier whether the service has changed its URLs. Using service identifiers, we can also discover multi-access-point services that provide the same service at different access points residing at the same device. For example, a printer that supports IPP (Herriot et al., 2000) and LPR access protocols may have two URLs service:printer:ipp://mpp.example.com and service:printer:lpr://mpp.example.com. A multi-access-point service advertises each access point separately, but all advertisements use the same service identifier to indicate that they point to the same service. A client can discover all advertisements of a multi-access-point service by specifying the service identifier and the service type in a `SrvRqst` message.

Using device identifiers, we can discover multi-function devices that provide different types of services at the same device. For example, a device that supports printing and scanning services may have two URLs service:printer://print.example.com and service:scanner://scan.example.com. A multi-function device advertises each service type separately, but all advertisements use the same device identifier to indicate that they reside at the same device. A client can discover all advertisements of a multi-function device by specifying the device identifier and a wildcard service type (or all the service types the device supports) in a `SrvRqst` message.

Using service identifiers and device identifiers together, we can discover replicated services, namely the same service being provided at different devices. A replicated service advertises the same service at each device separately, and all advertisements use the same service identifier but different device identifiers. In contrast, all

advertisements of a multi-access-point service residing at a single device use the same service identifier and the same device identifier.

6. Implementation

We have implemented the mesh enhancement, preference filters and global attributes in our release of enhanced SLP; the source code can be found at (Zhao and Schulzrinne, 2004d). To support the mesh enhancement, DAs need to manage peer relationships by maintaining a peer table, and need to control registration propagations by maintaining a summary vector for all registrations as well as an accept-id and a version-timestamp for each registration. At the same time, SAs need to use the `MeshFwd` extension in their registrations, but UAs do not need to be changed. To support preference filters and global attributes, an SLP server only needs to slightly adjust its processing of `SrvRqst` messages. For a `SrvRqst` message with a preference filter, the filter is ignored during the search, and then the filter is applied to the search results. When the filter has multiple select and sort filters, they must be processed in order, with the output of one filter as the input of the next filter. The output of the last filter is returned to the client. For a `SrvRqst` message that uses local attributes, it should have exactly one service type, and is handled as before. For a `SrvRqst` message that uses only global attributes, it may have multiple service types or a service type wildcard. In this case, the service type information is ignored during the search, and then those search results that do not match any of the specified service types are discarded.

7. Evaluation

To evaluate the proposed enhancements, we carried experiments in our local-area network (LAN) and on PlanetLab (PlanetLab, 2004). In the LAN, we use a cluster of 30 machines; each has a 1 GHz Intel Pentium III CPU, and 512 MB of memory. They all run Redhat 9.0 with Linux kernel 2.4.20-20.9, and are connected via 100 Mb/s fast Ethernet. PlanetLab consists of more than 300 nodes all over the world; each has a CPU of at least 1 GHz clock rate, and has at least 1 GB of memory. They all run Redhat 9.0 with Linux kernel 2.4.22-r3_planetlab, and use PlanetLab software 2.0. PlanetLab nodes have four types of network connections: DSL lines, Internet2, North America commodity Internet, and outside North America.

We implemented SLP DAs with the proposed enhancements using Java 1_4_2_03, and implemented SLP SAs and UAs using C. We evaluate each enhancement individually by comparing the results when the enhancement is enabled with that when the enhancement is disabled.

7.1. Mesh enhancement

To show the benefits of using mSLP, we measure consistency between two peer DAs after one recovers from failures. In this experiment, we run two DAs, one SA, and one UA, all at local machines, and they are all in the same scope. The SA performs n different registrations repeatedly at a fixed interval of I_r seconds, and the lifetime of each registration is set to $I_r * (n+1)$ seconds. Thus, each registration is refreshed every $R = I_r * n$ seconds, and each DA should have n registration entries at steady state. The UA queries both DAs at a fixed interval of I_q seconds to find out how many registration entries each DA has. Since the UA does not need to retrieve the entries themselves, it attaches a preference filter "*select(0)*" to each query. The SA (or UA) times out a registration (or query) if it cannot get a response after T seconds, in which case the DA is assumed to be failed (crashed or separated from the network). The experimental parameters are set as follows: $n = 100$, $I_r = 6$ s, $R = I_r * n = 600$ s, $I_q = 10$ s, and $T = 1$ s.

