

Service Location - Survey of mechanism to search and configure services

Stefan Schmidt stefan@datenfreihafen.org

2007-12-16

Abstract

The number of network service we can use with a computer grows every day. Printing documents, transfer files or listen to music on demand are just a small selection of such network services. But before you can use them you have to know their address and configure them correctly. On the other hand a service knows about the properties it has and users know what they want to do. A solution is required that will bring all the information together in nearly automatic way. The service discovery protocols discussed in this paper offer a dynamic way of searching and utilize a service. It aims to give an overview about the today mostly used protocols, describe advantages and disadvantages and compare the protocols in general.

1 Introduction

The services we use with network-enabled devices grow every day. Older services like printers, scanners and file servers will be supplemented by VoIP gateways, music on demand streaming servers and others. The rising numbers make it more and more complicated to detect and configure them manually. Users with mobile computers like notebooks or even mobile phones with wireless connectivity like to use services in foreign networks. Mobile Ad-Hoc networking requires services that work without any infrastructure.

Service Discovery or Location, used synonymously in this document, is located in the middle of a stack of protocols needed to reach this goal. See figure 1.

It can build upon a automatic network address distribution and provides its own capabilities to the above application layer. Nodes in a network need to know each other before they can share information about services. In a traditional network each PC has a static IP address configured on the PC itself or gets an IP address, a dynamic or static one, from a DHCP server. If the network does not offer the required infrastructure users are forced to handle address management on their own. A solution for this can be the one described in [1], a way of automatically using link local addresses. After a successful address allocation the service

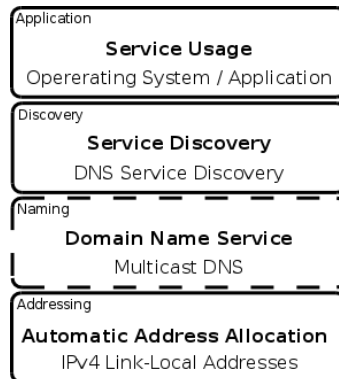


Figure 1: Zeroconf Protocol Stack

discovery can happen and supply applications with the discovered information. It is the responsibility of the upper layer to handle this information properly. A text processing software that wants to print the current document either needs to get the information about network addresses and properties of available printers itself or it has to rely on a printing API supplied by the operating system.

2 Use Cases

To illustrate what service discovery can do in scenarios most computer users have already encountered, we describe some common use cases of service discovery protocols. Of course these just focus on specific problems, but should also show the potential behind the protocols.

Two developers meet to work on a project together. Both carry their notebooks with integrated wireless cards. Instead of spending time exchanging network addresses, configuring network cards and creating logins for file transfers etc. they should be able to just create an Ad-Hoc wireless network. One publishes a versioning control system service which the other detects automatically, and both can get to work.

Another use case is all the bluetooth capable small gadgets we carry around with us. Imagine a car with an audio system with bluetooth. Once you enter the car it detects your mobile phone and your portable music player. Both gadgets announce their properties, handset and stereo audio. The car audio system discovers these services and establishes connections accordingly. Now it is possible to hear music through and also receive calls over the car audio system.

In addition to the benefits in mobile and Ad-Hoc networks it also helps in more traditional and static environments. Even in an office with wired computers and a fixed infrastructure the scenario is changing. A new network printer or file server can be plugged into the network and all clients automatically discover them. The user can be notified about the new printer or it will be offered the next time the user wants to print a document. Without service location, the

administrator has to walk from client to client and configure the service properly.

This is just a small overview of what service discovery protocols can be used for and what benefits they can bring. In the remaining part of this document some more details about the protocols and technologies will be shown. The third section gives a short survey of service location protocols in general. Sections four, five, six and seven discuss the most popular protocols in this area: Service Location Protocol (SLP) in section four, Simple Service Discovery Protocol (SSDP) in five, DNS Service Discovery (DNS-SD) in six, and some smaller ones in section seven. Afterwards we compare them in section eight. The conclusion takes place in section nine.

