Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik IV
Prof. Dr. rer. nat. Otto Spaniol

Instant Networking and Dynamic Service Discovey

Seminar: Ubiquitos Computing
WS 2005/2006

Tim Niemueller

Matriculation number: 236104

Supervisor: Karl-Heinz Krempels
Lehrstuhl für Informatik IV, RWTH Aachen

Abstract

In today's mobile world inter-machine communication has become the key to information exchange - be it file transfers, chats or web surfing. Until now if you wanted to work in an ad-hoc network with a few peers you had to set most (if not all) information by hand. Then you exchanged information for the services you wanted to use. So at best networking was an annoyance, at worst it was a show stopper.

Zero Configuration Networking aims to solve that problem by defining a set of protocols that can be used to assign IP addresses automatically, resolve names and discover services. This seminar paper describes several candidates to accomplish these tasks and gives reasons for a specific set of protocols, namely IPv4 Local-Link Addressing, Multicast DNS and DNS-based Service Discovery.

Contents

1 Introduction **5**

2 Instant Networking **6**

2.1 Networking parameters 6

2.1.1 Internet Protocol 6

2.2 Managed Network 7

2.2.1 Dynamic Host Configuration Protocol 7

2.3 Ad-hoc Network 9

2.3.1 Link-Local IPv4 addressing 9

3 Name Resolution in Ad-hoc Networks **11**

3.1 Multicast networking 11

3.2 Link-Local Multicast Name Resolution 12

3.3 Multicast DNS 12

3.3.1 Name Reservation 13

3.3.2 Querying Information 13

3.3.3 Caching 14

4 Dynamic Service Discovery **14**

4.1 Universal Plug and Play (UPnP) 15

4.2 Jini 15

4.3 Service Location Protocol (SLP) 16

4.4 DNS-based Service Discovery (DNS-SD) 17

4.4.1 Protocol Overview 17

4.4.2 DNS-SD in Ad-hoc Networks 19

4.4.3 Traffic Reduction 19

4.4.4 Service Registration 20

5 Comparison **20**

5.1 Name Resolution 20

5.2 Service Discovery 21

6 Conclusion **24**

List of Algorithms

1 Link-Local Address selection 11

1 Introduction

In today's mobile world inter-machine communication has become the key to information exchange - be it file transfers, chats or web surfing.

Until now if you wanted to work in an ad-hoc network with a few peers you had to set most (if not all) information by hand. You agreed on an IP subnet and assigned IP addresses to the hosts. Then you exchanged information for the services you wanted to use. In best cases these were some networking protocols that had inherent discovery features like SMB or AppleTalk. In worst cases someone ran a FTP server and gave the access information like IP address and port to the other users (this is a quite common procedure if Unix users meet somewhere to exchange files, unfortunately). So at best networking was an annoyance, at worst it was a show stopper.

There comes Zero Configuration Networking into the arena (ZeroConf). ZeroConf claims to solve these issues. It describes a set of standards (or proposed standards) that will make networking an experience. It includes how basic addressing settings are negotiated, names are resolved and services discovered on the network.

Today the basis for almost all networking is the Internet Protocol (IP) mostly applied on IEEE 802 networks (including cabled Ethernet and wireless LAN). IP provides the machines with a logical addressing framework on the low-level machine addressing (ARP).

In corporate and managed networks this is not much of a problem. There is usually a server running distributing IP configuration data via the Dynamic Host Configuration Protocol (DHCP). But in ad-hoc networks this infrastructure is usually not available and thus there is a need for a way to agree on IP address information in a peer-to-peer fashion. This is described in the section *Instant Networking*.

Of course the user should not be embarrassed by still having to type that IP address for anything he does on the network. For this reason in the section *Name Resolution* we describe two methods for resolving names of hosts on the local network without a central name server.

After the addressing information has been set the user needs to know which machines offer what kind of services (without having to know the host name where the service can be found). The knowledge about the available services is up to now usually distributed from the administrator to user teaching session – by mouth. What is missing is a general method for discovering services that are currently available. This starts from the problem of finding a printer to more sophisticated problems like automated cluster configuration with a fail-over behavior. In the section *Service Discovery* we will describe four different protocols that fulfill this task.

In the section *Comparison* we will compare the different approaches for name resolution and service discovery in terms of dispersion in the market and technical benefits.

Finally in the *Conclusion* we are going to argue for a specific sets of protocols that suite the needs of ZeroConf and that are in general considered to be *the* protocols of the ZeroConf suite.

2 Instant Networking

In this section we describe two standardized approaches of automatically configuring a host in a network. We are first going to discuss how this information is retrieved on networks where infrastructure is available. We will see that it is clear that it is insensible and in fact in common cases harmful to use that kind of service in an ad-hoc network. This leads then to IPv4 link-local addresses for ad-hoc networks.

2.1 Networking parameters

For two hosts to be able to communicate with each other the basic information they need is addressing information. In Ethernet these are defined with the media access control (MAC) as MAC addresses on the physical and data link layers of the OSI (Open Systems Interconnection) model. These addresses are only valid for machines that are physically connected to the same physical network segment which can only be extended by Ethernet bridging. So another layer is needed for addressing beyond the local network scope.

2.1.1 Internet Protocol

The *Internet Protocol (IP)* is the basis for almost all modern networking. It defines a header for data packets followed by a data blocked, which is formatted depending on the protocol field. In this document we will look at IPv4, version 4 of the IP.

The basic information like addressing is defined in the IP header (see table 1). The source and destination address are of special interest here. They define the origin and the destination of a packet sent over the network. Assume a simple network where every host has exactly one network interface. Then each of these interfaces on the network needs a unique network address. The Time-to-live (TTL) field will later be used to give an alternative interpretation of the link-local notion. The TTL can be considered as a maximum hop-count (in fact it is labeled with this in IPv6). The sending host sets an initial TTL. Every router this packet passes decreases the TTL by one. If the TTL reaches 0 after

<table>
<tr><td>0</td><td>4</td><td>8</td><td>12</td><td>16</td><td>20</td><td>24</td><td>31</td></tr>
<tr><td>Version</td><td>IHL</td><td colspan="2">Type of Service</td><td colspan="4">Length</td></tr>
<tr><td colspan="4">Identification</td><td>Flags</td><td colspan="3">Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td colspan="2">Protocol</td><td colspan="4">Checksum</td></tr>
<tr><td colspan="8">Source IP Address</td></tr>
<tr><td colspan="8">Destination IP Address</td></tr>
<tr><td colspan="8">Options</td></tr>
</table>

Table 1: IP header

decreasing it the packet is dropped and not send any further. This is use to avoid deadlocks of packets floating around the network forever due to misconfigured routers.