In the first case, mSLP is disabled, and the SA registers with both DAs. In a duration of 2000 s, the DA at host *ankara* works properly, but the DA at host *ottawa* crashes in two time periods: $[90, 440]$ s and $[1150, 1320]$ s. Fig. 4 shows the number of registration entries at both DAs sampled by the UA. For each sample, if the response from a DA has timed out, then the UA marks the number of registration entries at that DA as -1. We observe that *ottawa* misses m service registrations when it fails during $[t_0, t_1]$, where $m = n * (t_1 - t_0)/R$ assuming $t_1 - t_0 \leqslant R$. When *ottawa* recovers from failures, it has m

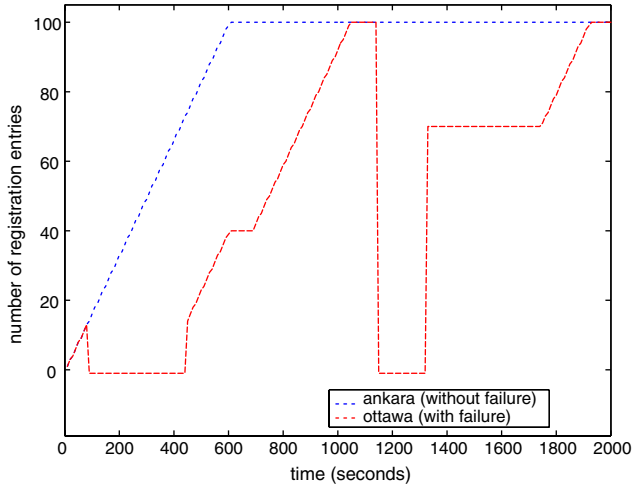

Fig. 4. Without using mSLP: consistency between two peer DAs at host *ankara* and host *ottawa* after *ottawa* recovers from two failures during $[90, 440]$ and $[1150, 1320]$ s.

registration entries less than *ankara* during $[t_1, t_0 + R]$, catches up missing entries during $[t_0 + R, t_1 + R]$, and has the same registration entries as *ankara* after $t_1 + R$ if it does not have new failures. Thus, *ottawa* needs to take a refresh interval of R seconds to catch up all missing registrations. When the UA queries *ottawa* during $[t_1, t_1 + R]$, it gets incomplete service information.

To quantify the relationship between missing entries and failure duration, we use μ to denote the average percentage of missing entries in the first R seconds after a DA recovers from failures, and use ρ to denote the ratio of failure duration over refresh interval R. We first consider $\rho \in [0, 1]$. In the above experimental setup, the missing entries in $[t_1, t_0 + R]$ is $n * \rho$, and the average missing entries in $[t_0 + R, t_1 + R]$ is $n * \rho/2$. Thus, we can compute μ as follows:

$$\mu = \frac{n * \rho * (1 - \rho) * R + n * \rho/2 * \rho * R}{R * n} * 100\% = \rho * (2 - \rho)/2 * 100\%$$

Fig. 5 shows the relationship between μ and ρ for $\rho \in [0, 1]$. We observe that μ increases as ρ increases, and $\mu = 0$ when $\rho = 0$, and $\mu = 50\%$ when $\rho = 1$. If $\rho > 1$, the DA will expire all registration entries when it recovers. Thus, $\mu = 50\%$ when $\rho > 1$, which is the same as $\rho = 1$.

In the second case, mSLP is enabled. For each registration, the SA only registers with one DA: it randomly chooses one DA to register; if the response from the chosen DA has timed out, then it registers with another DA. Fig. 6 shows the experimental results, where the failure scenario is the same as that in the first case. We observe that when both DAs are alive, each registration is propagated from one DA to another DA automatically via mSLP. Also, when *ottawa* recovers