3 Survey of the different protocols

Several years ago vendors, universities and networking groups in general recognized that the rising numbers of network services need a way to handle these services in a more automatic and dynamic way. As with many problems people start thinking about the same problem independently. Besides this, different vendors or organisations like to focus their technology on different areas or build them around different technologies. This leads to many varying solutions for the same problem. Some fit a problem in a niche product, others use a special software stack or aim to provide a general solution. Due to these facts it is rather normal that the market for service discovery protocols offers different solutions for the same problem. Some have the exact same focus, others differ slightly.

Today we can choose between many solutions for the problem of finding a service in a network. Interestingly it still seems impossible to identify a leader in this market. At the moment groups are focusing on different aspects and consequently pushing different solutions into the market and the future will show who can enforce their solution. This is also the biggest current problem of service discovery protocols. They do not work well in heterogeneous networks. As long as the software portfolio from one vendor is used, the whole process of searching and using a service should work fine without much interaction from user or administrator side. Unfortunately bigger networks are almost always heterogeneous with soft- and hardware from different vendors. In such an environment there is still much administrative work needed. One solution for this could be some kind of bridge between the different service discovery protocols as discussed in the conclusion on page 11.

While dealing with the different protocols, it is sometimes interesting to know more about the non-technical background of the protocols we discuss in this document. Facts about which company is involved in the design or by what interests a consortium is driven can allow a better understanding of designs used in the protocol.

The Service Location Protocol is developed by the IETF networking group. The Internet Engineering Task Force offers a forum to interested companies and developers. The remaining standards will be discussed and published as Request For Comments (RFC) and are freely available. This allows writing standards

conforming protocol stacks under e.g. open source licenses. Most authors of such an RFC are engineers of companies and are in the working group on behalf of their company. The IETF is known for well designed, vendor-independent standards.

The Simple Service Discovery Protocol in contrast was designed by the Universal Plug and Play vendor consortium chaired by Microsoft. There were some efforts to publish SSDP as an RFC but unfortunately it was never finished. After some very early drafts no new proposals were submitted and the draft expired.

DNS-SD is again developed by the IETF, published as an RFC draft and offers the same possibilities and drawbacks as other standards published as RFCs. Jini, on the other hand, is a completely different approach as it is driven by SUN as an extension of the Java programming language. However, other companies are also working on Jini. As the last protocol discussed here, the Service Discovery Protocol of the Bluetooth stack has been designed and developed by the Bluetooth group. The specifications of the Bluetooth group are freely available and an open implementation exists e.g. in the BlueZ project [2] for the Linux kernel and suitable userspace applications.

4 Service Location Protocol (SLP)

The Service Location Protocol, hereafter referred to as SLP, is designed by the Internet Engineering Task Force (IETF) networking group. More precisely, the service location working group, abbreviated as SrvLoc. The current version is SLPv2 defined by [3]. It supersedes SLPv1 from [4].

SLP uses TCP/IP as a networking protocol and is designed with scalability in mind. Due to these two facts it can be used in small car, home or office networks as well as in a large enterprise network with hundreds of nodes. To achieve this kind of scalability it describes three types of components and two types of discovery flows.

User Agent (UA) searches for services announced to the network on behalf of the user or an application. It also can cache the informations.

Service Agent (SA) announces the location and properties of a service. This is done on behalf of the service itself. A SA can handle different services on one node and also one service which like to announce himself as different service types. A printer can, for example, announce himself as an IPP printer and unix printer.

Directory Agent (DA) receives announcements and responds to requests after collecting the information. It also stores all available services in its database and can be used as a central place for categorization and control.

