A SCALABLE SERVICE DISCOVERY FRAMEWORK WITH LOAD SHARING CAPABILITIES

Manuel Urueña, David Larrabeiti and Pablo Serrano
Universidad Carlos III de Madrid Av. Universidad 30, Leganés. Spain
{muruenya,dlarra,pablo}@it.uc3m.es

Abstract In order to ease mobility, users should be able to access available services and resources of the local network without manually reconfigure its terminal at each visited network. Therefore, some kind of automatic mechanism is needed to dynamically locate the best available services from any network. The eXtensible Service Discovery Framework (XSDF) is a novel solution to this problem. XSDF is an evolution of the Service Location Protocol (SLP) architecture, that also integrates the load balancing and high-availability capabilities from the Reliable Server Pooling (Rserpool) framework in order to bridge together scalable service discovery with extensible load sharing selection policies. This paper provides a brief overview of XSDF, and compares it against SLPv2 (including its Attribute List Extension) employing several simulations in different scenarios.

1. Introduction

Mobility may become "the next big thing", wireless technologies as Wi-Fi or 3G will allow users to connect anytime, anywhere to the ubiquitous Internet, and laptops roaming between multiple networks may become commonplace. However, this future scenario poses many challenges at multiple layers, from low-level wireless transmission to user programs, in order to achieve seamless roaming between networks.

In particular, applications should not be statically configured anymore, they should adapt themselves dynamically to the new network. For example, the user's web browser should be configured with the local web proxy, the list of the available printers should be updated, etc.

Scalability is another important service-related challenge for mobile networks. They must be carefully planned to easily grow from a couple of users to hundreds when a special event takes place. In this case, the network would need additional bandwidth and service resources. The latter can be achieved by replacing the old server with a better one or just by deploying additional low-cost servers.

Although both issues have been hot topic for many years, as far as we know, load sharing has not been integrated in any service discovery framework to date.

This paper provides a brief overview of the eXtensible Service Discovery Framework (XSDF), a scalable architecture that integrates into a single process the discovery and selection of network services, even from remote locations across Internet. Thus, mobile terminals connected to a network supporting XSDF will be able to locate the best available services dynamically, without any kind of manual configuration.

Service Location Protocol (SLP)

The benefits of Service Discovery have driven the development of many protocols, including SLP [3], an IETF standard that has been designed to find available services in the local network. However, while most of the Service Discovery protocols are focussed on unmanaged LANs, as a typical SOHO environment, SLP has been carefully designed to scale from these small LANs to big enterprise sites.

This scalability is achieved by combining multiple mechanisms. In small networks without a stable infrastructure, **User Agents (UA)** locate services by issuing multicast queries to the so-called **Service Agents (SA)** which reside in all the computers providing Services. However, when the number of users and services increase, the multicast traffic could overwhelm the network. In that case, an optional entity called **Directory Agent (DA)** may be deployed as a central repository of service information, where Service Agents publish their Services. This allows User Agents to send unicast queries to the Directory Agent. Thus, multicast is not employed but to bootstrap and locate the DA[1]. In order to scale even more, or when an organization has different departments, groups of Users and Services may be split in several independent **Scopes** which may have their own Directory Agents.

Furthermore, SLP achieves this scalability while remaining lightweight and simple. However, this simplicity has become sometimes a problem. A good example is the way a "Service" is modeled. In SLP a Service is represented just as a URL plus an optional set of attribute-value pairs, and the URL is also employed as the Service's identifier. In [5], Guttman analyzes the problems derived from this design: 1) a service is tied to a single location, 2) it cannot be accessed by multiple protocols, and 3) URLs cannot tell which transport protocols may be employed to access a service.

To overcome these problems Guttman suggests [5] to identify services with a UUID URN, while its supported transport and application protocols, location

[1] Multicast is not even needed if DHCP is employed to configure the DA location

and additional information (name, description, etc) are carried as attributes. While this modification allows protocol messages to be backward compatible, the User Agents and even the applications must modify its behavior. Also, complex services may require too many textual-encoded attributes, which may lead to a severe overhead, as we will analyze in section 3.

Reliable Server Pooling (Rserpool)

As employing multiple replicated servers has become a common mechanism to achieve incremental scalability and high-available Services, load balancing between loosely-coupled servers has become a hot topic in both the literature and commercial arena, especially for web servers [1].

Although the DNS-Round Robin technique has become quite popular, it provides just "coarse-grained" load sharing, specially under high loads. Therefore many organizations have deployed L4/L7 Switches (a.k.a. "Load Balancers") in the front-end of the web-cluster in order to achieve a better load distribution. These elements behave as a NAT, hiding the back-end servers, and implement different selection algorithms to choose the best server based on metrics as the server weight, workload, response time or current number of connections.