For more information about *IP addresses* confer [Pos] and [FLYV93]. If we talk about the IP protocol we now always link to both documents. Some properties that are specifically interesting:

- IP addresses are unique on the local network, although they do not have to be globally unique.
- For the routing decision over which network a specific packet should be sent a subnet mask is needed.
- For easier usage by human beings the Domain Name System (DNS) has been established which provides mappings from names to IP addresses
- To be able to access machines outside the local network gateways are used, gateways may either route traffic for a specific network or act as a default gateway for all traffic that cannot be reached otherwise

The MAC address of a network adapter is fixed and set during the production process and should not be changed afterwards. Since MAC addresses are organized in ranges assigned to specific vendors global uniqueness can be guaranteed (as long as nobody changes his MAC address which is strongly discouraged). As mentioned above the IP protocol uses subnet masks to divide the address range into logical sections to divide the address range into multiple networks. So the IP address is a logical address. It is set for a specific host in a specific network. If the host moves through several networks (for example a laptop) it needs different logical addresses. These addresses are thus dependent on the network a host is operating in.

2.2 Managed Network

A managed network is governed by a single group of administrators that keep the network running and decide on the techniques used. It usually has a topology with some central servers for dynamic host configuration (DHCP) and name resolution (DNS) besides more specific services needed in the environment. In managed networks centralized systems decide on the basics of the configuration of the client machines which includes the IP properties mentioned above. The benefit of this is easy configuration of all clients without touching each. Basically you plug-in a new box and it will be configured automatically through a DHCP server (this applies to the network configuration, more sophisticated setups can go further).

2.2.1 Dynamic Host Configuration Protocol

The *Dynamic Host Configuration Protocol (DHCP)* is defined in [Dro97].

DHCP aims to be a mechanism that allows administrators to have a local configuration policy that is sent from the server to the clients. DHCP clients are Internet hosts that use DHCP to obtain network configuration parameters that have been mentioned above as a requirement and several others. A DHCP server returns these configuration parameters on request by a client. The goal is to avoid manual configuration on the client.

It allows for easy configuration of clients from a central server. In the DHCP server the administrator defines a range of IP addresses that will be used to configure the clients. It also provides information about subnet mask, the default gateway and a DNS server.

Communicating Network Parameters When a client starts a network device that needs configuration it starts sending a broadcast to the local network with a `DHCPDISCOVER` (figures 1 and 2(a)). This packet might be received by multiple servers. To keep it simple we will assume that there is only one DHCP server on the network. This server then determines a configuration for the discovering client and sends a `DHCPOFFER` with these parameters to the client. The client waits for a given time for `DHCPOFFER`s from DHCP servers on the local network. It then chooses one configuration and requests that configuration using a `DHCPREQUEST` packet from that very server. The server acknowledges this request with a `DHCPACK` (figure 2(b)). At this moment the client has all the network configuration data it needs to communicate with other hosts on the network (figure 2(c)). The DHCP server guarantees that no IP address is in use by more than one host at the same time. If a client requests a new IP address later the server will try to supply the client with the network configuration it had last time if at all possible.

Benefits and Drawbacks DHCP allows administrators to have a centralized configuration instance for every client on the network. It is easy to plug-in a new box or to offer "guest networks" for visitors without much hassle.

This centralized approach is also one of the biggest problems. In mobile ad-hoc networks such a centralized instance is not always available. [Dro97] explicitly states that a host should not by default

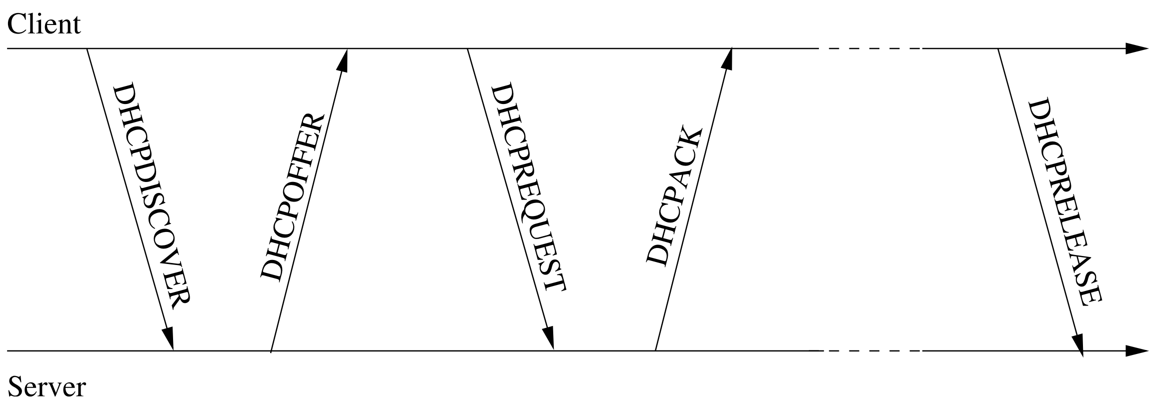
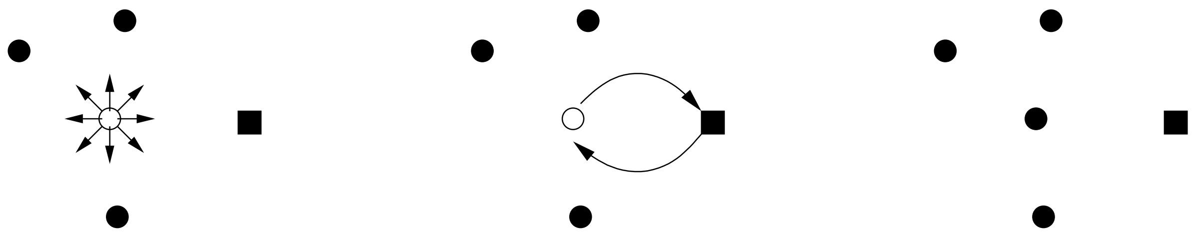


Figure 1: DHCP Protocol Handshake



(a) Client broadcasts DHCPDISCOVER

(b) Negotiation of parameters

(c) Client configured

Figure 2: Gathering network configuration via DHCP in a network. Filled circles mark hosts which are already configured, plain circles are hosts without network configuration data, the filled square is a DHCP server

act as a DHCP server unless it is configured to do so. Since this will rise a couple of problems it is not an option to just enable a DHCP server on every mobile host to negotiate network parameters.