As the DA is not mandatory, but just an optional feature, there are two different discovery flows. The one without DA, and shown in figure 2, looks like the following:

1. UA sends requests for the desired service to the SLP multicast group address (239.255.255.253).

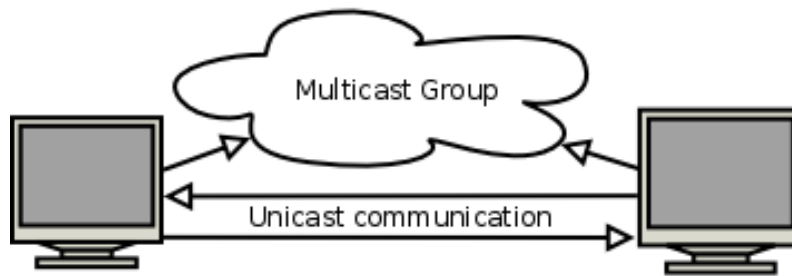


Figure 2: SLP direct discovery flow

2. SA listens to this address, checks whether it provides an adequate service and replies to the unicast address if it does. If the SA has no appropriate service it just waits for the next request.

Rather than merely listening, the SA can also announce its services periodically to the same multicast group address. The use of multicast group addresses allows an exchange of information between unknown nodes without wasting resources through broadcasting.

For larger installations, a separate DA could be a good idea. It offers the possibility to categorize services. In order to use the DA, the UA's and SA's have to know about its presence. There are three different ways to inform the UAs and SAs about the DA:

- **Static:** Advertise the DA address with DHCP.
- **Active:** SA's and UA's send service requests to the SLP multicast group address and DA replies.
- **Passive:** DA sends out advertisements via multicast.

Once the address of the DA is known, SA's can register their services and UA's can request services. The involved DA changes the normal discovery dataflow to the following (Figure 3):

- After a SA is connected to the network and has discovered the DA through one of the three ways described above, it sends a service registration to the DA. This registrations contain offered services and properties.
- Once a UA needs to discover a special service it asks the DA directly instead of using the multicast group.
- The UA now has all information required to directly utilize the service of the chosen SA.

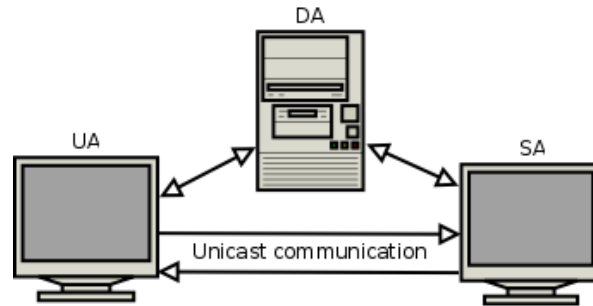


Figure 3: SLP discovery flow with Directory Agent

5 Simple Service Discovery Protocol (SSDP)

UPnP uses the Simple Service Discovery Protocol (SSDP) as their service discovery protocol. Although there was an attempt to publish SSDP as an IETF Internet Draft [5], it was never finished and the published version has been expired in April 2000. Therefore it is possible that different implementations can have tiny differences which can make them somewhat incompatible to each other. The following description is based on the last available draft and highlights differences to the UPnP implementation. An implementation of SSDP licensed under the GPL can be found at [6].

Also similar to SLP, SSDP uses a multicast group address (239.255.255.250) to discover and announce services. Instead of using a link local scope which would only allow discovery inside a subset of a local area network it uses an administrative scope to reach even resources or clients behind bridged or routed network equipment. SSDP contains three methods to reach its goals:

- Multicast Discovery Support
- Server Based Notification
- Discovery Routing

The transferred messages use the HTTP protocol over UDP, therefore the draft talks about HTTP clients and HTTP resources. It should be noted that SSDP is designed without an optional directory service or other central place to store and/or cache the service information.

Before we take a more detailed look on how such a message looks like, an overview of the message flow will be shown. At the first glance there are two major approaches to how clients and resources can discover each other. The client could send out a request periodically during its search to make sure that it will also be received by resources that joined the group after the initial request. The opposite approach would let the resource send out notifications consistently. SSDP combines these two approaches for lower bandwidth usage.

It could be considered an event driven system as messages will only be sent once an event occurs. Events are: Client or resource goes on-line, client or re-

source goes off-line, and status changes. In the end, such a system would allow far less messages to be sent to the group as no polling for changes is required.