An alternative mechanism for load sharing is the client-side server selection [2], which is fully compatible with the end-to-end architecture of Internet. The ongoing Rserpool IETF Working Group is a good example of this kind of technique. It defines an architecture [9] very similar to SLP, where a set of servers providing the same service are aggregated into a **Pool**. Before a client accesses a particular server of a Pool, the **Pool User (PU)** asks to the server pool manager -called **Name Server (NS)**- for all the available **Pool Elements (PE)**, and finally, the PU chooses the best one among them. As the server selection is very dependent of the service, Rserpool is extensible and allows several load balancing mechanisms and metrics. Moreover, as the client knows alternative servers for a service, it may choose, via the **Aggregate Server Access Protocol (ASAP)** [8], an alternative server if the first one fails. Besides, in order to provide scalability and high-availability to the name resolution process, Rserpool employs several Name Servers that are coordinated by the **Endpoint Name Resolution Protocol (ENRP)** [16].

Rserpool employs SCTP as a mandatory transport protocol. However, the delay of the SCTP connection setup when asking the Name Server may be too high for some interactive services. For example Web browsing, where the overhead of the much simpler DNS name resolution mechanism could affect [7] the user-perceived latency.

2. eXtensible Service Discovery Framework (XSDF)

The eXtensible Service Discovery Framework (XSDF) has been designed to try to solve the issues outlined in the previous section altogether, as it integrates both, Service Discovery and Load Sharing into a common architecture. Like other Service Discovery Frameworks, XSDF is composed of multiple Agents and the protocols to communicate among them, as shown in figure 1. In particular, XSDF borrows the architecture and nomenclature from SLP, although there are many differences between them. Maybe the most visible one is that XSDF defines four, simpler protocols instead of a single one as SLP does. Each of these client-server protocols deals with a different part of the Service Discovery process:

- **eXtensible Service Location Protocol (XSLP)** [12]: User Agents (UA) employ this protocol to get Service information from Service Agents (SA) and/or Directory Agents (DA). This is the main protocol of XSDF and allows either unicast queries to a DA, or multicast queries to all SAs when no DAs have been deployed.

- **eXtensible Service Register Protocol(XSRP)** [13]: When a Scope is managed by one or more Directory Agents, Service Agents must employ XSRP to register Service information at their DA, aggregating all Services of the Scope in a central cache to enable XSLP unicast queries.

- **eXtensible Service Subscription Protocol (XSSP)** [14]: This protocol allows any XSDF Agent to be subscribed to some Services by registering a notification channel. Then, all the changes of the Service information covered by the specified subscription are notified through that channel.

- **eXtensible Service Transfer Protocol (XSTP)** [15]: In order to achieve high availability and scalability, each Scope can be managed by multiple Directory Agents. These DAs employ the XSTP protocol to obtain all the Service information known by the other DAs, and later, to keep the common service repository in sync.

Although XSDF may seem more complex than SLP or Rserpool that only employ one or two protocols, the authors consider that this separation eases the operation and management of the whole framework. For example, in SLP and Rserpool, UA/PU queries and SA/PE registrations belong to the same protocol (SLP/ASAP), although each operation has very different semantics, source and destination Agents, and security requirements. By splitting these disjoint operations in two protocols, it is easier to enforce simple pool perimeter security by filtering all XSDF protocol ports but the XSLP one, to allow external UAs queries.

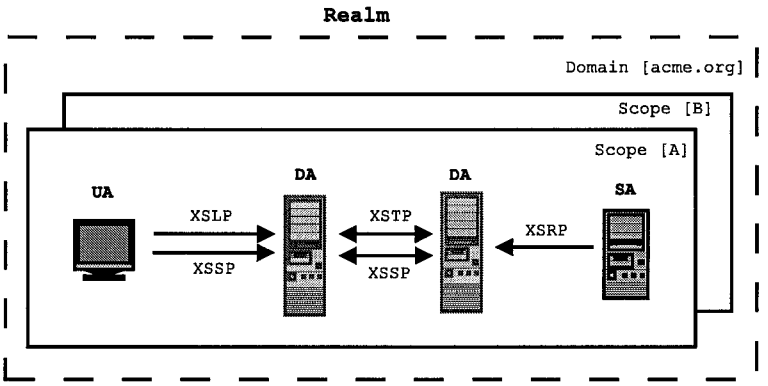

Figure 1. Overview of the XSDF architecture and protocols.