2.3 Ad-hoc Network

We have learned that in a managed network scenario there is not much to do for the user to get connected to the network. DHCP will distribute the needed configuration data to the client which due to this does not need manual network configuration. We also learned that this strategy is not applicable in so-called ad-hoc networks.

Ad-hoc networks are networks that work without a pre-existing infrastructure. In most cases these machines are laptops that form a wireless network without a base station or several hosts interconnected with either a cross-over Ethernet cable or a simple switch. [CAG05] states: "As the Internet Protocol continues to grow in popularity, it becomes increasingly valuable to be able to use familiar IP tools such as FTP not only for global communications, but for local communications as well." This means that the transfer of knowledge and experience gathered with protocols that were used over the Internet for years will help to also improve the local network experience and make it easier to effectively use machines while you go.

Since there is no infrastructure in place there is no central instance like a DHCP server that will manage the network configurations of the hosts. A special approach is needed for this hosts to agree on a valid and plausible network configuration.

2.3.1 Link-Local IPv4 addressing

To solve this issue link-local addressing has been defined in [CAG05].

Local Link The local link defines a local, closed, interconnected set of machines. The machines are on the same local link if:

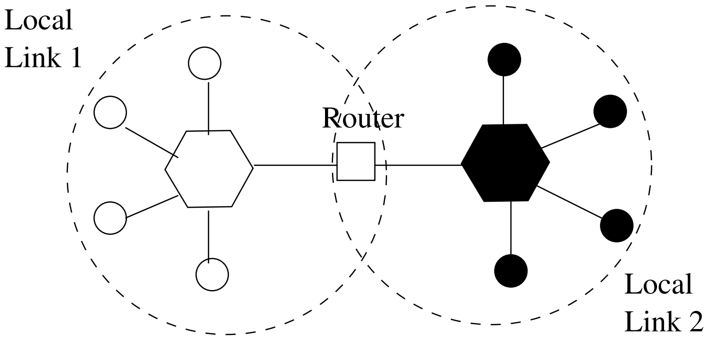


Figure 3: Local-link networks

- any host A from that set sends a packet to any other host B in that set, using unicast, multicast, or broadcast, the entire link-layer package payload arrives unmodified, and
- a broadcast sent over that link by any host from that set of hosts can be received by every other host in that set.

This definition has some implications:

- the link-layer header maybe modified, but not the *payload*
- packets may pass through devices such as repeaters, bridges, hubs or switches
- packets may *not* travel through devices like IP routers that decrement the TTL or otherwise modify the IP header.

As a simple definition one could say that machines are on the same local link if they can send packets to each other with a TTL of 1.

In figure 3 you can see two networks connected via a router (squared box). The one network consists of the plain components (circles marking hosts, the hexagon denotes a switch), the other network consists of the filled items. There are two networks, each forming a local link as marked by the dashed circles. Machines in the local-link 1 will not be able to contact machines in local-link 2 if they only use link-local communication with a TTL of 1 since the router would decrement the TTL counter thus reaching 0 and it would drop the packet.

Auto-configuration in an ad-hoc network Since there is no central managing component the hosts must agree on IP addresses in a peer-to-peer style fashion.

For this purpose a special IPv4 prefix 169.254/16 is registered with the Internet Assigned Numbers Authority (IANA). The first 256 and the last 256 addresses in this range are reserved for future use and must not be selected.

Algorithm 1 Link-Local Address selection

$A \leftarrow \{169.254.1.0, ..., 169.254.254.255\}$
$T \leftarrow \{\}$
repeat
  $ip \leftarrow random\_from\_set(A \backslash T)$
  $T \leftarrow T + ip$
  $R \leftarrow arp\_check\_ip\_taken(ip)$
until $R = false\ or\ T = A$

The algorithm used to determine an IP address on each machine is simple. (see algorithm 1). It first initializes A to be the set of available addresses and T to be the empty set of taken IPs. Now we take a random IP from the set $A \backslash T$ with a uniform random distribution. It then checks if the IP is already taken on the local link by using ARP to query for that IP address. Taking an IP and checking if it is available is done until a valid and not yet taken IP address has been found or until all IPs have been tested. Since link-local addresses are meant to be used in small networks this should rarely happen.

[CAG05] states that with 1300 hosts there is a 98% chance of selecting and unused IP address. After a second try the chance rises to 99.96%. Networks with more than 1300 hosts are in most cases candidates for a managed network anyway.

So we see that IPv4 link-local addresses can be assigned without a central infrastructure and without user interaction which is the basic foundation for simple ad-hoc networking.

3 Name Resolution in Ad-hoc Networks

In a managed environment there is a central Domain Name System (DNS) server that can be queried to transform the human-readable names of networks and hosts into a machine-readable IP address of the host on the network. But in an ad-hoc network there is by definition no central infrastructure and thus no central DNS server.

There are two proposals on how to solve that problem, Link-Local Multicast Name Resolution and Multicast DNS. We are going to describe both with a focus on the latter. We assume that the reader has a basic understanding of DNS and how it works, especially in regard to resource record (RR) types, SRV records and query protocol (confer [Moc87] and [GVE00]).

3.1 Multicast networking

As a prerequisite of both described name resolution protocols we have to give a short introduction to IP multicast networking.

IP multicasting is the transmission of an IP datagram to a ”host group”, a set of zero or more hosts identified by a single IP destination address. A multicast datagram is delivered to all members of its destination host group with the same "best-efforts" reliability as regular unicast IP datagrams, i.e., the datagram is not guaranteed to arrive intact at all members of the destination group or in the same order relative to other datagrams. The extension for Multicast IP networking has been defined in [Dee89].

Multicast addresses are in the designated range between 224.0.0.0 and 239.255.255.255. The addresses are assigned by the IANA which hosts a table of assigned multicast addresses on their website. The range of addresses between 224.0.0.0 and 224.0.0.255, inclusive, is reserved for the use of routing protocols and other low-level topology discovery or maintenance protocols, such as gateway discovery and group membership reporting. Multicast routers should not forward any multicast datagram with destination addresses in this range, regardless of its TTL.

Multicast networking can be understood as a kind of radio: The packets are broadcasted to all the stations on the local network which can decide to tune in on the right multicast addresses (which could be considered the signal frequency in the radio scenario) they want to listen to.