To describe a service inside a SSDP system four handles need to be known:

- Unique Service Name (USN) URI
- Service Type URI
- Expiration Time
- Location

Service type URI is only used as a kind of partition without being further specified in the draft. The USN URI's are used to identify a single instance of a service type. It will even stay the same when the location changes. This allows applications to still have a valid handle even if the location of the resource changes. The cache can update the location once a new notification is sent. As expected, the location describes the service location. Due to the dynamic nature service discovery protocols need to handle disappeared services. SSDP handles this with an expiration timer.

6 DNS Service Discovery (DNS-SD)

DNS Service Discovery is an approach of re-using standard DNS infrastructure to browse the network for services. It adds service information into DNS records to allow clients to receive this information through normal DNS communication. Besides this static way of distributing the service information, it also re-uses the dynamic update procedures which are currently used for updating the hostname from the client. Both approaches need the permission to change configurations on the DNS server and, even more importantly, they need infrastructure with a DNS server already running.

Aside from such static environments, DNS-SD can also be used with multi-cast DNS (mDNS) [7]. Together with automatic link local address configuration these three protocols define the Zeroconf protocol stack shown in figure 1. This stack is more familiar under the name Bonjour as Apple uses it for the Zeroconf stack in its Mac OSX products. It allows dynamical and infrastructure-free usage of service discovery suited for Ad-Hoc networks or small home or office local area networks.

Another interesting fact is that, due to the use of standard DNS, DNS-SD is also capable for worldwide usage in wide area networks or the internet. This is especially interesting as this is out of scope of all other service discovery protocols discussed in this document. It would be possible to discover services from domains you only can connect over the internet.

Based on the type of service and the domain in which the client is looking, the DNS can deliver a list with named instances of these services. The whole protocol requires no changes to the existing DNS record types or the structure of DNS messages. Everything can be accomplished by using standard DNS queries. This avoids the problem that changes in DNS software would be needed which

would probably take years for engineering and deployment - if it would happen at all. The draft [8] only introduces a convention about how record types can be named and structured. The complete DNS-SD records can be rerouted to another DNS server by delegating the `_tcp` subdomain to another DNS server if a separation or load balancing is needed.

The records used for DNS-SD are SRV records, or service records, defined in [9]. SRV records can define types of service. The following example defines an internet printing protocol service: `_ipp._tcp.domain.tld`

Since DNS servers with SRV records were available before DNS-SD, there is something new DNS-SD must bring in to extend this functionality. In fact it allows the client to search e.g. all IPP services and choose one of them by other properties like paper size. This does not work with SRV records alone. For this DNS-SD uses PTR records instead of plain SRV records to point to a heap of services of the same type. This allows a listing of all services from one type.

UTF8 can be used for service descriptions. This offers the possibility to show the user sane formatted names and labels instead of cryptic looking names due to prohibition of spaces or limitations of length [10]. Listing 1 shows an example for defining a service entry for a HTTP served website in a Bind9 DNS server. The service is named Intranet Website, hosted on `www.foo.org` port 80 and the path to the html document is `/index.html`.

Listing 1: DNS-SD example entry in Bind

```

; Enable browsing of services
b._dns-sd._udp IN PTR @ ; b = browse domain
lb._dns-sd._udp IN PTR @ ; lb = legacy browse domain

; List of services
_http._tcp PTR Intranet\ Website._http._tcp

; Describe the service with SRV and TXT records
Intranet\ Website._http._tcp SRV 0 0 80 www.foo.org.
                                TXT path=/index.html

```

7 Other protocols

7.1 Jini

Designed by SUN Microsystems, Jini is a technology used to build so called Jini communities. These communities are small Ad-Hoc networks which need a way to discover services of other members of the community. Jini extends the Java programming language hence it needs a Java Virtual Machine (JVM) running on the device [11].

The protocol to discover services is somewhat similar to the one used in SLP. Once a device joins a community it places its available services into a lookup table. A lookup table is comparable to a directory agent in SLP. For a better

handling of the dynamical joining and, even more importantly, leaving of communities without a special notification Jini uses the concept of leases. A device can just disappear e.g. it gets out of the range of the wireless connection to the other device. The lease is only of use for a certain time period. When the period expires and the device does not request a new lease its services will be removed from the lookup table. The concept of leases is similar to the one used, for example, in DHCP.