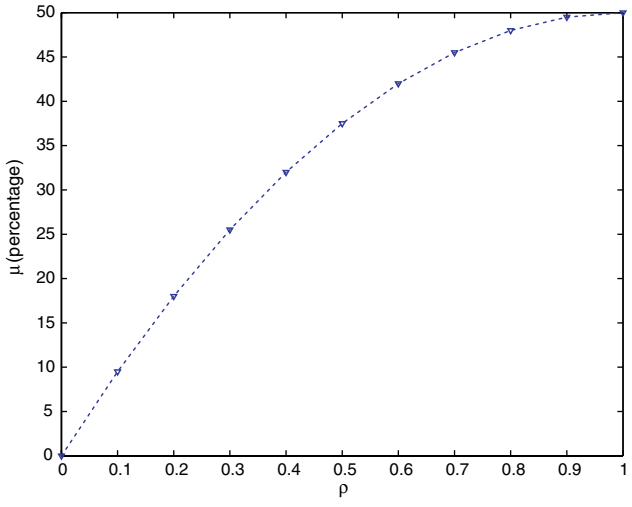

Fig. 5. The relationship between μ and ρ for $\rho \in [0, 1]$, where μ is the average percentage of missing entries in the first R seconds after a DA recovers from failures, and ρ is the ratio of failure duration over refresh interval R.

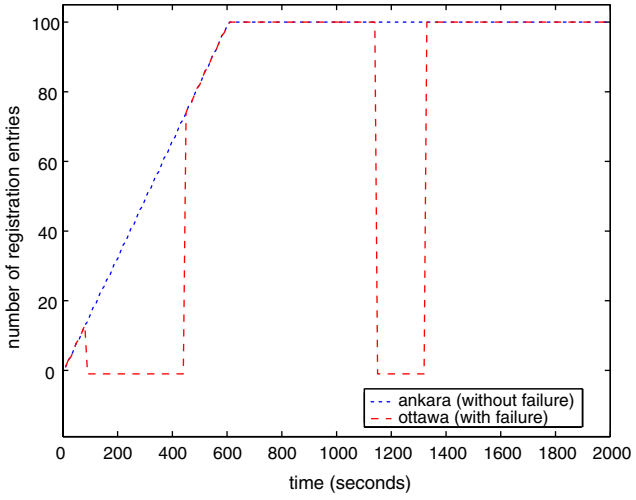

Fig. 6. Using mSLP: consistency between two peer DAs at host *ankara* and host *ottawa* after *ottawa* recovers from two failures during [90, 440] and [1150, 1320] s.

from failures, it gets all missing registrations at once from *ankara* via mSLP quickly, which takes about 100 ms in our experiments. Thus, when both DAs are alive, the UA can always get the same consistent reply whether it queries *ottawa* or *ankara*. In other words, mSLP can effectively fix the inconsistency problem among peer DAs after one of them recovers from failures.

To quantify the consistency improvement by using mSLP, consider the probability that a UA gets incomplete service information. Assuming each DA has an availability of p_1, and $\rho \leqslant 1$, then the probability that a DA has incomplete service information is $p_2 = (1 - p_1)/\rho$. Thus, the probability that a UA gets incomplete service information from two DAs is $p_3 = p_2 * 50\% + p_2 * 50\% + p_2^2$ when mSLP is not used, but is $p_4 = p_2^2$ when mSLP is used. For example, if $p_1 = 99.9\%$, and $\rho = 0.1$, then $p_2 = 1\%$, $p_3 = 1.01\%$, and $p_4 = 0.01\%$, where the probability that a UA gets incomplete service information has been reduced by two orders of magnitude by using mSLP.

7.2. Preference filters

To show the benefits of using preference filters, we measure the response time of a query when many entries match the query. As an example, consider the usage scenario in DotSlash (Zhao and Schulzrinne, 2004a), where different web servers register with mSLP DAs, and a web server discovers and utilizes spare capacity at other web servers to relieve its load spikes. Although many registered web servers may have spare capacities, a web server only needs to use a few of them in case of load spikes. In this experiment, we run one DA and one SA at local machines, and run two UAs at PlanetLab nodes, one at *gtidsl1* which is behind a DSL line, and another one at *ucla1* which connects to Internet2. The

SA registers with the DA to simulate service registrations performed by n web servers with different spare capacities, where n varies from 10 to 1000. The two UAs query the DA to obtain information about spare capacities at available web servers. For each n, each UA queries in two cases: (1) without using preference filters, all n entries are retrieved, and (2) by using the following preference filters "*sort(spare-capacity:i:−), select(10)*", only entries with the top 10 largest spare capacities are retrieved. We use TCP in the first case since the reply size can be quite large, but use UDP in the second case since the reply size is small and fixed.