3.2 Link-Local Multicast Name Resolution

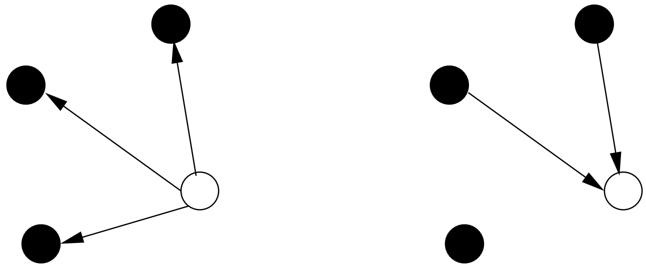
Link-local Multicast Name Resolution (LLMNR) is defined in [ATE05]. LLMNR is based on DNS – but it does not aim to replace it. It uses multicast communication to query names of other hosts on the local-link. It provides the user with name resolution in a peer-to-peer network if there is no DNS server available. LLMNR is meant to be used to resolve single-label names (names which do not include any domain information and thus no dots). Sending all DNS requests via LLMNR would lead to a security risk. Assume someone tries to resolve www.some-bank.com and someone on the local-link answers with his own IP address.

Existing DNS resolver implementations cannot simply be re-used because LLMNR needs some changes of the DNS header by changing the semantics of some flags. The changes affect special name conflict situations that can occur in an unmanaged network if two hosts have the same name. The retrieval recursion where a DNS server would ask further hosts up the hierarchy have been removed. This feature makes no sense in an ad-hoc network since there is only a flat hierarchy of DNS servers where all hosts can be reached directly since the scope is only the local link.

3.3 Multicast DNS

Multicast DNS (mDNS) is defined in [CK05b] (more informations may also be found at [URLa]). As the name suggests mDNS proposes a slight change on how DNS is used – via multicast networking. It does not require changes to the structure of DNS messages as LLMNR does. What it does is to describe what has to be taken care of if DNS responders start sending and answering queries targeted to or originating from a multicast address.

We are going to describe mDNS more detailed than LLMNR since later we will see how it nicely fits together with DNS-based service discovery and why LLMNR does not.



(a) Client multicasts query

(b) Client waits for answers

Figure 4: The plain circle marks the host querying the networks for some RR via mDNS. After a while it has received answers from two of the three hosts

The basic idea is to have a new top-level domain called .local. In this space all names are freely available. They are not assigned to a specific host or person. Usually every host has a name. In a managed network this is usually a fully qualified domain name (FQDN) like laptop.example.com. On the local-link the machine then tries to take the name in the .local domain. For the example this would mean that the laptop would claim laptop.local for itself. As the .local domain is not managed by a central station conflicts may occur. mDNS has methods to resolve these conflicts (see below).

3.3.1 Name Reservation

If a machine joins a network, starts a network device or awakes from sleep mode it needs to claim its host name. As mentioned earlier multiple machines could use the same name. This problem is known from protocols like NetBIOS or AppleTalk where the very same problem could occur. But in real-world usage this rarely happens.

If a mDNS host wants to claim its name it first probes if the name is already taken on the network. For this it sends a query message that asks for any record types assigned to the name it desires. It is assumed that a host should have exclusive ownership of its name and associated records as it could be confusing if one host owned the A record (IPv4 address) for a name and another host the HINFO record (hardware and operating system information). If it receives no answer it repeats the step two more times. If it still got no response the host assumes that the chosen name is available. It then sends an announce message containing all the resource records the host now claims ownership of. If the host gets any answer it has to assume that another host already uses the desired name and it must choose another name.

3.3.2 Querying Information

To query records a host sends a packet to the multicast packet with the desired query. To suppress known answers it places all known records in the answer section of the query. This is done to reduce the traffic imposed on the network. It then waits for a given time for answer packets. Some records (like the query for PTR RR in DNS-SD as we will see later) are not unique and may provoke multiple answers. So the mDNS responder has to wait for some time to collect all answers. Here we also see a major difference to regular DNS: During a regular DNS query the DNS server would respond with an error if the record did not exist (like NXDOMAIN, not existant domain). With mDNS this is not the case. An indication that the service does not exist must be taken from the fact that no answer has been received for the requested record type.

3.3.3 Caching

mDNS does not only send queries via multicast networking, but the answers are also send back as multicast traffic. This allows a station on the network to cache entries that have been requested by other hosts. This is also used to detect conflicts and to fill the known answers section as mentioned above. This reduces the traffic imposed on the network. To prevent stale date being in the cache forever DNS defines a time-to-live (TTL) value for each RR. The recommended TTL for records containing a host name is 120 seconds and 75 minutes for all other records. To avoid problems with hosts and services disconnecting from the network these hosts should send a "Goodbye message". This allows other host to remove the appropriate records from their cache.

4 Dynamic Service Discovery

The number of services offered in networks grows rapidly. New services emerge and old services get replaced. There are new mobile network-enabled devices every day. Almost all modern PDAs and notebooks are equipped with wireless networking technology. Following this observation it is a critical task to give users the possibility to find and use services available on a particular network.

The traditional way was to configure each service explicitly on each host. For example the administrator would define a list of printers and configure each of this list on every machine.

Service discovery protocols enable network devices, applications, and services to find other complementary network devices, applications, and services needed to properly complete specified tasks. In a service discovery environment, services advertise themselves, supplying details about their capabilities and service specific information needed for accessing it.

From the user's point of view, service discovery greatly simplifies the task of finding and utilizing services on a network. From the administrator's point of view, service discovery simplifies the task of building and maintaining a network, especially in regard to adding new devices and services to the network.

In the past there have been application specific protocols to ease the configuration. These methods only worked for a specific service. Examples are the CUPS (Common Unix Printing System) browsing feature. Service discovery aims to provide a unified solution for all services in a network. This reduces the amount of traffic on the network and makes implementation much easier since there is only one protocol to implement.

Some considerations regarding the protocol to use:

- *Directory and peer-to-peer environments*: Should the service discovery use a central directory server that registers services and distributes the lists across the network?
- *Openness*: Is the system based on open standards? Is everybody free to implement and use the protocol?
- *Strictness*: Should the protocol define every aspect from discovering the services up to an abstract way to access the service?
- *System and manufacturer independence*: Should the used protocol work beyond operating system barriers and not for just an eligible group of manufacturers?
- *Existing implementations*: Do implementations exist and are they in real-world usage?

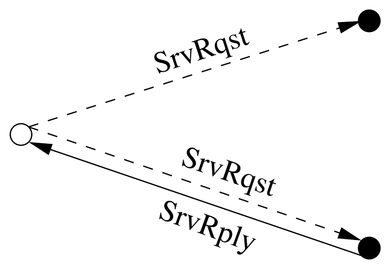
We will present some existing protocols with regard to the given criteria. In the end we will slightly focus on DNS-SD for reasons that will become obvious in sections 5 and 6.