The concept of the lookup table also holds the major difference between Jini and other discovery protocols. The table can not only store a pointer to a service, but also object code such as device drivers or similar. A device that likes to use this service downloads the java code and has direct access to the service. Such a solution has the advantage that the device that wants to utilize the offered service does not need to have a special device driver or client software installed. The lookup table contains all the necessary code. On the other hand this can be a huge security problem as a malicious service can inject code directly into a device.

The Jini Starter Kit code is available under a free software license from the Apache Software Foundation incubator.

7.2 Bluetooth Service Discovery Protocol

The Bluetooth specification also contains a protocol to discover services [12]. It is really only designed to discover services, no functionality is specified about how to access a service after its discovery. This job is done by other protocols of the Bluetooth protocol stack. Due to this fact it can also be used together with other such protocols although it does not require them.

The functionality SDP offers has three ways of searching for services.

- Type of service
- Attributes of the service
- Browse the complete list.

While the first two variants offer a parameter to the search, type or attribute, the third one allows to brows the whole list of offered services. A search with a known parameter is of course more efficient and should fit the normal usage scenario.

8 Comparison of the protocols

Even if the protocols were designed by different consortiums they all have more or less the same goal: To make it easier for users to discover and use network services. It is not surprising that the ideas behind them are pretty similar.

Announcing a service or a service request to a network, ideally via multi-cast. The full communication between the client and the server, after the initial discovery, is handled directly via unicast. Based on the underlying network

technologies this was the next logical step. Only service discovery protocols using other underlying technologies such as bluetooth and Java are using different approaches.

All protocols discussed here have their own positive and negative aspects. That is not limited to technical details, but also license and standardization issues. To cover these differences this section compares the protocols described above.

SLP The biggest advantage of SLP is its IETF background. It was designed by a vendor-independent group and has freely available documentation. No license fees and no need for a consortium membership. There are several reference implementations of SLP and it is already used in many commercial products. In addition to these non-technical facts, SLP can also show its strengths on scalability and flexibility. It can be used in tiny Ad-Hoc networks as well as in big enterprise networks. Adding options for LDAP as storage backend, IPv6 support, DHCP options [13] and a leasing model shows the real potential of SLP.

However, it shares a big disadvantage with all other candidates besides DNS-SD: It uses a new kind of protocol which needs to be implemented, tested and deployed in new devices and software products. But before it is not deployed in a wide range of software and hardware products it can not establish its full potential for the users.

SSDP As SSDP is used only in Microsoft's Universal Plug and Play platform it grows with the wider usage of UPnP. In 1999, a very early draft was developed to standardize SSDP through IETF. Unfortunately, these documents expired and were never refreshed; this leads into a situation where no standardization documents exist which makes it hard to interact with other implementations of SSDP. Besides these rather formal problems, SSDP contains a major technical drawback: It does not allow the client to search for the properties of a service. Due to this problem, SSDP is not a good choice regarding the technical aspects of the protocol, even if it gets a broader audience due to the spread of UPnP.

DNS-SD In comparison to the other protocols discussed here, DNS-SD extends an already existing approach instead of supplying a new one. It extends the Domain Name Service protocol which is used worldwide and thus well tested. It adds service attributes to the old but stable approach of name resolution. This reuse of existing technology is an advantage compared to its competitors. The DNS infrastructure already deployed in large enterprise or small home networks makes it an approach which is easy to use. For networks with a more dynamic nature multicast DNS closes the gap of changing infrastructure. As part of the Zeroconf stack DNS-SD has a promising future. Implementations for this stack have already been done in Bonjour for Mac OSX and Windows and Avahi for Linux.