Fig. 7 shows the experimental results. We observe that without using preference filters, the response time increases as the number of matching entries n increases since the DA needs to process more entries, and send more data to the UA. Further, when the reply size gets larger, the low bandwidth UA at *gtidsl1* takes significantly more time to get the reply than the high bandwidth UA at *ucla1*. In contrast, by using preference filters, the response time only increases slightly as n increases since the reply size is unchanged, but the DA needs a slightly more time to process a larger number of matching entries. The difference of response times is mainly due to the difference of reply sizes. When preference filters are used, the reply size if fixed, which is 927 bytes for all $n \in [10, 1000]$. But when preference filters are not used, the reply size increases as n increases, which is 920 bytes when $n = 10$, 9020 bytes when $n = 100$, and 90 020 bytes when $n = 1000$.

7.3. Global attributes

To show the benefits of using global attributes, we measure the time used for completing a location-based query. As an example, consider the problem of finding all services at a given location, where a number of different types of services exist, such as printer, projector, and fax. In this experiment, we run one DA and one SA at local machines, and run one UA at the PlanetLab node *gtidsl1*. The SA registers n types of services with the DA, where n varies from 1 to 20. The UA queries the DA to find services at a given location. Without using global attributes, the UA needs to use $n + 1$ queries, where one query for obtaining the service type list, and a separate query for each service type. Using global attributes, the UA only needs to use one query by specifying a service type wildcard.

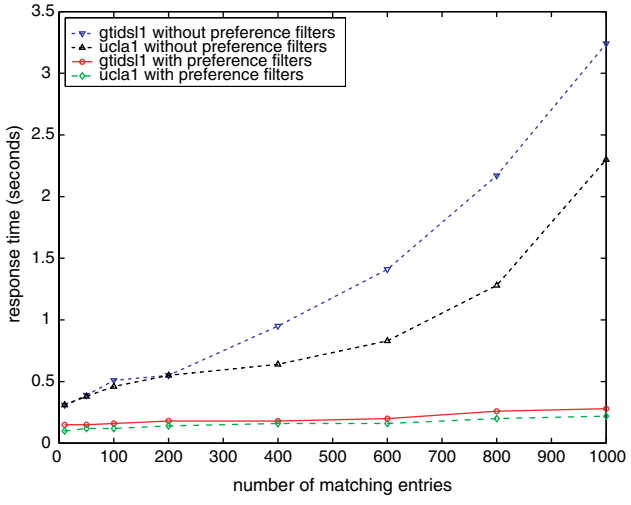

Fig. 7. Using preference filters versus without using preference filters: the response time measured from two PlanetLab nodes, *gtidsl1* and *ucla1*, where *gtidsl1* is behind a DSL line, and *ucla1* connects to Internet2.

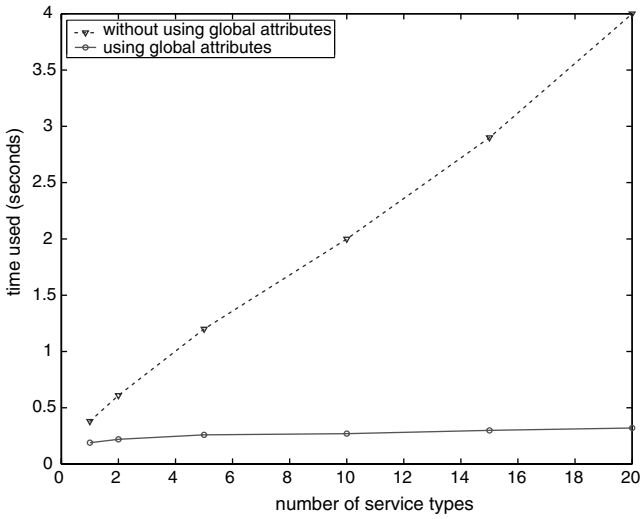

Fig. 8. Using global attributes versus without using global attributes: the time used for performing location-based queries from the PlanetLab node *gtidsl1*.

Fig. 8 shows the experimental results. We observe that without using global attributes, the total time used increases as the number of service types increases. In contrast, by using global attributes, the time used is roughly unchanged. We also run the above experiment using a local UA, and obtained a similar curve except that the time used by the local UA is about two orders of magnitude less than that of the UA at *gtidsl1*.