4.1 Universal Plug and Play (UPnP)

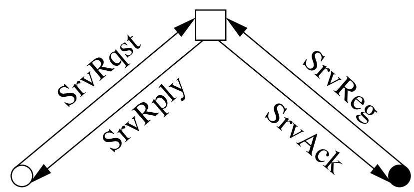
UPnP is being developed by an industry consortium which was founded and is lead by Microsoft. The recommended scope for this technique are home and small office networks (SOHO) where it enables peer-to-peer mechanisms for service discovery. UPnP does not support a central service directory which aggregates information about available services. This renders it useless for bigger networks (the traffic needed for inter-peer communication would cause too much traffic on the network). The consortium says UPnP is open. In fact you have to become a member of the UPnP Forum to claim that your products are UPnP-enabled. The steering committee is under control of Microsoft and the statutes grant special privileges to them. UPnP is very strict. It does not only define the service discovery process, but it also gives detailed description on how services have to be used. It defines detailed protocols for each service that UPnP offers. It has to be approved by the UPnP committee. UPnP can be considered monolithic because of this. A benefit is that vendors can count on specific abilities if a service has been discovered and that there is no ambiguity while talking to the service. On the other hand it takes a long time to get a new service UPnP-ready.

4.2 Jini

Jini is an architecture to federate groups of devices and services to a single, dynamic distributed system. Although it states that these connected devices and services form an ad-hoc network it still needs a central lookup server which denies the applicability of our ad-hoc definition. A more detailed introduction to Jini can be found in [Mic99].



(a) SLP without DA



(b) SLP with DA

Figure 5: Service discovery 5(a) without DA, 5(b) with DA, • SA, ○ UA, □ DA, – Unicast, - - Multicast

Each Jini device has to run a Java Virtual Machine (JVM). Jini consists of a specified API that can be used to participate in the network. The *Lookup Table* on the lookup server may not only contain pointers to information about the available services but also Java binary objects which can be considered as driver in the Jini system. They may implement any communication method the Java system is capable of for example resembling a proprietary protocol. These binary objects are sent over the network and executed on the machine that wants access to the service. This is the best and the worst thing about Jini at the same time. While it allows for really easy access to an arbitrary service it may also yield a security risk to the system since code from foreign machines gets executed in the local system.

Devices and applications start a process called *Discovery and Join* on startup where they will try to find the lookup server and place their available services into the Lookup Table. To find available services in the network a *Lookup* for the service is done on the lookup server.

The system is quite heavy meaning that it needs a JVM running on every single machine that wants to join the network and that there is a complex software infrastructure needed. It does not just handle the discovery of services but it also handles how a specific service is accessed and may even supply drivers needed for the usage. Since the Java binary objects are in principle portable between systems this avoids software installations per system. The Jini specification does not only include the infrastructure but also a programming model (since it is tied to the Java platform). This makes Jini strict in the sense that it defines not only the discovery but also means by which services can be accessed from an application. Jini was specified by Sun. They still more or less control the project but in March 2005 they changed their licensing model to build up a Jini community and ride the Open Source wave. All Sun contributions are now released under an Open Source license allowing everybody to use the specs and code freely for any kind of project.

4.3 Service Location Protocol (SLP)

SLP is defined in [GPVD99]. We will only discuss SLP version 2 (SLPv2). SLP allows computers and other devices to find services on the local network. It is a decentralized, lightweight and extensible protocol and it scales from small ad-hoc networks to large corporate networks.

SLP defines three different roles for devices. The *User Agents (UA)* are devices that search for services on the network (for example a desktop machine looking for printers); *Service Agents (SA)* announce one or more services on the network (in the example that would be the printer announcing its print ability) and *Directory Agents (DA)* that cache a list of available services. There may be any number of DAs (also none) on the network which allows SLP to scale from small ad-hoc and SOHO to large enterprise networks.

SLP supports service browsing for a number of attributes that can be logically concatenated (AND, OR) and queried with several comparators ($=$, $<$, $<=$, $>$, $>=$). This way the client can choose the best candidate for the requested service available on the network.

Figure 5 shows two service discovery processes. 5(a) shows the procedure if there is no DA available, for instance in an ad-hoc network. It sends a multicast query SrvRqst to the network asking for a specific set of attributes. All applicable SA answer with a SrvRply message with the details of the service they are offering. 5(b) shows what happens if there is a DA. First the SA registers with the DA by sending a SrvReg to the DA. The DA confirms this registration with a RegAck message. After that the service can be found. The UA sends all queries to the DA if there is one via Unicast. The DA answers with an appropriate set of services which registered with the DA. Service registrations have to be renewed with the DA (lease concept) to avoid invalid cache entries if a service did not unregister itself properly before shutdown (for example because of power failure).

SLP does only describe the service discovery itself and not how to access the services that can be found. It just defines some well-known service templates to have some kind of standard for similar services. The IANA keeps track of these service templates. See [GPK99] for details.

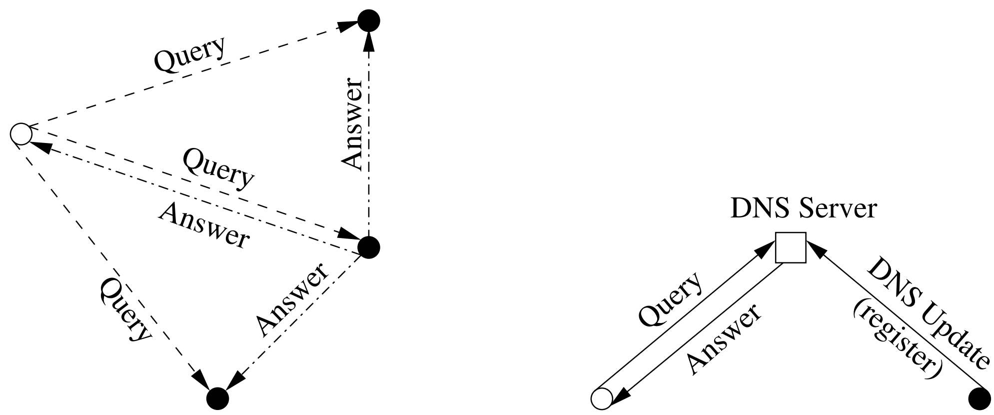
There exist a couple of implementations of SLP. OpenSLP is an Open Source implementation maintained by Novell. Novell also used SLP in their NetWare products version 5 to 7. Another implementation was written by Sun Microsystems. They are using SLP as a service discovery only fall back in Jini. Since SLP is an open standard everybody is free to implement and use it in any environment. There is no single company steering the protocol development.