Service Model

In SLP, a Service is defined by a URL and a set of attribute-value pairs. In Rserpool, Service information only contains one or more transport protocol identifiers, that is, a port and one or more IP addresses per transport protocol, along with its selection policy information.

The XSDF Service model mixes both approaches by defining an extensible description model, that allows complex Services to be represented in a compact-manner by employing the eXtensible Binary Encoding (XBE32) [10]. The XSDF Service model splits Service information into five parts:

- **Service Identifier**: Each Service is identified by a 128 bit UUID, which must be unique whithin a Domain.
- **Service State**: It contains the volatile information about the Service that cannot be cached for long time (e.g. the workload of the server). Also, Service State includes some version fields of the other Service Information elements, to easily check if they have changed.
- **Service Main Information**: These data includes the Type of the Service, its selection policy, and other important Service-dependent information. For example, whether a printer supports color or not.
- **Service Location Information**: In order to establish a connection with a network Service, clients need to know where to reach it, and the available protocols to access the Service. "Location Information" specifies such data, as IPv4/6 addresses, transport ports and upper application protocols.
- **Service Additional Information**: While previous information allows automatic Service Discovery, the so-called "Additional Information" con-

tains data intended for human interaction as a textual description of the Service or the contact data of the Service administrator.

When requesting Service information, a UA may specify which kind of data it is interested in. This allows non-interactive UAs to download Service State, Main and Location Information to choose a Service, and later just to check the State to see if other Service information needs to be updated. Everything without downloading the Additional information that may be quite big as it is very verbose.

This service model is so extensible, that it even allows XSDF Agents to be represented as Services[2], thus XSDF Agents may be discovered as any other Service and they also benefit from the load-sharing capabilities of the framework.

Service Selection Model

When a UA requests Service information via XSLP, it can find several Service instances that fit its needs. In that case, some kind of selection process should take place. For example, a list of all Services (e.g. printers) could be displayed to the user who chooses one of them.

For Services that allow automatic selection, XSDF provides a version of the selection model from Rserpool. Each Service Type may have an associated selection policy, and each server may provide some metrics about the "goodness" of its Service instance. XSDF allows each Service Type to implement its own selection policy and its related metrics, although XSDF defines the following standard selection policies:

- **Round Robin**: This policy states that each time a client wants to access a Service, it should choose a different one. Services may have an associated `weight` information, thus Service instances with greater `weight` should be accessed proportionality more times than servers with lower `weight` values.
- **Least Used**: Services under this selection policy indicate that clients should access the server with the lowest `workload` state value.
- **Most Resources**: Services with this policy tell clients that they should access the server with the greatest `resources` state value. When a service has zero `resources` left, clients could try the next discovered service, sorted by the selection policy in use.

Along with the `weight`, `workload` and `resources` values, XSDF defines a fourth selection metric that is applied before any other selection policies:

[2] As a comparison, SLP has special messages to advertise Directory Agents

`priority`. When two Services of the same type have different `priority` value, clients should access the Service with the greatest one. Only if the Service with the greatest priority is offline, clients are able to access the next Service with greater priority.

Unlike ASAP [8], where selection process is always performed on the client side, in XSDF the selection process could be done by the UA, the DA, or both. SAs may include the selection data inside the Service information for the UAs to choose the best discovered Service. Also SAs may specify in the XSRP registration some selection information for the DA. In that case, the DA sorts the registered Services and, when a UA requests some Service information, the DA provides just a subset of the best Service instances.

Deployment Scenarios

As XSDF extends the SLP architecture, it also inherits its scalability properties. It could be deployed in a broad set of scenarios: from unmanaged LANs to big enterprise networks and the Internet. Each scenario requires only a subset of XSDF Agents and protocols.

In unmanaged LANs, XSLP is the only protocol needed, as User Agents may easily query Service Agents in multicast/broadcast for services, and later choose among them. However, in bigger networks it could be better to define Scopes and to deploy Directory Agents as multicast searches could be too inefficient. Therefore XSRP is needed to register Services at the DA, and Service selection could be performed both by the DA cache or by the final UA. At last XSTP and XSSP should be employed to synchronize the service repository of multiple DAs when high-availability is required.

An XSDF **Realm** extends the SLP Scope concept and adds a DNS domain name. Therefore XSDF may be also employed to advertise public Services in the Internet (e.g. a pool of Web servers may employ XSDF to achieve load balancing). This could be done by deploying public DAs managing the pool Services, and advertising those DAs in DNS SRV Resource Records. This mechanism [17] was also outlined for SLP although it was not part of the base specification.