8. Related work

Fully meshed peer relationships are used in IBGP (Rekhter and Li, 1995). While an IBGP node is only in one mesh, a multi-scoped mSLP DA may belong to multiple meshes. mSLP supports flexible selective anti-entropy (Zhao and Schulzrinne, 2002) as well as traditional complete anti-entropy (Petersen et al., 1997). UDDI also uses anti-entropy to replicate registries, but only for full replications, whereas mSLP supports scope-based partial replications.

A number of service discovery systems such as Jini (Waldo, 1999) and UDDI (2004) support selecting and sorting search results, but none of them support generic preference filtering by composing these two basic operations. The LDAP sort control (Howes et al., 2000) and paging control (Weider et al., 1999) come closest to our proposed preference filters. But LDAP aims to send all search results back to the client via the paging control, whereas we simply try to send selected number of search results to the client. Furthermore, we support reference-based sorting and support composing multiple select and sort filters.

Our work on SLP global attributes was motivated by Guttman who describes how to use service identifiers as SLP service keys in (Guttman, 2002). Guttman's proposal needs to use hierarchical attributes when a service has multiple URLs with different properties. We believe that it is simpler to keep service URLs as service keys and define service identifier as a global attribute. Similar to global attributes, Jini (Waldo, 1999) defines a set of common entry classes in the *net.jini.lookup.entry* package. When a Jini search specifies multiple interfaces (such as *Toaster* and *FireAlarm*), it means to find services that implement all specified interfaces (logical "and"). In contrast, when an SLP `SrvRqst` message specifies multiple service types, it means to find services of any specified type (logical "or").

9. Conclusion

This paper presented three new mechanisms for SLP: the mesh enhancement that simplifies SA registrations and improves the consistency of peer DAs, preference filters that facilitate processing of search results in SLP servers, and global attributes that allow using a single query to search services across multiple types. These mechanisms can improve SLP efficiency and scalability, and support advanced discovery. Although we discuss these techniques in the context of SLP, we expect that they can also be applied to other service discovery systems. Preference filters and the mesh enhancement are now experimental Request for Comments (RFCs) (Zhao et al., 2002, 2003) after review by the IETF.

References

Apple, 2004. Rendezvous. Available from URL: <http://developer.apple.com/macosx/rendezvous/>.

Bakke, M., et al. 2003. Finding iSCSI targets and name servers using SLP. Internet draft, Internet Engineering Task Force, work in progress. Available from URL: <http://www.ietf.org/internet-drafts/draft-ietfips-iscsi-slp-06.txt>.

Berners-Lee, T., Fielding, R., Masinter, L., 1998. Uniform resource identifiers (URI): generic syntax. RFC 2396, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2396.txt>.

Bluetooth, 2004. Available from URL: <http://www.bluetooth.com>.

Bray, T., Paoli, J., Sperberg-McQueen, C.M., 2000. Extensible markup language (XML) 1.0 (second edition). W3C Recommendation REC-xml-20001006, World Wide Web Consortium (W3C), Available from URL: <http://www.w3.org/XML/>.

Droms, R.E., 1997. Dynamic host configuration protocol. RFC 2131, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2131.txt>.

Gulbrandsen, A., Vixie, P., Esibov, L., 2000. A DNS RR for specifying the location of services (DNS SRV). RFC 2782, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2782.txt>.

Guttman, E., 2002. The serviceid: URI scheme for service location. Internet draft, Internet Engineering Task Force, work in progress. Available from URL: <http://www.ietf.org/internet-drafts/draft-guttman-svrloc-serviceid-02.txt>.

Guttman, E., Perkins, C.E., Kempf, J., 1999a. Service templates and service: Schemes. RFC 2609, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2609.txt>.

Guttman, E., Perkins, C.E., Veizades, J., Day, M., 1999b. Service location protocol, version 2. RFC 2608, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2608.txt>.

Herriot, R., Butler, S., Moore, P.G., 2000. Internet printing protocol/1.1: Encoding and transport. RFC 2910, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2910.txt>.

Howes, T., 1997. The string representation of LDAP search filters. RFC 2254, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2254.txt>.