4.4 DNS-based Service Discovery (DNS-SD)

DNS-SD is the newest approach to service discovery. It was developed by Apple in the past few years and an Internet draft has been published in June 2005 (confer [CK05a] and [URLb]).

4.4.1 Protocol Overview

DNS-SD describes a convention for naming and structuring DNS resource records (RRs). As the draft states given a type of service that a client is looking for, and a domain in which the client is looking for that service, this convention allows clients to discover a list of named instances of that desired service, using only standard DNS queries.



(a) DNS-SD over mDNS (b) DNS-SD with DNS server

Figure 6: Service discovery 6(a) on an ad-hoc network, 6(b) on a managed network, • Service, ○ User station, □ DNS Server, – Unicast, - - Multicast Query, - · - Multicast Answer

DNS-SD can work over mDNS, which allows it to work on the local-link. This is a desired feature to be usable in an ad-hoc network. DNS-SD can also use the conventional DNS which works as a directory server where requests can be send to. This allows DNS-SD to scale to large corporate and enterprise networks (see figure 6).

To discover a service a host first sends a request for a PTR RR for the desired service and domain (a PTR record is a pointer to additional information, if a PTR RR exists for the queried resource then this points to some other location in the domain space). For example to query for printers the host might query for _ipp._tcp.example.com where _ipp._tcp determines the service (Internet Printing Protocol (IPP) over TCP) in the domain example.com. If used with mDNS the domain must be .local (see section 3.3 on page 12). The result is a list of zero or more PTR RR to service instance names. A service instance name is of the form <Instance> . <Service> . <Domain> where <Instance> is an arbitrary precomposed UTF-8-encoded text, <Service> is the service type queried for and <Domain> is the domain queried in.

The returned list is specific to the service requested by the user (for example a printer). The list of instance names is then presented to the user who decides which service to use.

To get the needed information to connect to the service three more queries are needed. First the SRV RR for the instance name is queried. SRV RR are defined in [GVE00]. They describe a naming convention for resource records of the form _Service._Proto.Domain (see the example for the PTR query above), where Service is a standardized service identifier and Proto is either UDP or TCP, and a resource record that specifies information where the service can be accessed (for example port number and target host).

Secondly a TXT RR (custom text) is queried for the instance name. This TXT RR may carry additional information that is needed to access a given service. These settings are given as name/value pairs of strings. For example old protocol like the LPR printing protocol do not include feature negotiation. So a client cannot query via the protocol itself which printing queues are available or what features these printers have (color or b/w printer, data format etc.). In this case the needed information is carried in the TXT RR.

At last the host name mentioned in the SRV RR is queried (query for A or AAAA RR). Since any host may serve for example _http._tcp.example.com this information is stored in the SRV RR.

4.4.2 DNS-SD in Ad-hoc Networks

Until now we have only mentioned the usage of DNS-SD in a managed network with a central DNS server that acts as a directory of services. Earlier we have seen that for ad-hoc networks a multicast name resolution approach is necessary for name resolution when there is no infrastructure in place (see section 3). In section 3.3 we have described a method to achieve this goal by using only standard DNS queries. Since DNS-SD still only involves standard DNS queries it makes sense to just use DNS-SD over mDNS thus allowing service discovery in an ad-hoc network via Multicast DNS (see figure 6(a)).

4.4.3 Traffic Reduction

As we have seen above querying a given service via DNS may involve quite a few DNS queries. If there are numerous stations on a network this might easily lead to a high volume of traffic. To avoid this DNS-SD features several mechanisms to lower the traffic caused to the network.

DNS Additional Record Generation When a DNS server (or mDNS responders) answers a query for a PTR record for a specific service it creates additional records in the Additional Section of the DNS Message. These additional records where typically not requested by the client but the sender of the response has reasonable grounds to expect that the client might request them shortly.

As we have seen in the Protocol Overview (see page 17) when a client requested a PTR record it is very likely that it will request at least once for SRV and TXT RR for a given instance name and for the appropriate address records. To lower the average traffic on the network the responder adds all SRV and TXT records for the instance names in the PTR resource data and all address records (A or AAAA RR) mentioned in any SRV records. Also if a specific SRV RR is queried it immediately sends all address records whose names appear in the SRV RR in the Additional Section. This mechanism is especially powerful if there is only one record in the PTR query response.

Caching and Known Answer Suppression We have seen that DNS-SD works just fine with mDNS in an ad-hoc network. mDNS uses multicast to send response packets for queries. This allows the clients for caching the answers that other clients requested. If a client now needs the information that another client requested and this has not been purged yet (records have a specified time-to-live to avoid dead entries in the cache) some queries may even unnecessary since the information is already there.

In some cases only part of the information needed will be available from the cache. To avoid re-sending of all these records again (or sending answers for this known information at all) the querying client puts all the relevant known records into the Answer Section of the DNS Message for the query. This is actually a feature of mDNS and not DNS-SD but one can imagine that this will be especially useful for DNS-SD.

4.4.4 Service Registration

To announce a service in an ad-hoc network no registration is needed. Since the mDNS responder will answer for all records it is authoritative for it will answer all queries for its service records anyway.

In a managed network there are two ways to manage the service directory with the DNS server. In the static case the administrator explicitly names all service instances in the domain's zone file. In this scenario the service registration is done manually. Problems can arise if the machine goes down (for example for maintenance) or gets a new IP address. For some time invalid information may be stored in the DNS server. Another approach which solves this problem is using Dynamic DNS Updates (confer [VTRB97]). In this case clients update the DNS server with their service information when they go live and order the removal of this information if they go out of service (see figure 6(b). To avoid problems when machines disappear suddenly (for example during a power failure) this has to be extended by Dynamic DNS Update Leases, which means that records have a specified time-to-live and if not re-newed by that time they are deleted automatically (confer [SCK05]).

5 Comparison

In this section we are going to compare the different approaches for name resolution in ad-hoc networks (LLMNR and mDNS) and service discovery (UPnP, Jini, SLP and DNS-SD). We will give a brief overview of some of the strengths and weaknesses and how widespread it is in the market today.

There is no need to compare anything for the instant networking section since there is only one standard (IPv4 local-link addressing) which is widely accepted and used on a variety of systems including Unix/Linux, Windows and Apple Macintosh for years. Before there was only DHCP for managed networks to configure machines automatically on the network. Now there is a method to agree on these settings in a peer-to-peer style network.