In this scenario, a UA willing to access a Service from the `example.com` Domain firstly asks for the `_xslp._udp.example.com` DNS SRV Resource Record. This record contains a list of the preferred public DAs. Then, the UA is able to ask, via XSLP, for the desired Service (e.g. www) in the remote `PUBLIC` Scope. Lastly, the DA replies with the available, less-loaded servers.

3. A quantitative comparison between XSDF and SLPv2

This section compares the resource usage of the XSDF protocols compared to SLPv2 (and some of its variants) by evaluating several simulation scenarios.

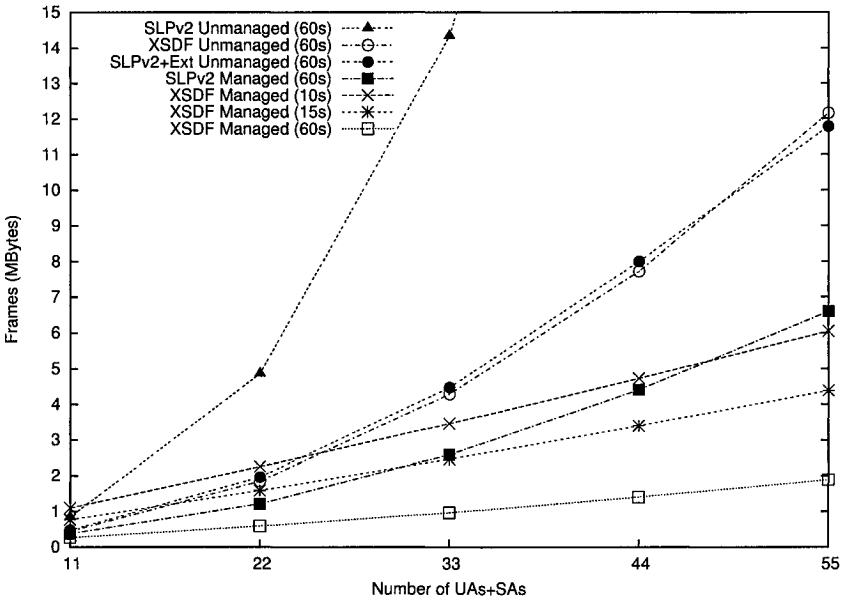

Figure 2. Simulation of XSDF and SLPv2 protocols

We have developed two simulation packages employing the OMNeT++ discrete event simulator [6]. The first one is an implementation of the SLPv2 protocol, the Attribute List Extension [4], the serviceid modification for complex services and the SLP API. The second one includes all three XSDF agents (UA, SA and DA), the XSLP and XSRP protocols, and a simple API derived from SLP's one. Both models are integrated in a simplified network environment with Server and Client hosts. Each Server has an associated Service Agent where it registers the Service it provides, and each Client requests services to its local User Agent.

Figure 2 shows the sum of all layer-2 frames [3] (in Megabytes) generated by several Service Discovery protocols in a 8 hour simulation of different scenarios. The X axis shows the number of User Agents and Service Agents of the network (there is a 10:1 ratio between clients and server, thus the 22 network has 2 SAs and 20 UAs). The Clients request services every 15 minutes, while the Servers update their Service information every 60 seconds.

As the `Unmanaged` scenarios does not include a Directory Agent, there is no Service update traffic and Service Requests must be sent in multicast packets. As it could be seen, basic SLPv2 multicast search does not scale well in big networks because two multicast requests are required, the first one (`SrvRqst`)

[3] A single multicast packet generates $N - 1$ layer-2 frames.

to obtain the Service URI and a second one (`AttrRqst`) to fetch the Service Attributes. As XSLP, the Attribute List Extension [4] for SLPv2 (`SLPv2+Ext`) allows UAs to get the Service URL and its attributes with a single multicast search (`SrvRqst`), thus both protocols show a dramatical improvement compared to basic SLPv2 muticast search.

The `Managed` scenarios have an additional host that runs a Directory Agent which centralizes all Service information. In those cases XSDF shows a clear advantage over SLPv2, due to the compact XBE32 encoding [10], and specially due to the XSDF Service Model. While SLP User Agents always need to fetch the entire Service data (including unnecessary one), XSDF UAs may keep the information in the local cache and request periodically just the Service State, which also allows UA to know when other Service information changes. For example, the traffic required by basic SLPv2 to update Service information every 60 seconds will allow XSDF to update Service information between 10 and 15 seconds, an improvement greater than 4x.