Howes, T., Wahl, M., Anantha, A., 2000. LDAP control extension for server side sorting of search results. RFC 2891, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2891.txt>.

IETF, 2004. Zero configuration networking working group. Available from URL: <http://www.ietf.org/html.charters/zeroconfcharter.html>.

Kempf, J., Montenegro, G., 2001. Finding an RSIP server with SLP. RFC 3105, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc3105.txt>.

Meyer, D.L., 1998. Administratively scoped IP multicast. RFC 2365, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2365.txt>.

Naugle, J., Kasthurirangan, K., Ledford, G., 2001. TN3270E service location and session balancing. RFC 3049, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc3049.txt>.

Perkins, C.E., Guttman, E., 1999. DHCP options for service location protocol. RFC 2610, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2610.txt>.

Petersen, K., Spreizer, M.J., Terry, D.B., Theimer, M.M., Demers, A.J., 1997. Flexible update propagation for weakly consistent replication. In: ACM Symposium on operating systems principles, Saint Malo, France.

PlanetLab, 2004. PlanetLab. Available from <http://www.planet-lab.org/>.

Poynor, T., 2001. Automating infrastructure composition for internet services. In: Systems Administration Conference (LISA). San Diego, California.

RDF, 2004. Resource description framework. Available from URL: <http://www.w3.org/RDF/>.

Rekhter, Y., Li, T., 1995. A border gateway protocol 4 (BGP-4). RFC 1771, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc1771.txt>.

Reynolds, F., 2001. An RDF framework for resource discovery. In: Semantic Web Workshop, Hong Kong, China.

Salutation, 2004. Available from URL: <http://www.salutation.org>.

Stewart, R.J., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., 2000. Stream control transmission protocol. RFC 2960, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2960.txt>.

UDDI, 2004. Universal description discovery and integration. Available from URL: <http://www.uddi.org/>.

UPnP, 2004. Universal plug and play. Available from URL: <http://www.upnp.org>.

UUID, 2004. Universal unique identifier. Available from URL: <http://www.opengroup.org/onlinepubs/9629399/apdxa.htm>.

Waldo, J., 1999. The Jini architecture for network-centric computing. Communications ACM 42 (7), 76–82.

Weider, C., Herron, A., Anantha, A., Howes, T., 1999. LDAP control extension for simple paged results manipulation. RFC 2696, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc2696.txt>.

Zhao, W., Schulzrinne, H., 2002. Selective anti-entropy. In: ACM Symposium on Principles of Distributed Computing, Monterey, California.

Zhao, W., Schulzrinne, H., 2004a. DotSlash: a scalable and efficient rescue system for handling web hotspots. Technical Report CUCS-007-04, Department of Computer Science, Columbia University.

Zhao, W., Schulzrinne, H., 2004b. Enabling global service attributes in the service location protocol. Internet draft, Internet Engineering Task Force, work in progress. Available from URL: <http://www.ietf.org/internet-drafts/draft-zhao-slp-attr-03.txt>.

Zhao, W., Schulzrinne, H., 2004c. Locating IP-to-Public switched telephone network (PSTN) telephony gateways via SLP. Internet draft, Internet Engineering Task Force, work in progress. Available from URL: <http://www.ietf.org/internet-drafts/draft-zhao-iptel-gwloc-slp-06.txt>.

Zhao, W., Schulzrinne, H., 2004d. Service location protocol enhancements project. Available from URL: <http://www.cs.columbia.edu/IRT/mslp>.

Zhao, W., Schulzrinne, H., Guttman, E., 2003. Mesh-enhanced service location protocol (mSLP). RFC 3528, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc3528.txt>.

Zhao, W., Schulzrinne, H., Guttman, E., Bisdikian, C., Jerome, W., 2002. Select and sort extensions for the service location protocol (SLP). RFC 3421, Internet Engineering Task Force. Available from URL: <http://www.rfc-editor.org/rfc/rfc3421.txt>.

Zhao, W., Schulzrinne, H., Guttman, E., Bisdikian, C., Jerome, W.F., 2004. Remote service discovery in the service location protocol via DNS SRV. Internet draft, Internet Engineering Task Force, work in progress. Available from URL: <http://www.ietf.org/internet-drafts/draft-zhao-slp-remote-da-discovery-06.txt>.