5.1 Name Resolution

In section 3 we presented two methods to resolve names in an ad-hoc network, LLMNR and mDNS.

LLMNR has gone further in the standardization process of the IETF but it was controversial because it had a number of issues that have to be discussed in more detail and the number of rewrites that have occurred. After finishing the last call it was not released but scheduled for another editing cycle.

One of the problems that were mentioned is that LLMNR does explicitly not use the .local domain for name lookup as mDNS does. Since mDNS is already being deployed in the market with Apple Bonjour and with several printers from major manufacturers this might lead to a problem since in LLMNR only single-label names (no domain part in the name and hence no dots) and so it would send questions for .local (which users may already have encountered with mDNS products in the market) would lead to a lot of lookups to the root DNS servers.

Another problem is that one cannot distinguish if the answer from a lookup comes from another LLMNR host on the local-link or an authoritative name server. This can be considered a security flaw since it allows spoofing of host names. mDNS solves this problem by using the .local domain. Because of this LLMNR can be considered to be a DNS extension while mDNS is more a different usage method of DNS.

LLMNR has also some inherent differences to mDNS that make it incompatible with DNS-SD. The LLMNR FAQ (confer [Abo]) states some differences between LLMNR and Bonjour (which is an implementation by Apple of both, mDNS and DNS-SD). These differences cause trouble if DNS-SD would be used together with LLMNR. We take a closer look to the differences between mDNS and LLMNR:

- Bonjour allows multiple questions to be asked within a single query; LLMNR does not (can cause high network load)
- Bonjour allows responses to be sent to a multicast address; LLMNR only allows unicast responses (no caching on other notes on the network, causes higher load)
- Bonjour allows the TC bit to be set within queries; LLMNR does not (mDNS allows queries and answers to be split over several messages while LLMNR does not, but in some cases DNS-SD answers may exceed the size of a single DNS Message)

While it is possible that LLMNR and mDNS are used at the same time on the same subnet (they have different multicast addresses and port numbers) this could lead to confusion since you wouldn't know the name was resolved.

There are several implementations of mDNS today, in most cases in union with a DNS-SD implementation (since this is one of the most prominent uses). Apple's implementation is called Bonjour and can be considered to be the reference implementation. It is available for a variety of systems including Mac OS, Windows and Linux. A new implementation is Avahi which gains some momentum. It seems likely that it will become the default implementation used by the Gnome desktop. As mentioned mDNS is already deployed with a number of products so it has a substantial share in the market already. [Abo] mentions two existing LLMNR implementations, but in fact LLMNR has not appeared "in the wild" until today.

5.2 Service Discovery

We have discussed four different service discovery protocols, see table 2 for an overview.

	Jini	UPnP	SLPv2	mDNS/DNS-SD
Developer	Sun Microsystems	Microsoft	IETF	Apple
License	Open Source	open (only for members)	Open Source	Open Source
Programming Language	Java	independent	independent	independent
Implementation (example)	Sun Porter Project	Windows	OpenSLP	Bonjour, Avahi
Attributes searchable	yes	no	yes	no (subtypes only)
Directory Support	Lookup Table required	not supported	yes (optional)	yes (optional)
Service Announcement	Discovery/Join protocol	Advertisement (ssdp:alive)	SrvReg with DA	Multicast Announce or DNS Update
Service Registration Lifetime	Leasing	CACHE-CONTROL header in alive message	Lifetime in service registration	TTL for RR
Access to Service	Service proxy object based in Java RMI	Invoking Action via SOAP, query for state	Service type (protocol, port)	Service type (protocol, port)

Table 2: Summary of major service discovery protocols

Until today we do not see service discovery in every day computer usage but the demand is growing. There have been some specific solutions for an isolated software like the browsing feature for the Common Unix Printing System (CUPS).

Another reason for this might be that there have been other problems that had to be solved on the Desktop (where service discovery is particularly useful) before there was a real need for service discovery. Today we see progress on all major desktops like Mac OS, Gnome and Windows. If there were a single standard this could become extremely powerful.

Nowadays the desktop metaphor is somewhat popular. Users are used to browse their files with a graphical front end. They can navigate through their files which allows them to open such without knowing the explicit location as a path but more with a spatial feeling. Service discovery extends this to different kinds of network services. Different applications that allow some kind of information exchange can easily search for compatible services and present the user with a list he can choose from – be it a collaborative text editor during a meeting, a printer in a different location or just a chat in the next Internet cafe.

Jini We have seen that *Jini* is quite heavy in the sense that it is dependent on the Java platform. This way only hosts that have a Java Virtual Machine installed can join a Jini network. Because of this it seems to be more suited for specific environments like corporate and enterprise networks where this dependency can be solved by a simple "install policy". It may be handy in some situations in an ad-hoc network but since Java is still not installed by default on most systems it is too likely that users trying to communicate in an ad-hoc network fail because one of the users does not have Java (or Jini, which is even more likely) installed.

UPnP *UPnP* has been authored by Microsoft and is now governed by a UPnP forum with strict rules for membership. Because of the rather strict licensing and membership policies it is unlikely that we see free implementations of UPnP in the near future. UPnP has been implemented in Windows XP for example. One of the most popular uses is the ability to punch holes in firewalls or to forward ports from a router appliance to a host machine. Time has to show how quickly UPnP can react to new services and if the tight standardization is speaking for or against UPnP. But until UPnP spreads to more platforms it is only an isolated application waiting to be freed and not a tool to work in an ad-hoc network with anybody you want.

The absence of a possibility to have a central service directory might become a problem for corporate networks because of multicast storms if many hosts are looking for services at the same time.

SLP With *SLP* as an open standard there was a chance that service discovery could be widely deployed in the network. But as we see today this has not happened or at least it did not reach the critical mass of market penetration.

SLP version 1 had a couple of short comings that were fixed with SLP version 2. UPnP was designed about at the time when the SLPv2 RFC was written. There were a couple of considerations that made the UPnP designers to go with the Simple Service Discovery Protocol (SSDP) instead of SLP. One of the problems they had with SLP was that it required a lot of work to be done from scratch that already existed. SLP defines a whole new communication protocol, a syntax for service descriptions and a directory protocol – things that have already been addressed by older standards like HTTP, XML and LDAP.

Several software companies included SLP in their products. Novell added support for SLP to their NetWare products. They are also the maintainers of OpenSLP, a free implementation of SLP. Suse (also a Novell product now) had support for SLP since version 9.1 of their Linux distribution. Mac OS also supports SLP with their Network Services Location Manager.