Jini Jini is the java based approach for service discovery. The usage of Java makes it platform independent and also offers the possibility to run on every device which runs a Java virtual machine. The JavaVM can of course be a problem for low-memory and low-power devices. Small network devices might just not be able to run a full JavaVM to offer its services in a dynamic way. This makes Jini a niche product even though it is a somewhat big niche as many devices have

Feature	SLP	SSDP	DNS-SD	Jini	SDP
Organisation	IETF	UPnP Consortium	IETF	Sun Microsystems	Bluetooth group
License	Open Source	Open Source	Open Source	Apache 2.0	Open Source
Language	independent	independent	independent	Java	independent
Network Protocol	TCP/IP	TCP/IP	TCP/IP	independent	Bluetooth
Ad-Hoc feasible	Yes	Yes	Yes	Yes	Yes
Central directory	Yes, optional	No	Yes, optional	Yes	No
Leasing	Yes	Yes	Yes	Yes	No
WAN capable	Limited	No	Yes	No	No
Notes	-	Never finished	In progress	Java RMI	-

Table 1: Differences between Service Discovery protocols

the possibility to run a JavaVM. Jini also offers an interesting feature which is not present in other service discovery protocols. It allows to store object code such as a driver or client program which the client can download to use the service. This is also based on a Java feature called Java Remote Method Invocation (RMI). As interesting as this feature is a bad feeling remains when a client simply executes code a service offers.

SDP As Jini SDP is more of a niche product instead of a generic solution for a service discovery protocol, SDP was designed to live inside the Bluetooth stack. The only function SDP has is to discover services without an idea how to access them later. Services discovered by SDP are handled by other parts of the Bluetooth stack afterwards. This small focus makes it unuseable without other protocols. Besides the other Bluetooth protocols it can also be used in combination with full service discovery protocols such as SLP.

9 Conclusion

We now have discussed the most commonly used service discovery protocols. It shows that they often have a rather similar design: Using multicast for discovery and announcements without a central place of registration. With such a place the initial request for address and service goes to the central directory and all following communications are handled by unicast and take place directly between server and client.

It is also apparent that an approach is needed that works fine with completely dynamical Ad-Hoc networks built up by small devices with wireless connectivity. These devices come and go without further notification, hence some kind of leasing concept is needed; but also good support for a more static infrastructure should be present. It should be possible to attach a network printer in the office and all clients learn about its presence and attributes.

As an approach for generic protocols, SLP and DNS-SD seem to have the best chances. Both already are -or should be soon- IETF standardized, well documented and have reference implementations which are already used in many products available on the market.

One of the most important things in the area of service discovery is some kind of bridging between the different protocols. It is a fact that there will never be only one protocol used in the networks. We must make sure that the discovery of services works even across protocols. At least the functionality to bridge service requests and service announces between SLP and DNS-SD would allow one to use the two biggest protocols simultaneously.

References

- [1] S. Cheshire, B. Aboba, and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses," Tech. Rep. 3927, May 2005.
- [2] BlueZ website: <http://www.bluez.org/>.
- [3] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," Tech. Rep. 2608, June 1999. Updated by RFC 3224.
- [4] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, "Service Location Protocol," Tech. Rep. 2165, June 1997. Updated by RFCs 2608, 2609.
- [5] P. L. Yaron Y. Goland, Ting Cai and S. Albright, "Simple Service Discovery Protocol/1.0," tech. rep., 1999.
- [6] Gupnp website: <http://svn.o-hand.com/repos/gupnp/trunk/gssdp/>.
- [7] S. Cheshire and M. Krochmal, "Multicast DNS," tech. rep., 2006.
- [8] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," tech. rep., 2006.
- [9] A. Gulbrandsen, P. Vixie, and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)." RFC 2782 (Proposed Standard), Feb. 2000.
- [10] DNS-SD website: <http://www.dns-sd.org/>.
- [11] Jini website: <http://www.jini.org/>.
- [12] B. SIG, "Bluetooth Core Specification v2.0 + EDR Part B," tech. rep.
- [13] C. Perkins and E. Guttman, "DHCP Options for Service Location Protocol," Tech. Rep. 2610, June 1999.