Although the Attribute List Extension has many advantages in multicast search, it may become a problem for unicast requests of complex Services, due to MTU restrictions. SLPv2 and XSDF messages must fit into a single UDP datagram, thus the number of Services in the reply is limited. This is not a problem for basic SLPv2 as the first `SrvRply` just contains the Service URIs, and their Attributes are sent in separate `AttrRply` messages. With the Attribute List Extension, the Service URI and its Attributes are returned inside a single message, and therefore, the Service information size may become a problem. For example in the last scenario, basic SLPv2 and XSDF protocols were able to retrieve the information about all the 5 complex Services, while SLPv2 with the Attribute List Extension will only get the information of a single Service per request.

4. Conclusions

When a mobile user enters into a new network, she has two choices: The first one is to ask the network administrator for all the services she needs, and then to manually configure her applications. The second one is to employ some Service Discovery mechanism to dynamically locate the best available services.

This paper has provided an overview of one of these automatic mechanisms: the eXtensible Service Discovery Framework (XSDF), a novel solution for Service Discovery and Load Sharing that brings together the scalable architecture of the Service Location Protocol (SLP), with the high available load sharing service of the Reliable Server Pooling (Rserpool) framework.

In section 3 we have compared the resource usage of XSDF versus SLPv2 in different scenarios, including managed (i.e. with a Directory Agent) and

unmanaged Scopes. From these simulations we conclude that Services in a XSDF managed framework could be updated more frequently (4 times) than in a SLPv2 environment, while consuming less resources.

This study has also analyzed the Attribute List Extension and it shows that employing of this SLPv2 extension greatly improves the scalability of SLP in unmanaged environments. However, this extension should be deployed with care in managed scopes, as centralized requests may return only a small number of complex Services with many attributes, because SLPv2 messages are restricted to fit in a single UDP datagram.

References

[1] V. Cardellini, E. Casalicchio, M. Colajanni and P. S. Yu. The State of the Art in Locally Distributed Web-server Systems, IBM Technical Report RC22209. October 2001.

[2] S. Dykes, K. Robbins and C. Jeffery. An Empirical Evaluation of Client-Side Server Selection Algorithms. Proc. of INFOCOM 2000.

[3] E. Guttman, C. Perkins, J. Veizades and M. Day. RFC 2608: Service Location Protocol, Version 2. June 1999.

[4] E. Guttman. RFC 3059: Attribute List Extension for the Service Location Protocol. 2001.

[5] E. Guttman. The serviceid: URI Scheme for Service Location <draft-guttman-svrloc-serviceid-02.txt>. August 2002.

[6] OMNeT++ simulator site: <http://www.omnetpp.org>

[7] A. Shaikh, R. Tewari and M. Agrawal. On the Effectiveness of DNS-based Server Selection. Proc. of IEEE INFOCOM 2001. April 2001.

[8] R. Stewart, Q. Xie, M. Stillman and M. Tuexen. Aggregate Server Access Protocol (ASAP) <draft-ietf-rserpool-asap-08.txt>. October 2003.

[9] M. Tuexen, Q. Xie, R. Stewart, M. Shore, L. Ong, J. Loughney and M. Stillman. Architecture for Reliable Server Pooling <draft-ietf-rserpool-arch-07.txt>. October 2003.

[10] M. Urueña and D. Larrabeiti. eXtensible Binary Encoding (XBE32) <draft-uruena-xbe32-00.txt>. March 2004.

[11] M. Urueña and D. Larrabeiti. XSDF: Common Elements and Procedures <draft-uruena-xsdf-common-00.txt>. March 2004.

[12] M. Urueña and D. Larrabeiti. eXtensible Service Location Protocol (XSLP) <draft-uruena-xslp-00.txt>. March 2004.

[13] M. Urueña and D. Larrabeiti. eXtensible Service Registration Protocol (XSRP) <draft-uruena-xsrp-00.txt>. March 2004.

[14] M. Urueña and D. Larrabeiti. eXtensible Service Subscription Protocol (XSSP) <draft-uruena-xssp-00.txt>. March 2004.

[15] M. Urueña and D. Larrabeiti. eXtensible Service Transfer Protocol (XSTP) <draft-uruena-xstp-00.txt>. March 2004.

[16] Q. Xie, R. Stewart and M. Stillman. Endpoint Name Resolution Protocol (ENRP) <draft-ietf-rserpool-enrp-07.txt>. October 2003.

[17] W. Zhao. Finding Remote Directory Agents and Service Agents in the Service Location Protocol via DNS SRV <draft-zhao-slp-remote-da-discovery-06.txt>. March 2004.