It seems today that SLP is at a dead end.

DNS-SD *DNS-SD* is a light-weight protocol that uses the well-known DNS for service discovery. Usually we see mDNS/DNS-SD implementations in one piece of software.

In an ad-hoc network it can be combined with mDNS to work without any additional infrastructure. In a managed environment a DNS server can be used as a central service directory. Virtually every managed network has its own DNS server. Adding some static entries for important services to the appropriate zone files is very simple. And also a dynamic update mechanism is available with DNS Update.

Several companies are pushing products with support for mDNS/DNS-SD into the market. Apple mention half a dozen printer manufacturers on their website. Implementations for the major desktop systems like Mac OS, Linux and Windows exist so it already gained some importance in the market.

One of the major benefits of DNS-SD is that it recycles existing knowledge. There are already many administrators that are familiar with DNS servers – so setting up a service directory is just the easy task of configuring the name server.

DNS-SD is simpler than any other protocol described, but it also leaves some things open. The additional attributes that are required by a service are set in the additional TXT record as name/value pairs. But the names of parameters are not defined. So two printer manufacturers might use two totally different parameter sets for the same settings. There is a presumption that some kind of de facto standard will arise. There have also been some more formal approaches. For example Apple have released [App05] that describes how printers should be accessed using mDNS/DNS-SD which should prevent the very problem of balkanization.

DNS-SD defines the names of the devices (the instance part of the instance name) as UTF-8 strings. These strings are presented to the user hiding all the little details needed for the service configuration. No more entering of host names (or even worse: IP addresses), port numbers and other service specific details.

6 Conclusion

In this seminar paper we presented all major components needed for a Zero Configuration Networking (confer [URLc] or [SC05]) – IP address configuration, name resolution and service discovery.

We will now argue for the set of IPv4 local-link addressing, mDNS and DNS-SD as the set of protocols favored by ZeroConf – which is what ZeroConf de facto is about. Later we will see that this even scales well to corporate managed networks.

For IP address configuration there is a widely accepted standard defined by an RFC so there is not much to argue.

We are now going to the top of the stack for reasons that will become obvious soon. We have shown four different methods of service discovery. Jini is heavy and Java-dependent, thus it would not fit a protocol suite of open protocols for ZeroConf. UPnP is governed by a strict consortium of companies and defines query strictly how to access services, and not only how to find them. So this does not suite a lightweight protocol suite. SLP is an open standard that might have done the job. But it defines a whole new communication protocol. After being out there for almost ten years it has still not reached the critical mass needed to be widely usable. DNS-SD is a rather young approach but there are already a variety of platforms supporting it. It is very simple and clean and implementations for all major systems already exist and it is an open standard. Especially in the Open Source world it is gaining some momentum. It does only define the basics about how to discover services, not how to use them so existing applications do not need to be restructured to fit the needs of a specific protocol (although some applications may benefit from doing so and new applications exploiting the new features available may seem appropriate in some cases) but they can just be announced as-is via DNS-SD. These are the reasons why DNS-SD is chosen as the service discovery protocol for ZeroConf by most vendors.

For name resolution there are two approaches. We have shown that there are a few problems with LLMNR, especially in conjunction with DNS-SD. Since we already argued that it makes sense to use DNS-SD for ZeroConf environments we have to go with mDNS – not just because it looks like the superior solution but also because we need it to make use of DNS-SD in local area and ad-hoc networks.

So with this combination of protocols for ZeroConf – IPv4 local-link addressing to assign IP addresses in ad-hoc networks, mDNS send and receive DNS queries/answers on the local link to resolve names and for use with DNS-SD, the chosen service discovery protocol – we have a complete protocol suite to solve the problems outlined in this seminar paper.

References

[Abo] Bernard Aboba. LLMNR Frequently Asked Questions (FAQ). `http://www.drizzle.com/~aboba/DNSEXT/llmnrfaq.html`.

[App05] Apple Computer, Inc. Bonjour Printing Specification. White Paper, April 2005.

[ATE05] Bernard Aboba, Dave Thaler, and Levon Esibov. Linklocal Multicast Name Resolution (LLMNR). Internet-Draft, Internet Engineering Task Force, October 2005.

[CAG05] Stuart Cheshire, Bernard Aboba, and Erik Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927, Internet Engineering Task Force, May 2005.

[CK05a] Stuart Cheshire and Marc Krochmal. DNS-Based Service Discovery. Internet-Draft, Internet Engineering Task Force, June 2005.

[CK05b] Stuart Cheshire and Marc Krochmal. Multicast DNS. Internet-Draft, Internet Engineering Task Force, June 2005.

[Dee89] Steve Deering. Host Extensions for IP Multicasting. RFC 1112, Internet Engineering Task Force, August 1989.

[Dro97] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, Internet Engineering Task Force, March 1997.

[FLYV93] Vince Fuller, Tony Li, Jessica (Jie Yun) Yu, and Kannan Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519, Internet Engineering Task Force, September 1993.

[GPK99] Erik Guttman, Charles Perkins, and James Kempf. Service Templates and Service: Schemes. RFC 2609, Internet Engineering Task Force, June 1999.

[GPVD99] Erik Guttman, Charles Perkins, John Veizades, and Michael Day. Service Location Protocol, Version 2. RFC 2608, Internet Engineering Task Force, June 1999.

[GVE00] Arnt Gulbrandsen, Paul Vixie, and Levon Esibov. A DNS RR for specifying the location of services (DNS SRV). RFC 2782, Internet Engineering Task Force, February 2000.

[Mic99] Sun Microsystems. Why Jini Technology Now? White paper, Sun Microsystems, January 1999.

[Moc87] R. Mockapetris. Domain Names - Implementation and specification. RFC 1035, Internet Engineering Task Force, November 1987.

[Pos] Jon Postel. Internet Protocol, Protocol Specification. Technical report, Internet Engineering Task Force.

[SC05] Daniel H. Steinberg and Stuart Cheshire. *Zero Configuration Networking*. O'Reilly, December (est.) 2005. Not yet released.

[SCK05] Kiren Sekar, Stuart Cheshire, and Marc Krochmal. Dynamic DNS Update Leases. Internet-Draft, Internet Engineering Task Force, June 2005.

[URLa] `http://www.multicastdns.org`.

[URLb] `http://www.dns-sd.org`.

[URLc] `http://www.zeroconf.org`.

[VTRB97] Paul Vixie, Susan Thomson, Yakov Rekhter, and Jim Bound. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136, Internet Engineering Task Force, April 1997.